DOCUMENTATION SAE 3.02

Projet : SAÉ 3.02 — Solution de consultation centralisée de journaux systèmes

Yacine Ben Omrane Étudiant en 2ème année de BUT Réseaux et Télécommunications Parcours Cybersécurité

1. Présentation du projet

Le besoin

Dans le cadre de l'administration système, gérer plusieurs serveurs Linux demande de vérifier régulièrement les logs. Se connecter sur chaque machine une par une pour lire le fichier /var/log/syslog est trop long .

La solution

J'ai développé une application Web centralisée qui permet de :

- Se connecter via une interface sécurisée.
- Gérer différents niveaux de droits (Utilisateur, Gestionnaire, Admin).
- Récupérer et afficher les logs de plusieurs serveurs distants au même endroit.
- Gérer le parc de machines et les comptes utilisateurs.

2. Choix Techniques

Pour réaliser ce projet, j'ai choisi une architecture légère et modulaire basée sur Python.

Technologies utilisées

- **Langage :** Python 3.
- **Framework Web :** Flask (choisi pour sa simplicité et pour l'avoir étudié durant mes TP).
- **Base de données :** MariaDB.
- **SSH :** Librairie Paramiko (pour exécuter des commandes sur les serveurs distants).
- **Style :** HTML / CSS.

Architecture

J'ai organisé mon code pour séparer les responsabilités, même si tout tient dans une structure simple pour faciliter le déploiement :

1. **Modèle :** La structure de la base de données (Tables SQL).
2. **Vue :** Les fichiers HTML dans le dossier /templates.
3. **Contrôleur :** Le fichier app.py qui gère les routes et la logique.

3. Structure des données et Privilèges

Base de données

La base de données sae302 contient 4 tables principales :

- utilisateurs : Stocke les infos de connexion et le lien vers le rôle.
- roles : Définit les types d'utilisateurs (Utilisateur, Gestionnaire, Admin).
- machines : Liste des serveurs à surveiller (Nom et IP).
- privileges : Liste descriptive des droits.

Gestion des droits (Bit à Bit)

Comme demandé dans le sujet, j'ai géré les permissions en utilisant des opérations "bit à bit". Cela permet de cumuler des droits avec une simple addition de puissances de 2.

- **Droit 1** (1) : Consulter les logs.
- **Droit 2** (2) : Gérer les serveurs (Ajout/Suppression).
- **Droit 4** (4) : Gérer les utilisateurs (Admin).

Exemple d'implémentation :

Un administrateur a le droit 7, car 1 + 2 + 4 = 7.

Dans le code Python, je vérifie les droits avec l'opérateur & :

```python
# Verifie si l'utilisateur a le privilege requis
def a_privilege(privilege_requis):
    """Verifie si l'utilisateur a le privilège requis (opération bit à bit)"""
    if 'privileges' in session:
        privileges_user = session['privileges']
        # bit à bit
        if (privileges_user & privilege_requis) == privilege_requis:
            return True
    return False
```

4. Fonctionnalités et Développement

4.1 Authentification Sécurisée

Je n'ai pas stocké les mots de passe en clair. Lors de la création d'un utilisateur ou de la connexion, j'utilise l'algorithme **SHA-256** via la librairie hashlib de python.

- Si le hash du mot de passe saisi correspond au hash en base de données, l'utilisateur est connecté.
- Les informations sont ensuite stockées dans une session Flask sécurisée.

4.2 Récupération des logs (SSH)

C'est le cœur du projet. J'ai créé une fonction qui distingue deux cas :

1. **Serveur Local :** Si l'IP est celle de la machine locale, je lis directement le fichier avec open().
2. **Serveur Distant :** J'utilise paramiko pour ouvrir une connexion SSH, lancer la commande tail -n 100 /var/log/syslog et récupérer le résultat.

Extrait du code de récupération :

```
def recuperer_logs(ip, nb_lignes=100):
    """Recupere les logs syslog d'un serveur"""
    # Si serveur local, lecture directe
    if ip == '192.168.122.100':
        try:
            with open('/var/log/syslog', 'r') as f:
                lignes = f.readlines()
                return ''.join(lignes[-nb_lignes:])
        except:
            return "Erreur lecture fichier local"

    # Sinon connexion SSH
    try:
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect(ip, username=SSH_USER, password=SSH_PASSWORD, timeout=10)

        stdin, stdout, stderr = client.exec_command('tail -n ' + str(nb_lignes) + ' /var/log/syslog')
        resultat = stdout.read().decode('utf-8')
        client.close()
        return resultat
    except Exception as e:
        return "Erreur SSH: " + str(e)
```

4.3 Interface dynamique

Le menu de l'application s'adapte automatiquement. Grâce à Jinja2, j'affiche ou je masque les onglets "Serveurs" et "Utilisateurs" selon le niveau de droit de la personne connectée.

5. Installation et Déploiement

Pour faciliter la correction et le déploiement, j'ai créé un script Bash nommé install.sh.

Ce script automatise toutes les tâches fastidieuses :

1. Mise à jour du système (apt update).
2. Installation de MariaDB et Python.
3. Création automatique de la base de données et de l'utilisateur MySQL.
4. Injection des données par défaut (Admin par défaut, rôles).
5. Création de l'environnement virtuel Python et installation des librairies.

Il suffit de lancer ./install.sh pour avoir un projet fonctionnel.

6. Bilan et Difficultés

Plusieur défis techniques rencontrés comme par exemple :

- **Droits de lecture :** Au début, je n'arrivais pas à lire /var/log/syslog car c'est un fichier protégé. J'ai résolu le problème en ajoutant mon utilisateur Linux au groupe adm (sudo usermod -aG adm user).
- **Connexion BDD :** J'ai eu quelques soucis pour configurer la connexion MySQL au début, résolus en créant un utilisateur dédié sae302 au lieu d'utiliser root.

Conclusion

Cette SAÉ a été très formatrice car elle m'a permis de mettre en pratique des notions de plusieurs matières : le réseau , le système Linux avec les droits et les scripts Bash, et le développement Web avec Flask. C'est satisfaisant de voir tous ces éléments fonctionner ensemble dans une seule application.

L'application est aujourd'hui fonctionnelle et respecte le cahier des charges.