ADRAR FORMATION

Déploiement automatisé de GLPI via Ansible

MARAVAL Liam
07/10/2024

Ce document sera décomposé en plusieurs chapitres :

1.Contexte : Définition des besoins ainsi que le choix des solutions en réponse à la demande client.
2.Configuration technique : Procédure technique de mise en place des solutions
3.Conclusion

Table des matières

1. Contexte

1.1 Demandes du client

Nous sommes contactés par l'ADRAR afin de mettre en place une solution de déploiement de configuration automatisé.
Cette mise en place leurs permettra de simplifier leurs futurs déploiements via de l'automatisation.

Par ailleurs, l'ADRAR souhaitant déporter ces serveurs OnPrem vers une solution PaaS, nous sommes également en charge de déployer un outil de gestions de parc via la solution d'automatisation retenue sur le datacenter de leur hébergeur.

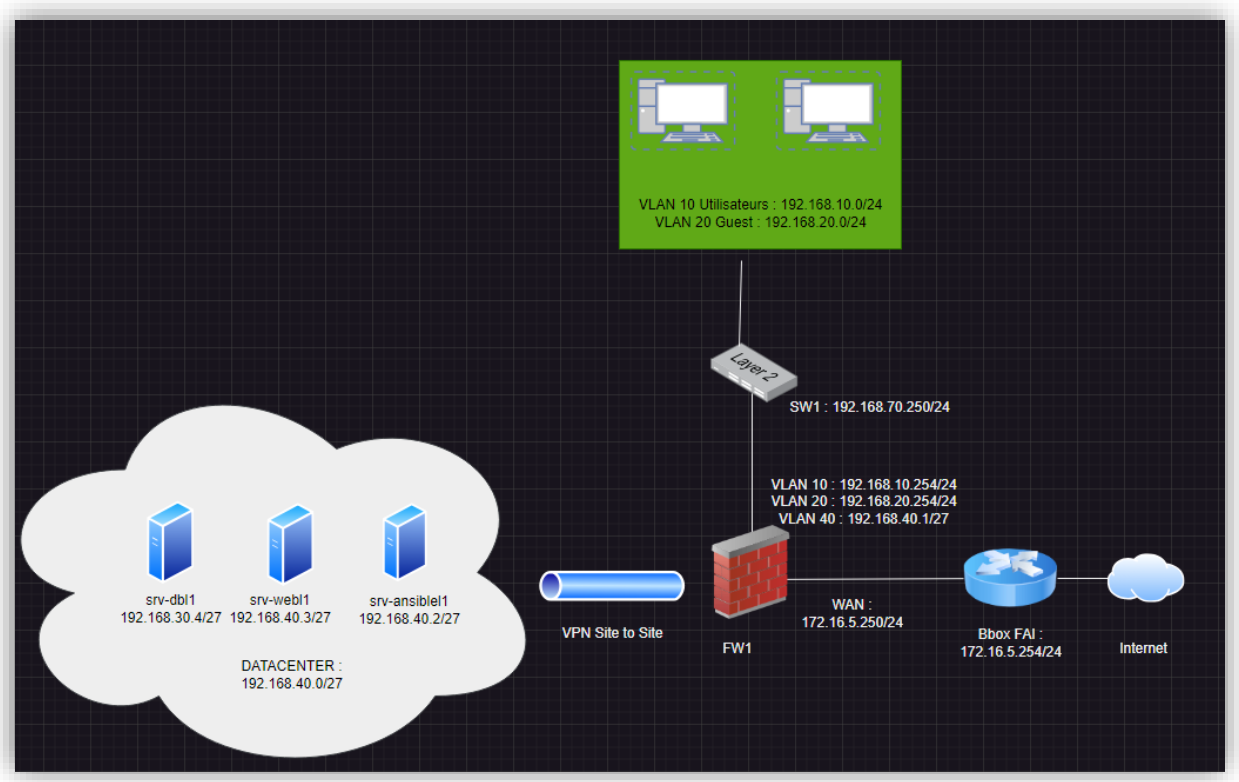
L'outil de gestion de parc demandé est GLPI

Voici le cahier des charges défini avec l'ADRAR :

- Installation et configuration des services d'automatisation
- Déploiement d'un serveur LAMP
- Déploiement de GLPI
- Sécurisation des serveurs en cas d'échec de connexion SSH ainsi qu'a GLPI

1.2 Evolution de l'infrastructure

Pour mieux comprendre la mise en place de la nouvelle infrastructure, veuillez-vous référer au nouveau schéma effectué :



Afin de répondre à la demande de l'ADRAR, nous allons implémenter un serveur Ansible.
Celui-ci sera en charge de déployer les configurations sur les serveur clients.

De plus, à l'avenir Ansible sera utile pour maintenir des configurations identiques entres les équipements et faciliter leurs déploiements.
Il peut aussi être utilisé pour des déploiements de VM, déploiement d'OS, MAJ de firmware d'équipements réseau, configuration d'équipement réseau ect…

Dans notre cas, nous l'utiliseront pour déployer la base de données sur le serveur **« srv-dbl1 »** et le serveur GLPI sur **« srv-webl1 »**

1.3 Choix de la solution

Nous avons opté pour une solution d'automatisation via Ansible et voici les raisons qui ont motivé notre choix :

- Solution simple d'utilisation sans agents à déployer
- Fichier de configuration en YAML qui est un langage accessible
- Grande communauté
- Nombreux modules disponibles
- Flexible et évolutif

Concernant les couts, Ansible est opensource.
Cependant une version payante est disponible pour disposer d'un support, d'une assistance pour les upgrade ect..
Le cout de cette version est estimé entre 5000$ et 10.000$/an en fonction du nombre d'équipement à manager.
Le temps de déploiement de la solution, quant à lui, est estimé à quatre jours.

1.4 Sécurisation des accès

Afin de sécuriser notre solution, nous allons mettre en place les configurations suivantes :

- Administration via un compte dédié
- Gestion des mots de passe via le Vault Ansible qui permet de stocker et chiffrer des données sensibles. Dans notre cas, nous utiliserons un fichier chiffré qui contiendra les variables sensibles (nom d'utilisateur et mot de passe)
- L'authentification depuis le serveur Ansible vers les serveurs clients se fera via clé SSH
- Pour une sécurisation accrue, nous allons isoler Ansible dans un Virtual Environment Python. Ce processus nous permettra notamment une isolation de dépendance entre chaque projet, des mises à jour modulaires sans impacter le système ect…

Ansible utilisant SSH cela garantis également des échanges sécurisés.

Concernant la sécurisation des connexions SSH pour les serveurs clients Ansible nous allons mettre en place **Fail2ban**.
Celui-ci servira à définir des règles en cas d'échec de connexion SSH via un user/password erroné.
Dans notre cas, voici les règles définies :

- Filtrage sur le port SSH
- Tentative de connexion en échec max définie à 2 en l'espace de 5min
- Suite à ces échec l'IP sera banni définitivement et devra être autorisé manuellement
- Afin de ne pas bloquer des services interne au serveur, après une mauvaise manipulation par exemple, ces règles seront ignorées sur l'interface IP du serveur.

Nous utiliserons également Fail2ban pour sécuriser notre application GLPI en cas d'échec de connexion, en suivant le même principe que pour le SSH.

2. Configuration technique

Afin de faciliter la compréhension de cette procédure voici quelques termes à connaitre :

- **Control Node** (aussi parfois appelé **Node Manager**) : c'est le serveur Ansible.
- **Node** : C'est un serveur clients Ansible
- **Roles** : C'est un ensemble structuré d'actions à effectuer afin d'arriver à un résultat donné (Déployer un serveur Web par exemple)
- **Task** : C'est une action à effectuer dans un rôle (Installer le rôle Apache)
- **Playbook** : Sert d'intermédiaire entre les rôle et l'inventory. Il définit les rôles à lancer et sur quelles nodes les appliquer.
- **Inventory** : C'est un fichier qui va contenir tous les Nodes, qu'il est aussi possible de regrouper par groupes.
- **Had-hoc** : C'est une commande ou tache unique exécutée via Ansible sans Playbook

Voici maintenant les prérequis de mise en place de notre solution :

- Serveur installé avec distribution Debian 12 dans notre cas
- Configuration IP et DNS effectué sur tous les serveurs de l'infrastructure
- Service SSH-server et Python installé sur les serveurs.

2.1 Configuration des nodes

Dans cette partie nous allons effectuer quelques actions nécessaires pour nos déploiements :

- Créer le virtual environment Python
- Créer le fichier inventory
- Créer un utilisateur dédié pour Ansible sur les nodes appartenant au groupe « **sudo »** afin de ne pas utiliser le compte **« root »**
- Générer nos clé SSH et déployer la clé publique sur nos nodes
- Créer notre arborescence pour le déploiement de nos rôles.

Nous commençons par créer le venv (virtual environment) puis nous l'activons :

```
python3 –m venv ansible
source ansible/bin/activate
```

Nous installons ensuite ansible via la commande :

```
pip install ansible
```

```
(ansible) user-ansible@srv-ansible11:~/ansible$ pip install ansible
Collecting ansible
  Downloading ansible-11.1.0-py3-none-any.whl (51.4 MB)
     ---------------------------------------- 51.4/51.4 MB 11.3 MB/s eta 0:00:00
Collecting ansible-core~=2.18.1
  Downloading ansible_core-2.18.1-py3-none-any.whl (2.2 MB)
     ---------------------------------------- 2.2/2.2 MB 15.3 MB/s eta 0:00:00
Collecting jinja2>=3.0.0
  Downloading jinja2-3.1.5-py3-none-any.whl (134 kB)
     ---------------------------------------- 134.6/134.6 kB 13.7 MB/s eta 0:00:00
Collecting PyYAML>=5.1
  Downloading PyYAML-6.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (762 kB)
     ---------------------------------------- 763.0/763.0 kB 16.0 MB/s eta 0:00:00
Collecting cryptography
  Downloading cryptography-44.0.0-cp39-abi3-manylinux_2_28_x86_64.whl (4.2 MB)
     ---------------------------------------- 4.2/4.2 MB 15.2 MB/s eta 0:00:00
Collecting packaging
  Downloading packaging-24.2-py3-none-any.whl (65 kB)
     ---------------------------------------- 65.5/65.5 kB 32.9 MB/s eta 0:00:00
Collecting resolvelib<1.1.0,>=0.5.3
  Downloading resolvelib-1.0.1-py2.py3-none-any.whl (17 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-3.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl (23 kB)
Collecting cffi>=1.12
  Downloading cffi-1.17.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64
.whl (467 kB)
     ---------------------------------------- 467.2/467.2 kB 14.9 MB/s eta 0:00:00
Collecting pycparser
  Downloading pycparser-2.22-py3-none-any.whl (117 kB)
     ---------------------------------------- 117.6/117.6 kB 32.2 MB/s eta 0:00:00
Installing collected packages: resolvelib, PyYAML, pycparser, packaging, MarkupS
afe, jinja2, cffi, cryptography, ansible-core, ansible
```

Nous vérifions que l'installation c'est bien effectuée :

```
Some actions do not make sense in Ad-Hoc (include, meta, etc)
(ansible) user-ansible@srv-ansible11:~$ ansible --version
ansible [core 2.18.1]
  config file = None
  configured module search path = ['/home/user-ansible/.ansible/plugins/modules'
, '/usr/share/ansible/plugins/modules']
  ansible python module location = /home/user-ansible/ansible/lib/python3.11/sit
e-packages/ansible
  ansible collection location = /home/user-ansible/.ansible/collections:/usr/sha
re/ansible/collections
  executable location = /home/user-ansible/ansible/bin/ansible
  python version = 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0] (/home/user
-ansible/ansible/bin/python3)
  jinja version = 3.1.5
  libyaml = True
```

Nous allons maintenant créer notre fichier **« inventory.ini »** :

```
nano inventory.ini
```

```
  GNU nano 7.2                    ansible/inventory.ini
[web]
srv-webll

[db]
srv-dbll
```

Dans le fichier ci-dessus, nous avons défini nos groupes via des **[]** et sous les groupes nous trouvons nos nodes.

Maintenant nous allons déployer notre 1ere configuration en mode **had-hoc** pour créer un utilisateur **« user-ansible »** sur nos nodes.

Pour cela nous commençons par créer un hash de mot de passe via la commande suivant :

```
ansible localhost -i inventaire.ini -m debug -a
"msg={{'Votre_MDP'|password_hash('sha512','secretsalt')}}"
```

Ci-dessus nous avons défini les paramètres suivants :

- **« localhost »** pour lancer la commande sur le Nodes Manager
- Le fichier **« inventory.ini »** qui permet de cibler les nodes
- Le module à utiliser qui est **« debug »**
- Le mot de passe à hacher via sha512 ainsi que **« secretsalt »** qui va rajouter une donnée aléatoire dans le hash

Voici le retour de la commande en **« SUCCESS »** :

```
(ansible) user-ansible@srv-ansiblell:~/ansible$ ansible localhost -i inventaire.
ini -m debug -a "msg={{'stmhjtn'|password_hash('sha512','secretsalt')}}"
[WARNING]: Unable to parse /home/user-ansible/ansible/inventaire.ini as an
inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
localhost | SUCCESS => {
    "msg": "$6$rounds=656000$secretsalt$6iCIB9lwJlehJzZ05OGykop7mNNWfWqIL5.uW2Tm
Yv5lMnD6EZS4DODMKzBlAYH94aPV/aBQfa3/7beuRuaQh/"
}
```

Cela nous génère donc un hash que nous allons copier dans notre prochaine commande pour définir le mot de passe de l'utilisateur à créer :

```
ansible -i inventory.ini -m user -a 'name=user-ansible
password=$6$rounds=656000$secretsalt$n5LY9Y4Ef8Fw4DZNJ0IVBetYyus0hsxF4qUXdjL1YtdJxV
XdPJ6Tz.LaHX/gFCWXONt9YASTDX5I5PMYyAbtK1' --user root --ask-pass all
```

La commande ci-dessus va :

- Utiliser le fichier « **inventory.ini** » pour cibler les nodes
- Utiliser le module « **User** » pour créer notre utilisateur
- Définir le nom de l'utilisateur
- Définir le mot de passe via le hash généré précédemment

- Nous lançons la commande en « **root** » via « **–user root** » ce qui est obligatoire pour le moment car aucun autre utilisateur n'a les droits sudo sur les nodes
- Demande le mot de passe root pour la connexion SSH via **« –ask-pass »**
- Pour les nodes cibles nous avons définie « **all** » pour déployer la configuration sur tous les nodes du fichier inventory.

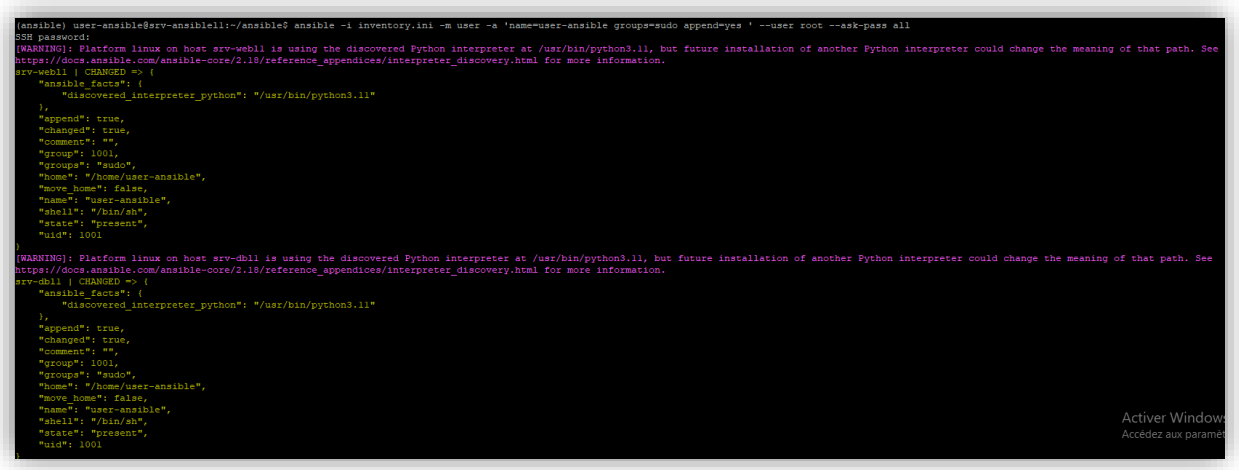
Voici le retour de commande en état « CHANGED » :

```
(ansible) user-ansible@srv-ansible11:~/ansible$ ansible -i inventory.ini -m user
 -a 'name=user-ansible
password=$6$rounds=656000$secretsalt$n5LY9Y4Ef8Fw4DZNJ0IVBetYyus0hsxF4qUXdjL1Ytd
JxVXdPJ6Tz.LaHX/gFCWXONt9YASTDX5I5PMYyAbtK1' --user root --ask-pass all
SSH password:
[WARNING]: Platform linux on host srv-db11 is using the discovered Python
interpreter at /usr/bin/python3.11, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
srv-db11 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": true,
    "comment": "",
    "create_home": true,
    "group": 1001,
    "home": "/home/user-ansible",
    "name": "user-ansible",
    "password": "NOT_LOGGING_PASSWORD",
    "shell": "/bin/sh",
    "state": "present",
    "system": false,
    "uid": 1001
}
[WARNING]: Platform linux on host srv-web11 is using the discovered Python
interpreter at /usr/bin/python3.11, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
srv-web11 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": true,
    "comment": "",
    "create_home": true,
    "group": 1001,
    "home": "/home/user-ansible",
    "name": "user-ansible",
    "password": "NOT_LOGGING_PASSWORD",
    "shell": "/bin/sh",
    "state": "present",
    "system": false,
    "uid": 1001
}
```

Nous pouvons maintenant ajouter notre utilisateur au groupe **« sudo »** afin de ne plus utiliser l'utilisateur **« root »** :

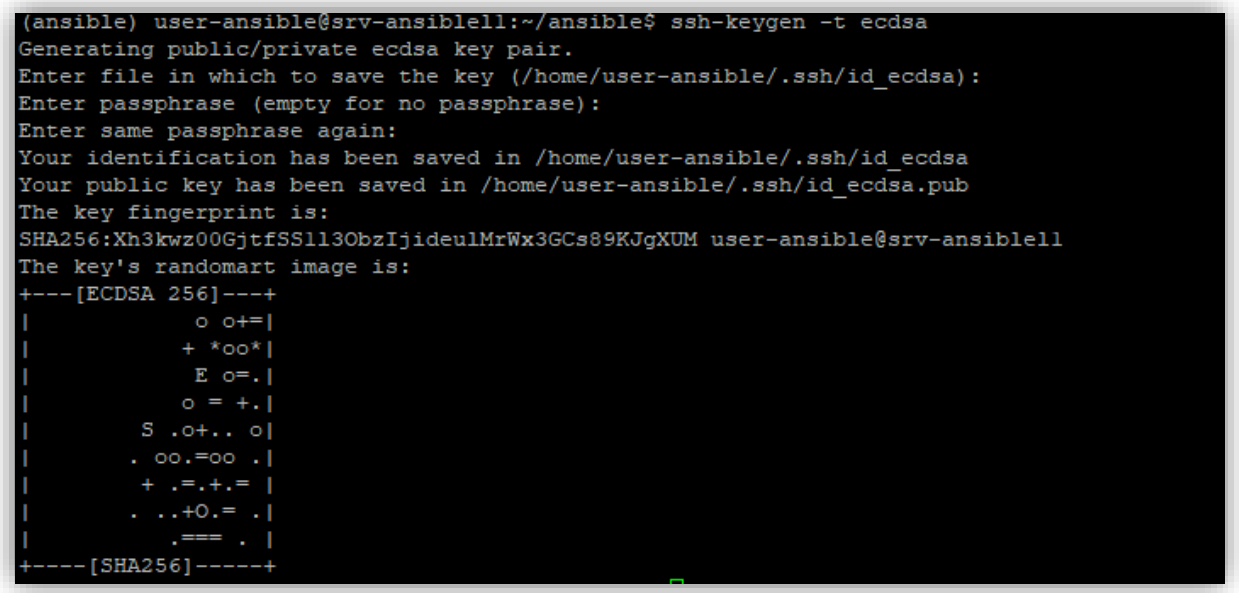
```
ansible -i inventory.ini -m user -a 'name=user-ansible groups=sudo append=yes ' --
user root --ask-pass all
```

Dans cette commande, nous utilisons les mêmes paramètres que pour la création de l'utilisateur mais ajustons ceux-ci pour le groupe d'appartenance à **« sudo »**



Nous allons maintenant générer nos clé SSH :

```
ssh-keygen -t ecdsa
```

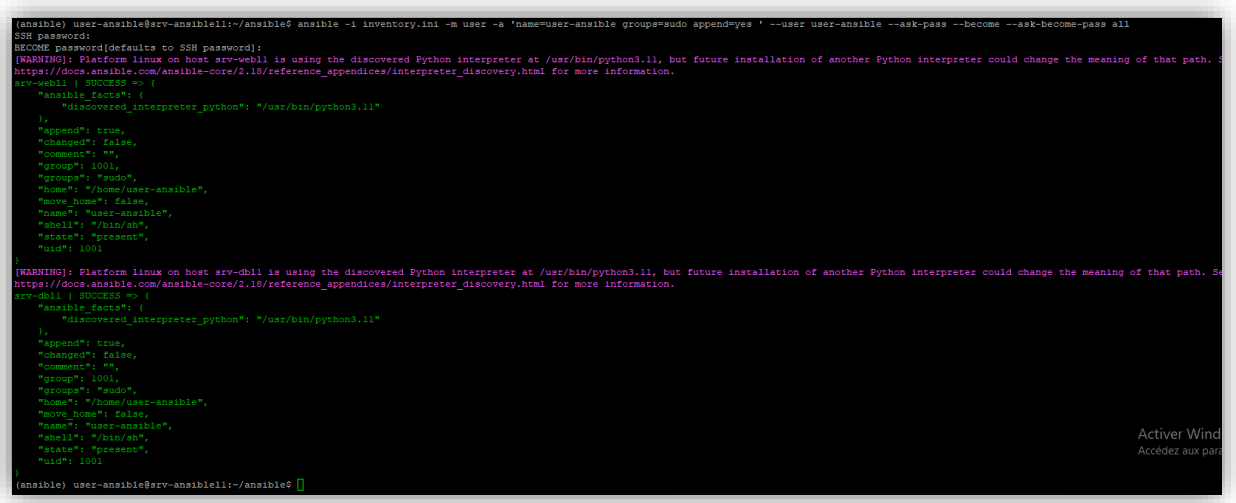


Ci-dessus nous voyons notre clé publique ainsi que le répertoire d'enregistrement de celle-ci.

Nous allons maintenant déployer la clé sur nos nodes :

```
ansible -i inventory.ini -m authorized_key -a 'user=user-ansible state=present
key="{{
lookup("file", "/home/user-ansible/.ssh/id_ecdsa.pub") }}"' --user user-ansible --
ask-pass --become --ask-become-pass all
```

Ici nous utilisons le module **« authorized_key »** pour copier la clé vers les nodes distant.
Des option se sont également ajouté qui sont le « **become** » et **« ask-become-pass ».**
Celle-ci sont nécessaire afin de faire une élévation de privilèges pour les droits sudo.
Voici le retour de la commande en « SUCCES » :



Nous créons maintenant l'arborescence de nos dossiers et fichiers pour nos différents rôles.
En effet, dans Ansible une arborescence propre et bien définie est nécessaire par soucis de lisibilité et d'organisation.
Voici donc l'arborescence du dossier **« roles »** :

```
(ansible) user-ansible@srv-ansible11:~/ansible/roles$ tree
.
├── apache
│   ├── handlers
│   │   └── main.yml
│   └── tasks
│       ├── main.yml
│       └── php-install.yml
├── fail2ban
│   └── tasks
│       └── main.yml
├── glpi
│   ├── commun
│   │   └── defaults
│   │       └── main.yml
│   ├── confapache
│   │   ├── meta
│   │   │   └── main.yml
│   │   ├── tasks
│   │   │   ├── main.yml
│   │   │   └── main.yml.save
│   │   └── templates
│   │       └── virtualhost.conf
│   └── confdb
│       ├── meta
│       │   └── main.yml
│       └── tasks
│           └── main.yml
└── mariadb
    └── tasks
        └── main.yml

18 directories, 12 files
(ansible) user-ansible@srv-ansible11:~/ansible/roles$
```

Nous allons donc avoir :

- Un rôle **« apache »** qui servira pour l'installation d'apache.
  - Le dossier **« handlers »** pour redémarrer le service.
  - Le dossier **« task »** composé de :

- **« main.yml »** qui sera le script d'installation du rôle.
- **« php-install.yml »** pour l'installation de php (qui aurais pu être intégré avec l'installation d'apache mais que nous avons choisi de séparer) .

- Un rôle **« mariadb »** qui reprends la même arborescence que le rôle apache mais qui servira pour l'installation mariadb

- Un rôle **« GLPI »** qui contient :
  - **« commun »** dans lesquels nous placerons un fichier contenant des variables communes pour le serveur apache2 et mariadb
  - « **confapache** » composé de :
    - **« Meta »** qui servira pour indiquer les dépendances avec le dossier **« commun »** afin de retrouver les variables
    - **« Tasks »** : pour les taches de configuration du service Apache
    - **« templates »** : sera utilisé pour définir le template du fichier VirtualHost de Apache2
  - **« confdb »**
    - **« Meta »** qui servira pour indiquer les dépendances avec le dossier **« commun »** afin de retrouver les variables
    - **« Tasks »** : pour les taches de configuration du service MariaDB

- Un rôle « **fail2ban** » pour les taches de déploiement et configuration du service

2.2 Configuration et déploiement du service Apache

Nous commençons par écrire notre script YAML pour le déploiement d'APACHE.
Attention : L'indentation est très importante dans le YAML, il est obligatoire de bien la respecter pour les bon fonctionnements des script !

Nous créons donc le fichier **« roles/apache/tasks/main.py »** :

```yaml
---

# 1. Install Apache2
-  name: "apache install" # Nom de la tache
   apt: # Module utilisé
    name: "apache2"
    state: "present"

# 2. Active le service Apache
- name: "apache service activation"
  service:
    name: "apache2"
    state: "started"
    enabled: yes
```

```
# 3. Install php
- name: "install php"
  include_tasks: "php-install.yml" # Lance le script d'installation de PHP
  when: php_install | default(False) | bool
```

Nous allons maintenant rédiger le script pour PHP dans **« roles/apache/tasks/php-install.yml »** :

```
#1. Installation de PHP
- name: "install php"
  apt:
    name: "php,php-mysql,php-xml,php-mbstring,php-gd,php-intl"
    state: latest
  changed_when: yes
  notify: [ "apache restart" ]
```

Nous pouvons ensuite créer notre script pour redémarrer le service dans **« roles/apache/handlers/main.yml » :**

```
---
# Restart Apache
- name: "apache restart"
  service:
    name: "apache2"
    state: "restarted"
```

Il faut ensuite créer notre playbook pour jouer notre rôle.
Nous créons donc un fichier **« install-apache.yml »**:

```
(ansible) user-ansible@srv-ansible11:~/ansible$ ls
bin  include  install-apache.yml  inventory.ini  lib  lib64  pyvenv.cfg  roles
(ansible) user-ansible@srv-ansible11:~/ansible$
```

Et nous écrivons notre playbook :

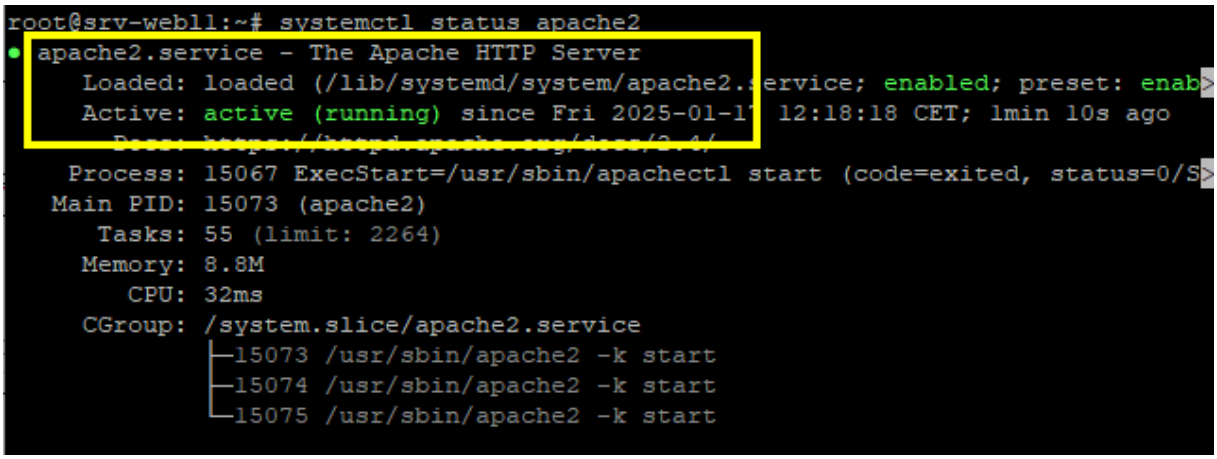
```
---
-  name: "Installation apache"
   hosts: srv-web11
   roles:
    - role: "apache"
   vars:
     php_install: yes
```

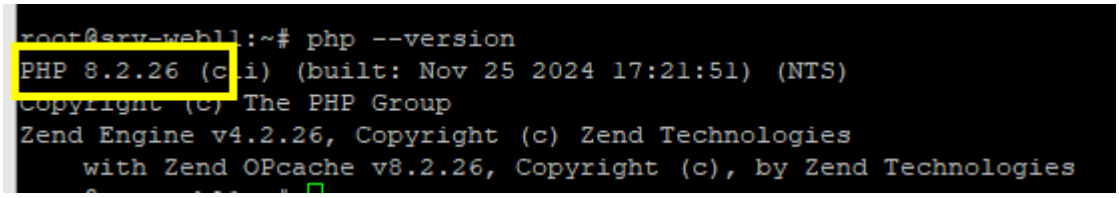
Puis nous lançons la commande suivante pour l'exécuter :

```
ansible-playbook -i inventory.ini --user user-ansible --become --ask-become-pass
install-apache.yml
```

```
(ansible) user-ansible@srv-ansiblel1:~/ansible$ ansible-playbook -i inventory.in
i --user user-ansible --become --ask-become-pass install-apache.yml
BECOME password:
[WARNING]: Could not match supplied host pattern, ignoring: SRV-WEBL1

PLAY [Installation apache] *****************************************************
skipping: no hosts matched

PLAY RECAP *********************************************************************

(ansible) user-ansible@srv-ansiblel1:~/ansible$ nano install-apache.yml
(ansible) user-ansible@srv-ansiblel1:~/ansible$ ansible-playbook -i inventory.in
i --user user-ansible --become --ask-become-pass install-apache.yml
BECOME password:

PLAY [Installation apache] *****************************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: Platform linux on host srv-webl1 is using the discovered Python
interpreter at /usr/bin/python3.11, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
ok: [srv-webl1]

TASK [apache : apache install] *************************************************
changed: [srv-webl1]

TASK [apache : apache service activation] **************************************
ok: [srv-webl1]

TASK [apache : install php] ****************************************************
included: /home/user-ansible/ansible/roles/apache/tasks/php-install.yml for srv-
webl1

TASK [apache : install php] ****************************************************
changed: [srv-webl1]

RUNNING HANDLER [apache : apache restart] **************************************
changed: [srv-webl1]

PLAY RECAP *********************************************************************
srv-webl1                  : ok=6    changed=3    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Ci-dessus nous voyons le récapitulatif de notre playbook.
Sur le serveur **« srv-webl1 »** nous vérifions le statuts du service Apache et la version de PHP :





Notre installation c'est correctement déroulé, nous allons donc passer à l'installation de MariaDB.

2.3 Installation de MariaDB

Dans la logique que pour Apache, nous allons écrire notre script pour installer mariadb.

Nous commençons par **"role/mariadb/task/main.yml"**

```
---

#1. Installation de mariadb et Python
- name: "Install Mariadb"
  apt:
    name: "mariadb-server,python3-mysqldb"
    state: "present"

#2. Démarre le service MariaDb
- name: "start service MariaDb"
  service:
    name: "mariadb"
    state: "started"
    enabled: "yes"

#3. Configure le port d'écoute
- name: "change 50-server.cnf"
  command:
    /usr/bin/sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/mariadb.conf.d/50-
server.cnf # Insert le paramètre suivant dans le fichier de configuration

#4. Redémarre MariaDb
```

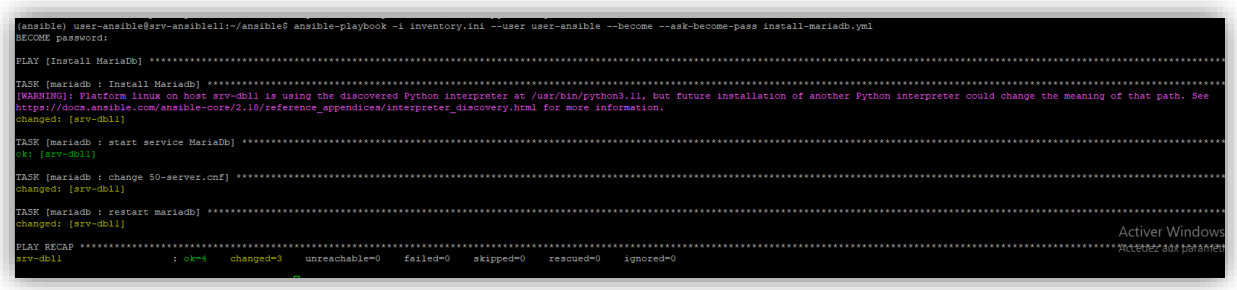
```
- name: "restart mariadb"
  service:
    name: "mariadb"
    state: "restarted"
```

Puis notre Playbook **“install-mariadb.yml”**:

```
---
- name: "Install MariaDb"
  hosts: "srv-dbl1"
  gather_facts: no
  roles:
    - role: mariadb
```

Nous lançons ensuite le playbook :

```
ansible-playbook -i inventory.ini --user user-ansible --become --ask-become-pass
install-mariadb.yml
```

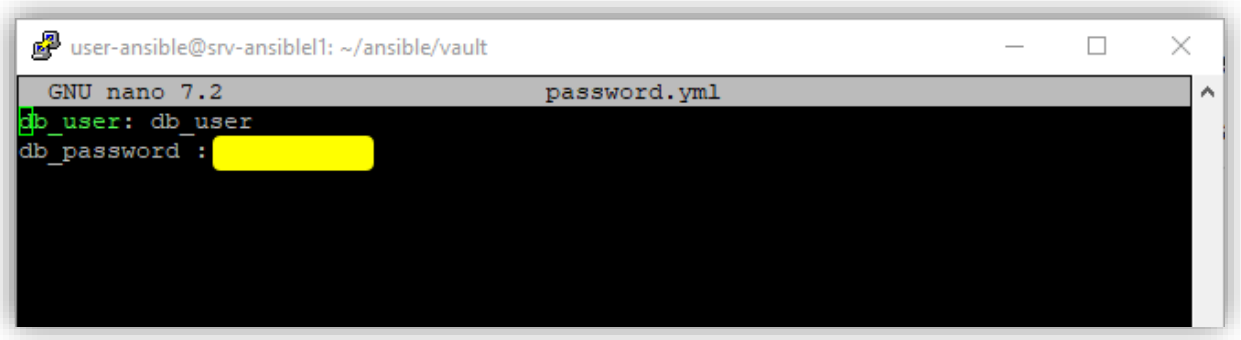


Nous vérifions que le service est bien installé :

2.4 Configurations des services et installation de GLPI

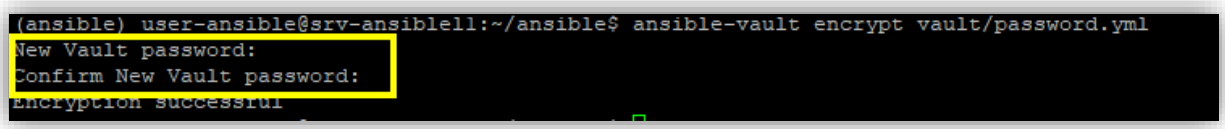
Nous pouvons maintenant passer au déploiement de GLPI.
Cependant, avant cela nous allons créer un dossier **“Vault”** qui contiendra un fichier **“password.yml”** qui sera chiffrée via le vault ansible afin de rentrer les variables ne devant pas apparaitre en clair (dans notre cas l’utilisateur et mot de passe pour la base de données).

Nous créons donc notre dossier puis le fichier que nous éditons :

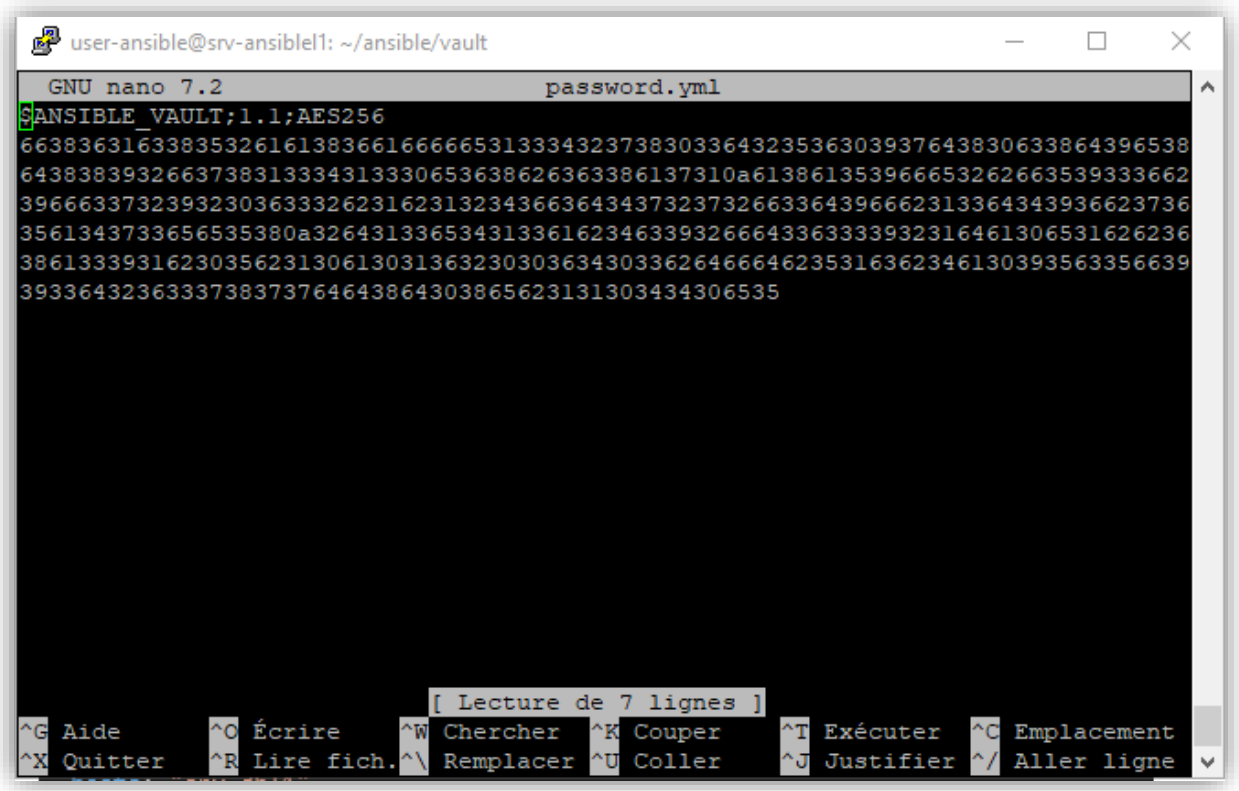


Nous chiffrons celui-ci avec la commande :

```
ansible-vault encrypt vault/password.yml
```



Maintenant si nous ouvrons ce même fichier sans le mot de passe défini précédemment nous pouvons voir le contenue chiffré via AES256



Si besoin, pour le déchiffrer utiliser la commande :

```
ansible-vault decrypt password.yml
```

Maintenant nous allons pouvoir créer notre script **« roles/glpi/commun/main.yml »** pour définir les variables communes à Apache et MariaDB :

```
# Nom de la base de donnée utilisé par GLPI
glpi_db_name: "GLPI"

# User et password (Variable définie dans le ficher « password.yml »)
glpi_db_user: '{{db_user}}'
glpi_db_password: '{{db_password}}'

# Nom de glpi
glpi_name: "glpi"

# Répertoire d'installation
glpi_directory: "/var/www/{{glpi_name}}"
glpi_directory_log: "/var/log/{{glpi_name}}"
glpi_directory_css: "/var/lib/{{glpi_name}}"
glpi_directory_config: "/etc/{{glpi_name}}"

#Source GLPI
```

```
glpi_archive_url: "https://github.com/glpi-
project/glpi/releases/download/10.0.10/glpi-10.0.10.tgz"
```

Une fois cela fait nous écrivons notre script pour la configuration de MariaDB :

```
---
- name: "Defini le fichier pour les vars des MDP"
  include_vars:
    file: "~/ansible/vault/password.yml"

# 1. Création de la BDD
- name: "glpi database"
  community.mysql.mysql_db:
    name: "{{glpi_db_name}}"
    state: present

# 2. Création de l'utilisateur et définitions des droits
- name: "GLPI user and privileges"
  community.mysql.mysql_user:
    name: "{{glpi_db_user}}"
    password: "{{glpi_db_password}}"
    priv: "{{glpi_db_name}}.*:ALL"
    host: "srv-webl1"
    state: present
```

Maintenant nous rédigeons le script pour la configuration du service Apache2 pour GLPI:

```
---
- name: "Defini le fichier pour les vars des MDP"
  include_vars:
    file: "~/ansible/vault/password.yml"

#0. Ajout du compte user ansible dans le groupe www-data pour la gestion des
droits
- name: Add user-ansible to www-data group
  command: usermod -a -G www-data user-ansible

#1. Création des répertoires pour GLPI
- name: "glpi directory"
  ansible.builtin.file:
    path: "{{item}}" # Variable dynamique qui sera remplacée par chaque
élément de la liste "with_items"
    owner: "www-data" # Définie le propriétaire
    group: "www-data" # Groupe dédie aux utilisateurs de l'application
    state: directory # Spécifie que chaque élément doit être un répertoire
    mode: "0775" # Assigne les droits aux dossiers
    recurse: yes
```

```
  with_items: # Dossiers à créer
    - "{{glpi_directory}}"
    - "{{glpi_directory_log}}"
    - "{{glpi_directory_css}}"
    - "{{glpi_directory_config}}"
    - "/etc/ssl/glpi"

#2. Décompresse les fichiers sources
- name: "uncompress GLPI archive"
  unarchive:
    src: "{{glpi_archive_url}}"
    dest: "{{glpi_directory}}"
    owner: "www-data"
    group: "www-data"
    mode: "0775"
    remote_src: yes # Indique que la source est distante

#3. Génère la private Key
- name: Generate SSL private key
  command: openssl genrsa -out /etc/ssl/glpi/glpi.key 2048
  args:
    creates: /etc/ssl/glpi/glpi.key

#4. Génère le certificat SSL
- name: Generate SSL certificate
  command: >
    openssl req -x509 -new -nodes
    -key /etc/ssl/glpi/glpi.key
    -sha256 -days 365
    -out /etc/ssl/glpi/glpi.crt
    -subj "/C=FR/ST=IDF/L=Paris/O=GLPI/CN=srv-webl1"
  args:
    creates: /etc/ssl/glpi/glpi.crt

#5. Copie les fichiers GLPI dans les répertoires
- name: "Copie les fichiers de configuration"
  ansible.builtin.copy:
    src: /var/www/glpi/config
    dest: "{{glpi_directory_config}}"
    remote_src: true
    mode: '0775'
    owner: user-ansible
    group: www-data

- name: "Copie les fichiers CSS"
  ansible.builtin.copy:
    src: /var/www/glpi/files
    dest: "{{glpi_directory_css}}"
    remote_src: true
```

```
    mode: '0775'
    owner: user-ansible
    group: www-data

#6. Créer les fichiers nécessaire pour GLPI
- name: Configure GLPI downstream.php
  ansible.builtin.blockinfile:
    path: /var/www/glpi/inc/downstream.php
    create: yes
    marker: ""
    block: |
      <?php
      define('GLPI_CONFIG_DIR', '/etc/glpi/');
      if (file_exists(GLPI_CONFIG_DIR . '/local_define.php')) {
          require_once GLPI_CONFIG_DIR . '/local_define.php';
      }

- name: Configure GLPI local_define.php
  ansible.builtin.blockinfile:
    path: /etc/glpi/local_define.php
    create: yes
    marker: ""
    block: |
      <?php
      define('GLPI_VAR_DIR', '/var/lib/glpi/files');
      define('GLPI_LOG_DIR', '/var/log/glpi');

#7. Créer le fichiers VirtualHost Apache
- name: Setup virtualhost
  ansible.builtin.template:
    src: /glpi/templates/virtualhost.conf
    dest: "/etc/apache2/sites-available/glpi.conf"

#8. Active les modules Apache/PHP
- name: Configure Apache modules and sites
  command: "{{ item }}"
  with_items:
    - a2enmod ssl
    - a2enmod rewrite
    - a2enmod proxy_fcgi
    - a2enmod setenvif
    - a2enconf php8.2-fpm
    - a2ensite glpi.conf
    - a2dissite 000-default.conf

#9. Finalise l'installation de GLPI
- name: Install GLPI
  ansible.builtin.command:
```

```
    "php bin/console -n db:install -H 192.168.40.4 -P 3306 -d glpi -u
{{db_user}} -p {{db_password}} -L fr_FR --reconfigure --force"
  args:
    chdir: "/var/www/glpi"

#10. Nettoie le fichier d'installation PHP
- name: Clean install.php file
  ansible.builtin.file:
    path: "/var/www/glpi/install/install.php"
    state: absent

#11. Restart le service Apache
- name: Restart Apache
  ansible.builtin.service:
    name: apache2
    state: restarted
```

Il est également important de créer les dépendances avec les fichiers contenant les variables communes.
Pour cela dans les dossiers **« roles/confapache/meta »** et **« roles/confdb/meta »** nous créons le fichier **« main.yml »** suivant :

```
dependencies:
- role: "mediawiki/commun"
```

Enfin nous créons le playbook **« install_glpi »** :

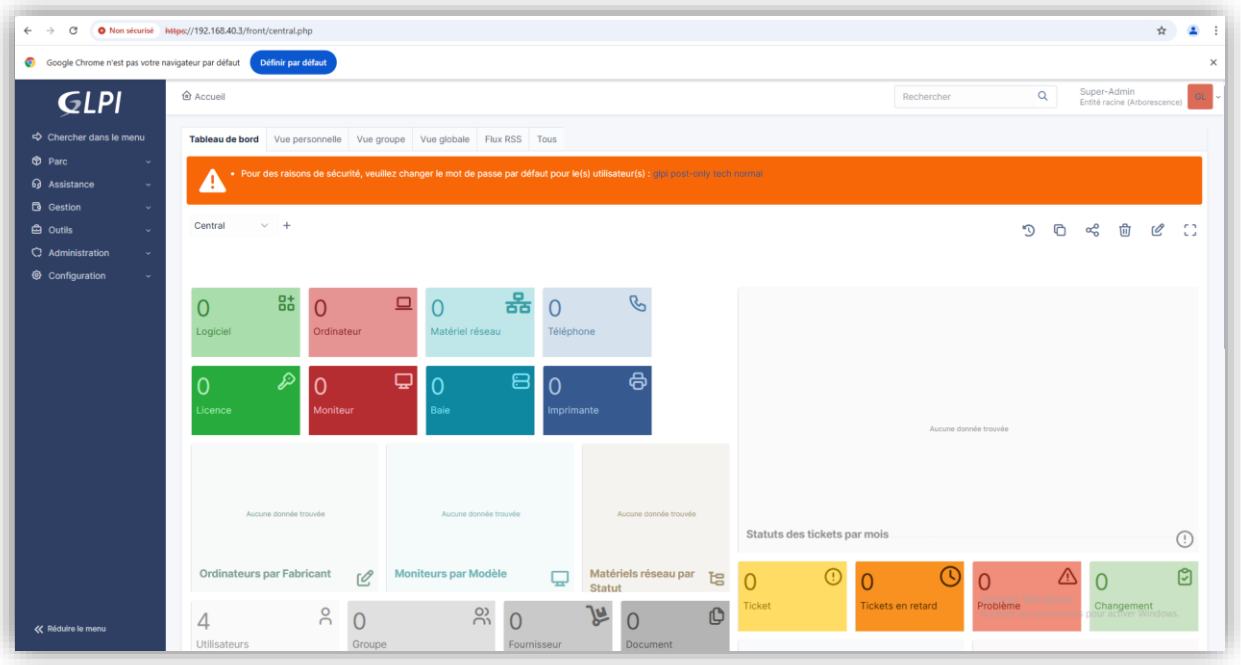
```
---
- name: "glpi db conf"
  hosts: "srv-dbl1"
  gather_facts: no
  roles:
    - role: "glpi/confdb"

- name: "glpi apache conf"
  hosts: "srv-webl1"
  gather_facts: no
  roles:
    - role: "glpi/confapache"
```

Nous pouvons maintenant lancer notre déploiement et voir le récapitulatif :

```
ansible-playbook -i inventory.ini --user user-ansible --become --ask-become-pass
install-glpi.yml --ask-vault-pass
```

```
PLAY RECAP *********************************************************************
srv-dbl1                   : ok=3    changed=2    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
srv-webl1                  : ok=14   changed=14   unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Nous vérifions le bon déploiement de GLPI via l'adresse IP de notre serveur « **srv-webl1** »



2.5 Hardening via Fail2ban

Maintenant que nous avons déployé notre solution et qu'elle est fonctionnel, nous pouvons passer au hardening de nos serveurs.

Nous avons donc créé un nouveau rôle **« fail2ban »** en suivant la même logique que pour les précédents.
Nous rédigeons notre script pour l'installation et la configuration de fail2ban qui sera déployé sur nos nodes :

```
---

# 1. Installation des packages Fail2Ban
-  name: "fail2ban install"
   apt:
    name: "fail2ban,iptables"
    state: "present"

# 2. Active le service Fail2ban
```

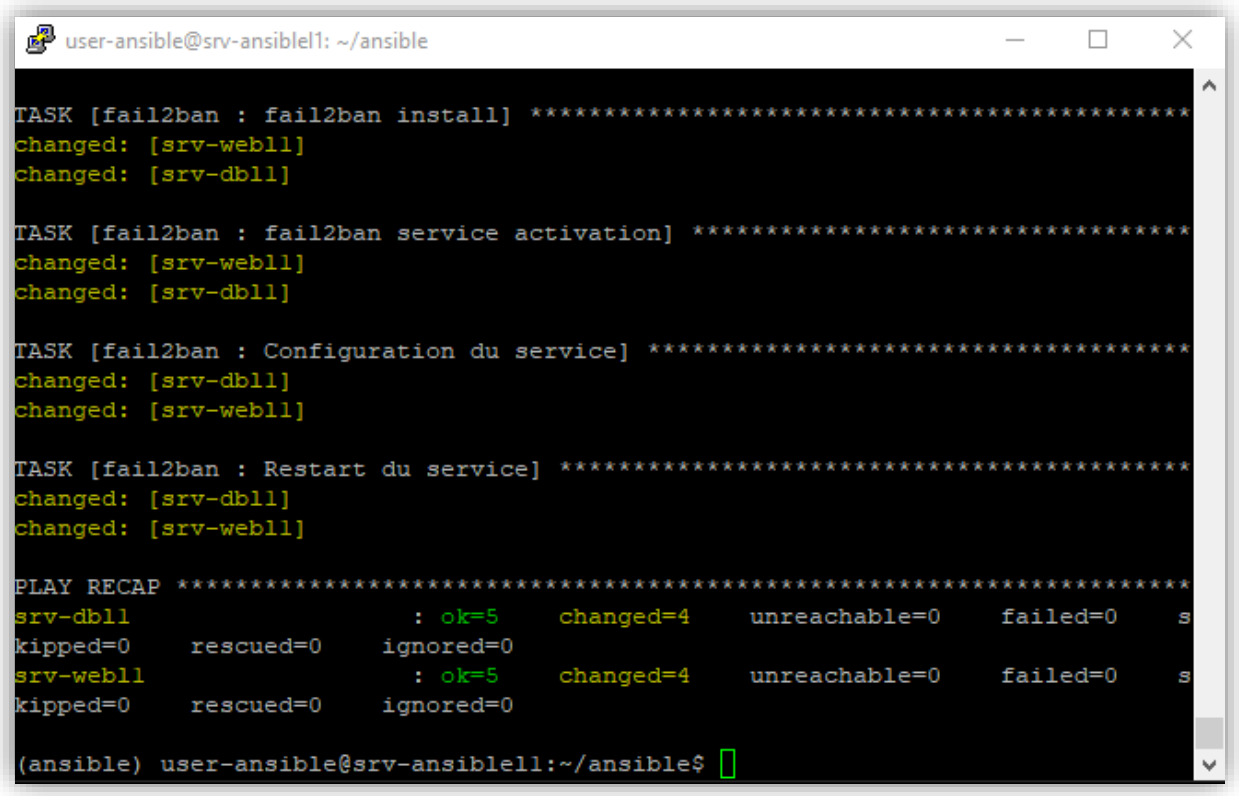
```
- name: "fail2ban service activation"
  service:
    name: "fail2ban"
    state: "started"
    enabled: yes

# 3. Renseigner les paramètres nécessaires
- name: "Configuration du service"
  ansible.builtin.blockinfile:
    path: /etc/fail2ban/jail.local # Chemin du fichier
    create: yes  # Cree le fichier s'il n'existe pas
    block: | # Insert le texte suivant
      [DEFAULT]
      ignoreip = 127.0.0.1 192.168.1.159
      findtime = 3600
      bantime = 3600
      maxretry = 3

      [sshd]
      enabled = true
      port = ssh
      filter = sshd
      logpath = journal
      backend = systemd
      maxretry = 3

      [glpi]
      enabled  = true
      port     = http,https
      filter   = glpi
      logpath = /var/log/glpi/event.log
      maxretry = 3
      findtime = 600

#4. Créer le fichier regex pour lire les logs GLPI
- name: "Configuration du service"
  ansible.builtin.blockinfile:
    path: /etc/fail2ban/jail.local # Chemin du fichier
    create: yes  # Cree le fichier s'il n'existe pas
    block: | # Insert le texte suivant
    [Definition]
    prefregex = ^(?:\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2} \[\@\^\]\+\\n)?
    failregex = ^.*\[login\] 3: Connexion échouée de .* depuis l'IP <HOST>$
    journalmatch = _SYSTEMD_UNIT=glpi.service

# 5. Restart du service
- name: "Restart du service"
  service:
    name: "fail2ban"
```

```
    state: "restarted"
```

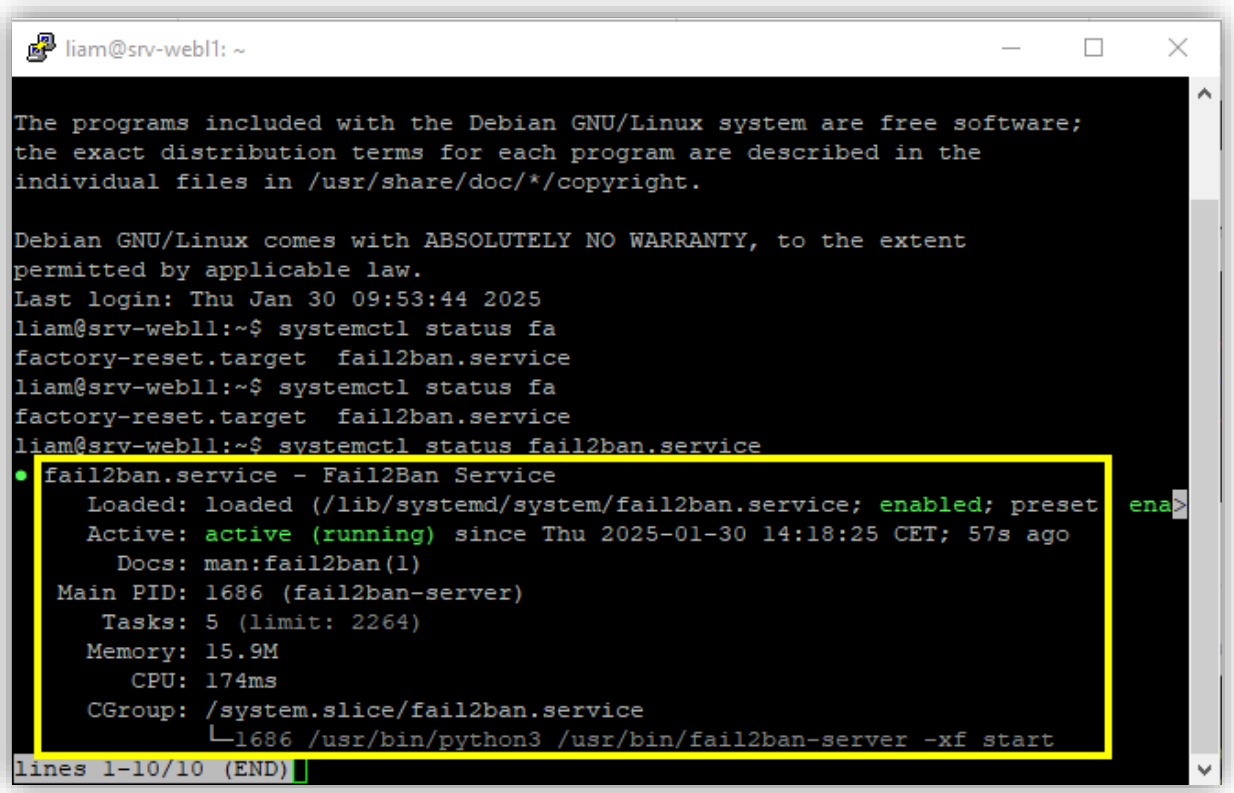
Enfin nous créons le Playbook **« install-fail2ban.yml »:**

```
---
-  name: "Installation fail2ban"
   hosts: all
   roles:
    - role: "fail2ban"
```

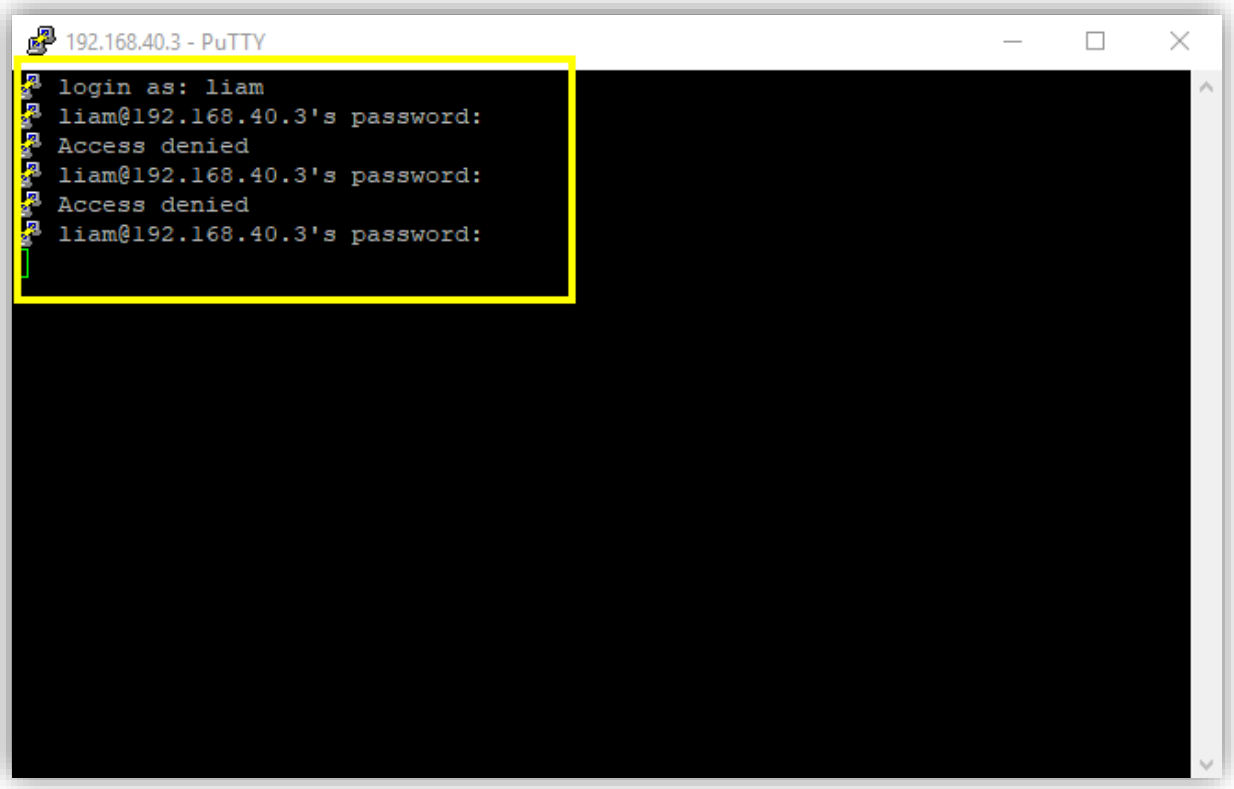
Nous pouvons maintenant l'exécuter :



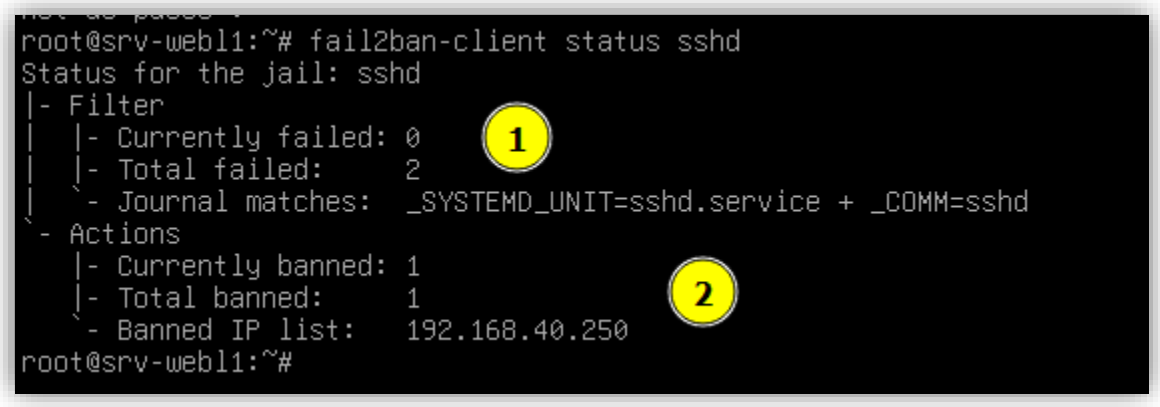
Nous vérifions le statut sur le serveur **« srv-webl1 »** :



Nous allons maintenant effectuer des connexions SSH avec un mot de passe erroné pour s'assurer du bannissement de l'IP :

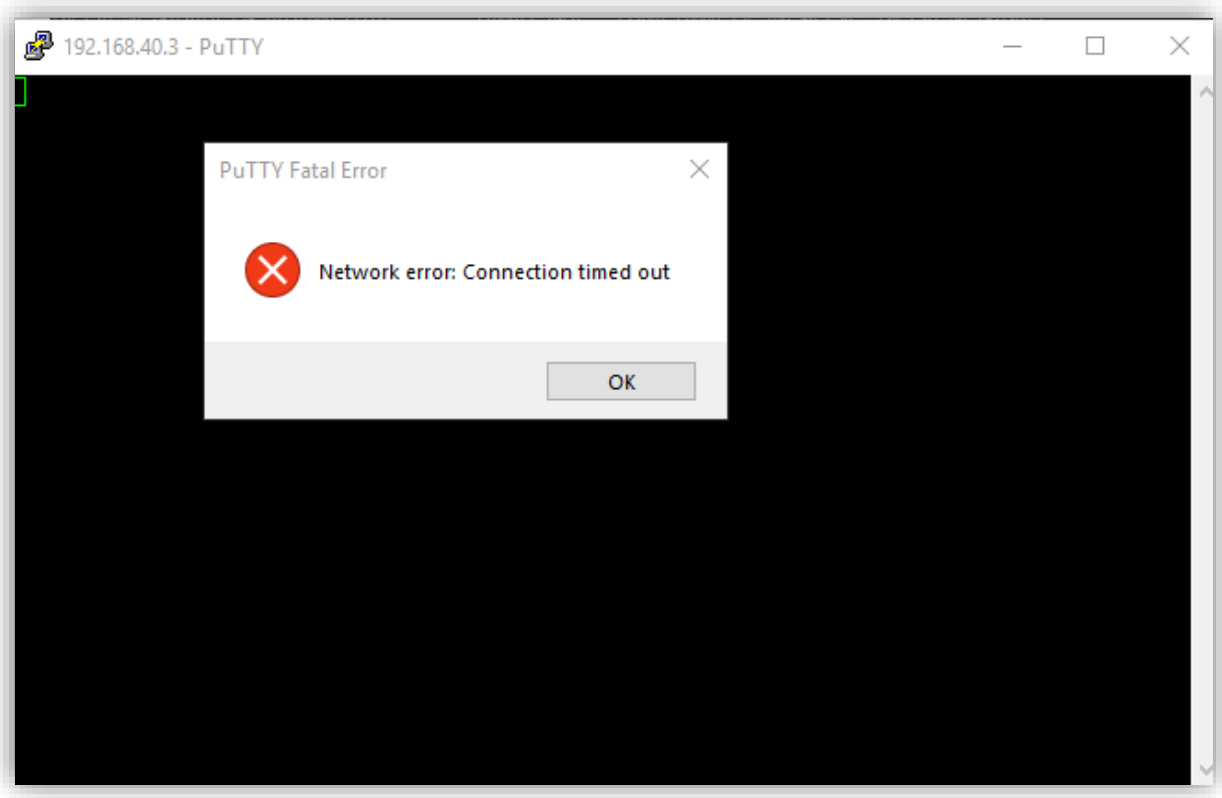


Puis nous vérifions les IP bannies via la commande « **fail2ban-client status sshd** »

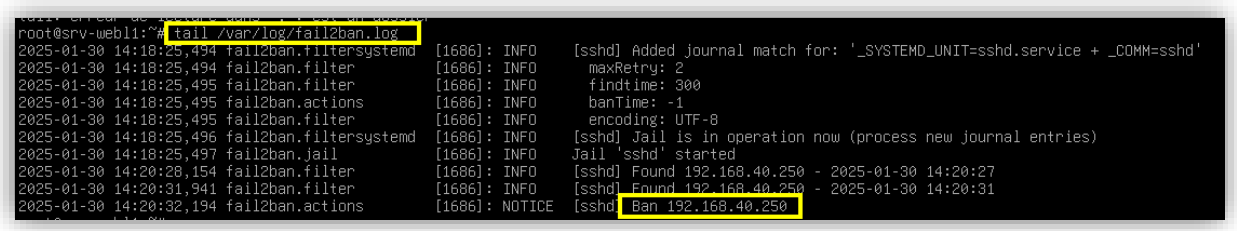


1. Nombres total de connexion failed
2. Nombres d'adresse IP banni en cours, nombre total d'adresse banni ainsi que l'adresse IP.

Nous pouvons désormais essayer de se connecter à nouveau en SSH mais celle-ci passe en timeout :

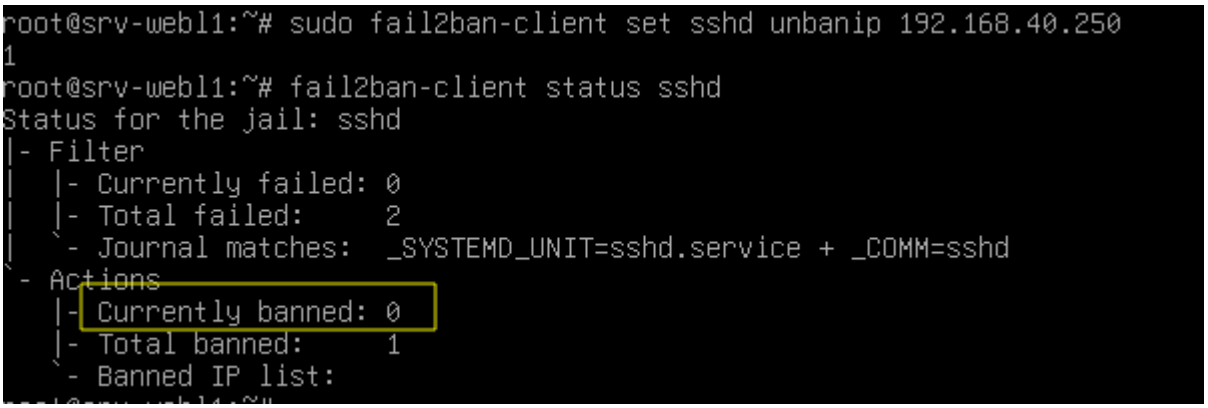


Nous pouvons aussi retrouver les logs dans **« /var/log/fail2ban.log »** :



Afin de unban une IP il faut effectuer la commande suivante :

```
sudo fail2ban-client set sshd unbanip 192.168.40.250
```



Nous voyons que l'IP est bien unban avec l'historique des logs qui est conservé.

Puis nous effectuons les mêmes tests avec des connexions **« failed »** vers l'application GLPI.
Puis nous vérifions les logs via « **fail2ban-client status glpi** » :

3. Conclusion

Dans le cadre de ce projet, nous avons mis en place une solution d'automatisation avec Ansible afin de faciliter le déploiement et la gestion des configurations pour l'ADRAR. Cette solution permettra d'optimiser les futurs déploiements tout en garantissant une uniformité et une sécurité accrue des infrastructures.

Nous avons également déployé GLPI sur un environnement PaaS, accompagné d'un serveur LAMP et d'une sécurisation renforcée des accès, notamment via Fail2Ban et l'authentification par clés SSH.

Le choix d'Ansible s'est imposé pour sa simplicité, sa flexibilité et son coût nul en version open-source. À terme, cette solution permettra à l'ADRAR d'améliorer la gestion de son parc informatique et d'évoluer vers une infrastructure plus automatisée et sécurisée.

4. Annexes

Installation et configuration de Ansible : https://openclassrooms.com/fr/courses/2035796-utilisez-ansible-pour-automatiser-vos-taches-de-configuration

Installation de GLPI : https://www.it-connect.fr/installation-pas-a-pas-de-glpi-10-sur-debian-12/

Configuration de Fail2ban : https://slash-root.fr/fail2ban-installation-et-configuration/