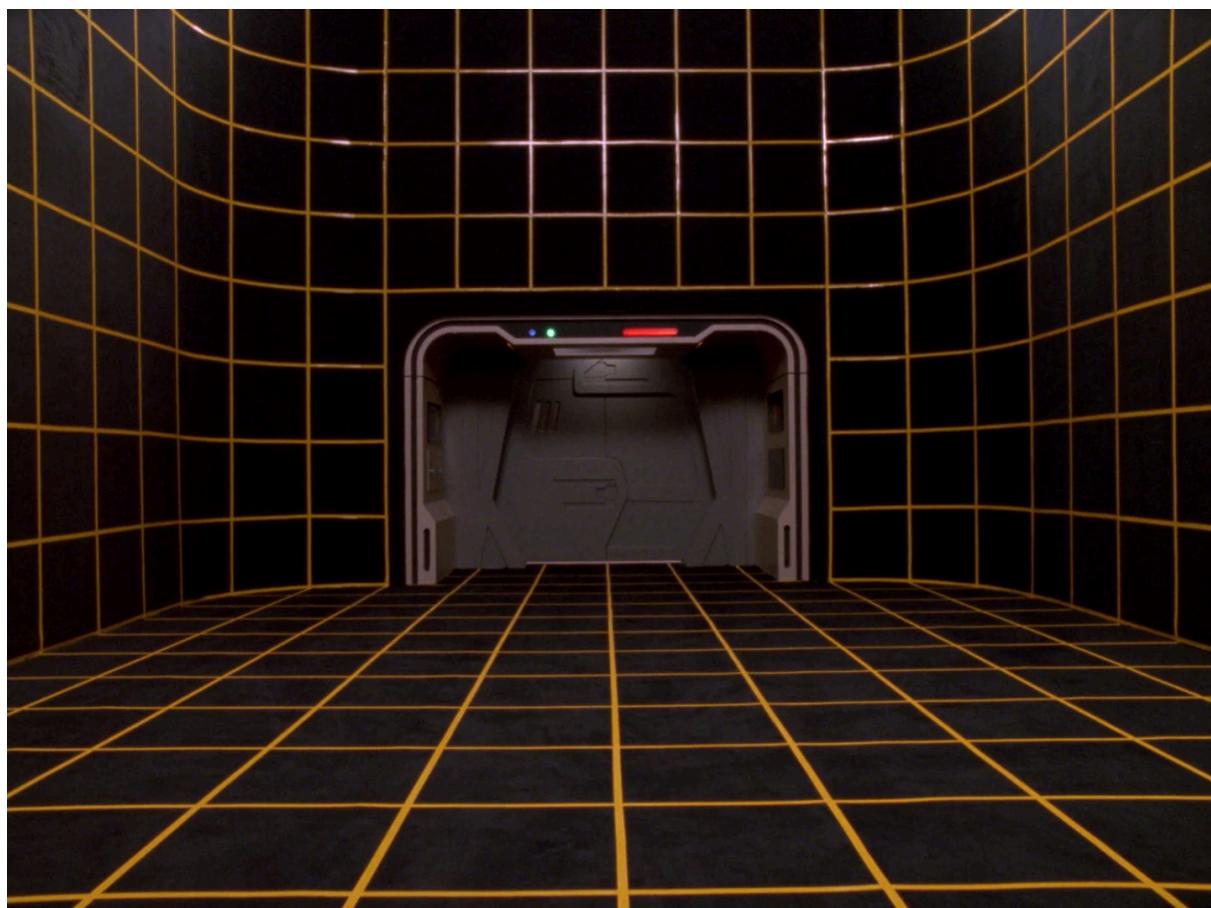


PSMM



projet réalisé par Youcef,Jordan

Résumé-projet PSMM:

1. Création des environnements virtuels :

1.1 3 VM Debian :

- Serveur FTP : 1 Go RAM, 1 vCPU, 8 Go disque
- Serveur Web : 1 Go RAM, 1 vCPU, 8 Go disque (Apache/Nginx, authentification basique)
- Serveur SQL : 2 Go RAM, 2 vCPU, 8 Go disque

2. VM dédiée aux scripts Python :

- Debian sans interface graphique, avec Python, MySQL/MariaDB (client uniquement), FTP et outils d'envoi de mails.

3. Scripts de gestion des serveurs :

3.1 SSH :

- ssh_login.py : Connexion SSH et exécution de commandes
- ssh_login_sudo.py : Connexion SSH avec commandes sudo

3.2 MariaDB/MySQL :

- ssh_mysql.py : Vérification des accès au serveur
- ssh_mysql_error.py : Récupération des erreurs de connexion

3.3 FTP :

- ssh_ftp_error.py : Extraction des logs d'erreurs FTP

3.4 Web :

- ssh_web_error.py : Récupération des erreurs d'accès web

4. Notifications et sauvegardes :

4.1 Envoi de mails :

- ssh_serveur_mail.py : Rapport quotidien des connexions échouées

4.2 Sauvegarde :

- ssh_cron_backup.py : Sauvegarde des bases de données toutes les 3 heures, avec rétention de 7 sauvegardes

5. Surveillance des ressources systèmes :

5.1 Monitoring :

- ssh_system_status.py : Suivi des ressources (RAM, CPU, disque)
- ssh_system_mail.py : Alerte par e-mail si dépassement des seuils critiques

La Plateforme

5.2 Limitation des alertes :

- Modification pour limiter les alertes à une par heure

6. Mises à jour et automatisation :

6.1 Mises à jour :

- ssh_update.py : Vérification et application des mises à jour (Alcasar), avec notification en cas de redémarrage requis

6.2 Automatisation Google Chat :

- Envoi de rapports périodiques à l'équipe via un espace Google Chat



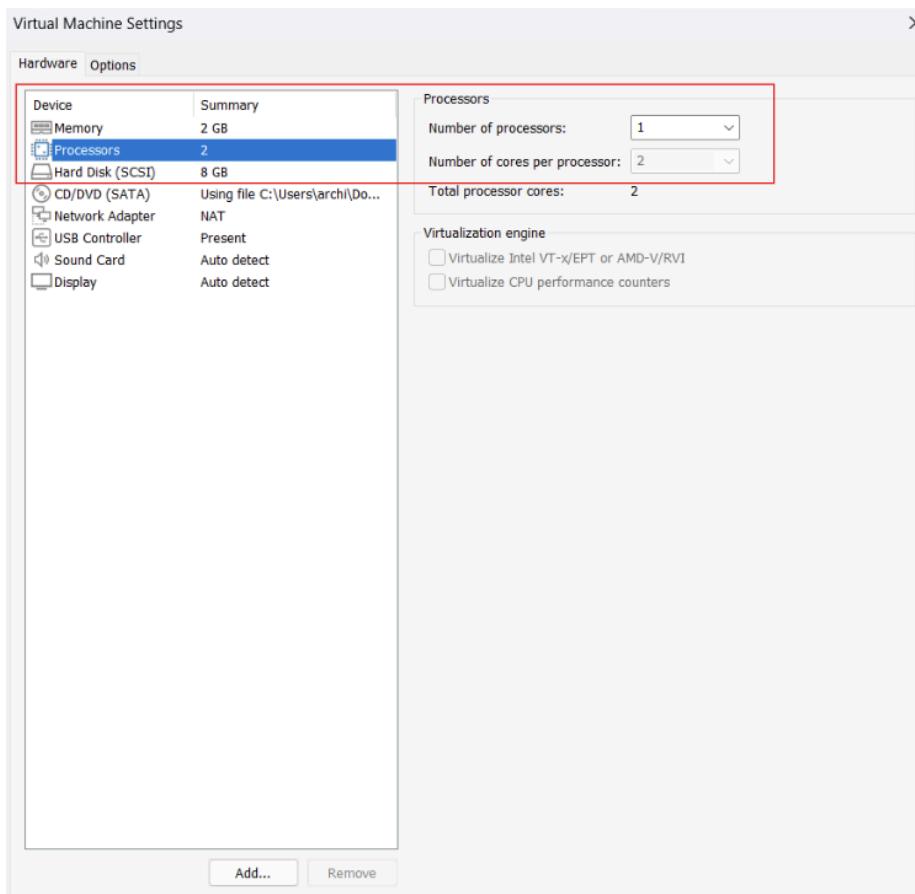
PROJET

1. Création et configuration des environnements virtuels .

1.1 Création des 3 VMs.

Serveur FTP : 1 Go RAM - 1 vCPU - 8 Go disque

On commence par faire la configuration matérielle(Hardware) de notre VM ; en lui attribuant 2 Go de RAM pour que ce soit plus rapide, 1 vCPU et un disque de 8 Go.



La Plateforme

Pourquoi choisir une configuration VM avec 1 processeur et 2 cœurs :

Optimisation des performances multitâches : En configurant la machine virtuelle avec un processeur doté de deux cœurs, on permet une meilleure gestion des tâches parallèles grâce au multi-threading. Cela permet d'améliorer l'efficacité globale pour des charges de travail légères à modérées, tout en maintenant une utilisation raisonnable des ressources en mémoire et en stockage.

Gestion efficace des ressources : Avec 2 Go de RAM, la mémoire disponible est limitée. Opter pour un processeur avec deux cœurs au lieu de plusieurs processeurs permet d'éviter une consommation excessive de RAM. Cette approche assure un équilibre entre les ressources processeur et mémoire, garantissant une bonne stabilité et des performances fluides.

Préservation des ressources de l'hôte : Si la machine virtuelle cohabite avec d'autres VMs ou des applications sur l'hôte, limiter le nombre de processeurs à un seul (avec 2 cœurs) réduit l'impact sur les ressources globales. Cela permet à l'hôte de maintenir de bonnes performances sans risquer la surcharge ou le ralentissement des autres systèmes.

- Lors de l'installation de la VM on ne définit pas de mdp au root et on donne à notre utilisateur les droits “sudo” :

```
monitor@Jftp:/etc/ssh$ su  
Mot de passe :  
su: Échec de l'authentification
```



La Plateforme

- Avant de passer à l'installation de notre serveur FTP on vérifie d'abord que notre adresse ip soit en statique et que notre serveur soit déclaré :

```
GNU nano 7.2                                         interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens33
iface ens33 inet static
    address 192.168.197.134
    netmask 255.255.255.0
    gateway 192.168.197.2
```

```
GNU nano 7.2                                         hosts
127.0.0.1      localhost
127.0.1.1      Jftp
192.168.197.134 Jftp
# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
```



La Plateforme

Création du groupe sftp pour le service sftp :

```
monitor@Jftp:~$ sudo groupadd sftp_user  
monitor@Jftp:~$ mkdir /sftp
```

On se rend dans le fichier config `sshd_config` et on décommente et modifie les lignes suivante :

```
# override default of no subsystems  
#Subsystem      sftp    /usr/lib/openssh/sftp-server  
Subsystem      sftp    internal-sftp  
# Example of overriding settings on a per-user basis  
Match Group sftp  
    X11Forwarding no  
    AllowTcpForwarding no  
#    PermitTTY no  
    ForceCommand internal-sftp
```

Ce paramétrage du fichier SSH définit un environnement SFTP sécurisé pour les membres du groupe sftp. Voici les éléments principaux :

- **Subsystem sftp internal-sftp** : Utilise le sous-système SFTP interne pour gérer les transferts de fichiers, éliminant le besoin d'un processus externe, ce qui améliore la sécurité.
- **Match Group sftp** : Applique des configurations spécifiques aux utilisateurs du groupe sftp.
- **X11Forwarding no** : Désactive le transfert X11, inutile pour les utilisateurs SFTP, afin de réduire les risques potentiels.
- **AllowTcpForwarding no** : Désactive le forwarding TCP pour prévenir d'éventuelles connexions non autorisées.
- **ForceCommand internal-sftp** : Restreint les utilisateurs sftp à l'usage exclusif de SFTP, bloquant ainsi tout accès au shell et l'exécution de commandes SSH.

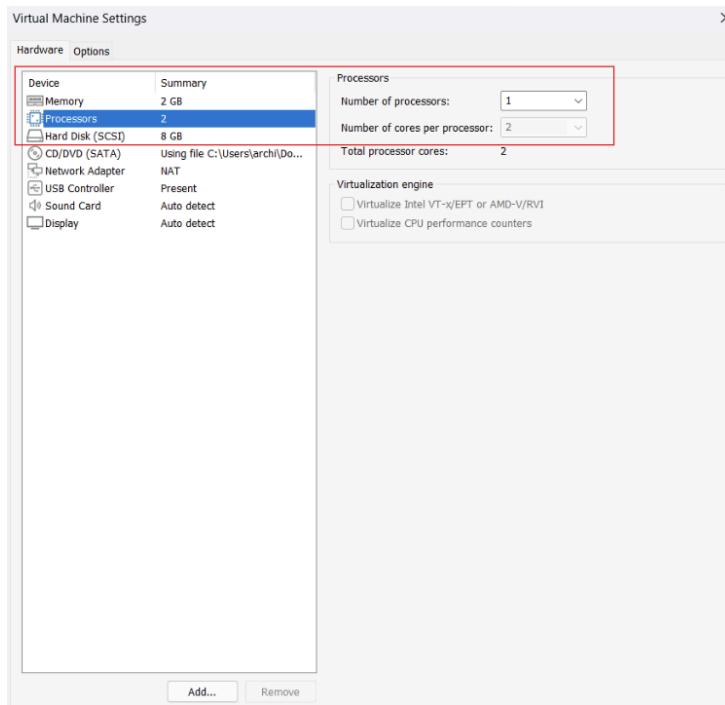
```
PS C:\Users\archi> sftp monitor@192.168.197.134  
Connected to 192.168.197.134.  
sftp>
```



La Plateforme

Serveur WEB : 1 Go RAM - 1 vCPU - 8 Go disque

- On commence par faire la configuration matérielle(Hardware) de notre VM ; en lui attribuant 2 Go de RAM pour que ce soit plus rapide, 1 vCPU et un disque de 8 Go.



- Avant de passer à l'installation de notre serveur WEB on vérifie d'abord que notre adresse ip soit en statique

```
GNU nano 7.2                                         interfaces *
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens33
iface ens33 inet static
    address 192.168.197.135
    gateway 192.168.197.2
```

La Plateforme

- On installe les paquets correspondant à “apache2” :

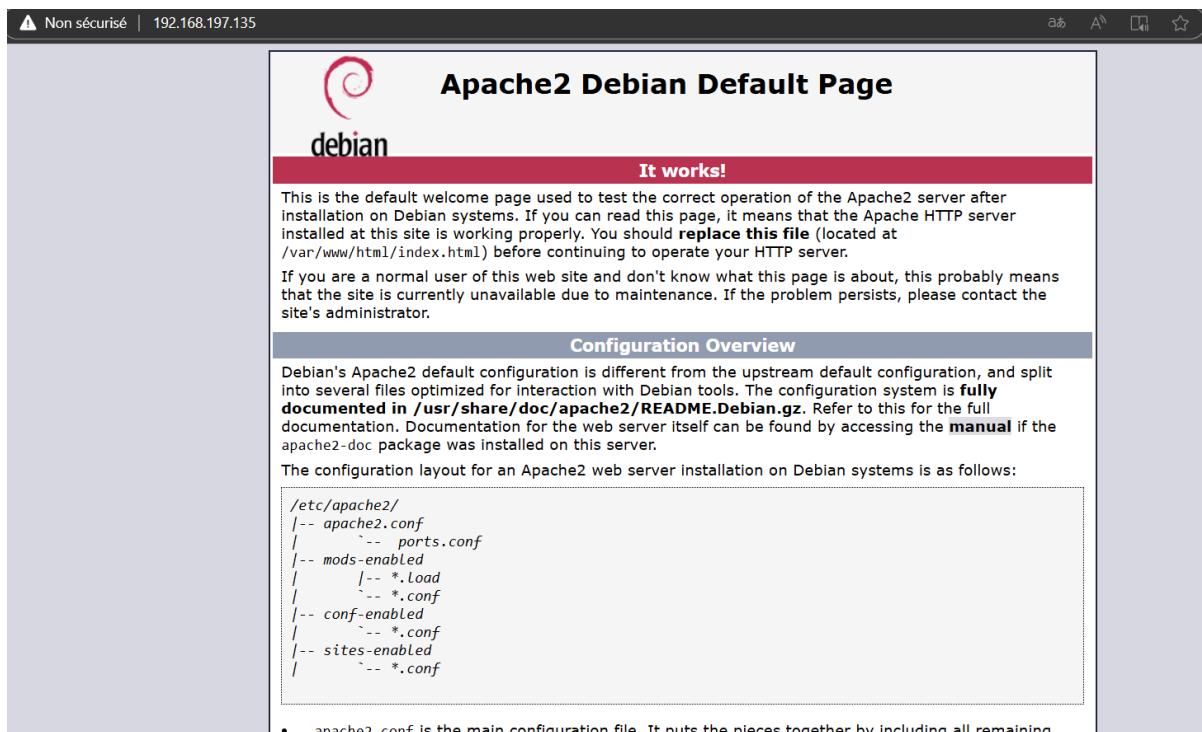
```
monitor@Jweb:/etc/network$ sudo apt install apache2
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap libcurl4
  liblulu5.3-0 ssl-cert
Paquets suggérés :
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser
Les NOUVEAUX paquets suivants seront installés :
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap libcurl4
  liblulu5.3-0 ssl-cert
0 mis à jour, 11 nouvellement installés, 0 à enlever et 0 non mis à jour.
Il est nécessaire de prendre 2 368 ko/2 727 ko dans les archives.
Après cette opération, 9 224 ko d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [0/n] o
Réception de :1 http://deb.debian.org/debian bookworm/main amd64 libcurl4 amd64 7.88.1-10+deb12u7 [390 kB]
Réception de :2 http://deb.debian.org/debian bookworm/main amd64 apache2-bin amd64 2.4.62-1~deb12u1 [1 385 kB]
Réception de :3 cdrom://[[Debian GNU/Linux 12.4.0 _Bookworm_ - Official amd64 DVD Binary-1 with firmware 20231210-17:57]
l...[lines 1-16/16 (END)]
```

- On démarre notre service et on vérifie qui fonctionne :

```
monitor@Jweb:/etc/network$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enabled)
    Active: active (running) since Tue 2024-09-24 12:30:40 CEST; 39s ago
      Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 2250 (apache2)
     Tasks: 55 (limit: 1065)
    Memory: 13.0M
       CPU: 139ms
      CGroup: /system.slice/apache2.service
              └─2250 /usr/sbin/apache2 -k start
                  ├─2251 /usr/sbin/apache2 -k start
                  ├─2252 /usr/sbin/apache2 -k start

sept. 24 12:30:40 Jweb systemd[1]: Starting apache2.service - The Apache HTTP Server...
sept. 24 12:30:40 Jweb apachectl[2249]: AH00558: apache2: Could not reliably determine the server's fully qualified dom...
sept. 24 12:30:40 Jweb systemd[1]: Started apache2.service - The Apache HTTP Server.
[lines 1-16/16 (END)]
```

- On se rend ensuite sur notre machine hôte et on vérifie que apache s'ouvre correctement



La Plateforme

- Nous allons maintenant configurer une authentification de base pour l'utilisateur "monitor", afin de lui permettre de se connecter à notre site web de manière sécurisée.
- Ensuite, si ce n'est pas déjà fait, nous installons le paquet "apache2-utils", qui fournit une série d'outils facilitant la gestion et l'administration d'Apache.

```
monitor@Jweb:/etc/network$ sudo apt-get install apache2-utils
[sudo] Mot de passe de monitor :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
apache2-utils est déjà la version la plus récente (2.4.62-1~deb12u1).
apache2-utils passé en « installé manuellement ».
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

- Cette commande permet d'ajouter l'utilisateur "monitor" ainsi que son mot de passe à la base de données d'authentification.

```
monitor@Jweb:/etc/network$ sudo htpasswd -c /etc/apache2/.htpasswd monitor
New password:
Re-type new password:
Adding password for user monitor
```

- En consultant le contenu du dossier, on peut voir le nom d'utilisateur ainsi que le mot de passe chiffré pour chaque enregistrement.

```
monitor@Jweb:/etc/network$ cat /etc/apache2/.htpasswd
monitor:$apr1$KkhYnyKE$yxIN8VhWH7rTNwbzMzQRu/
```

- Ensuite, nous accédons au fichier de configuration par défaut d'Apache pour y définir les règles d'authentification appliquées au répertoire spécifié : /var/www/html.

```
GNU nano 7.2                                     000-default.conf
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

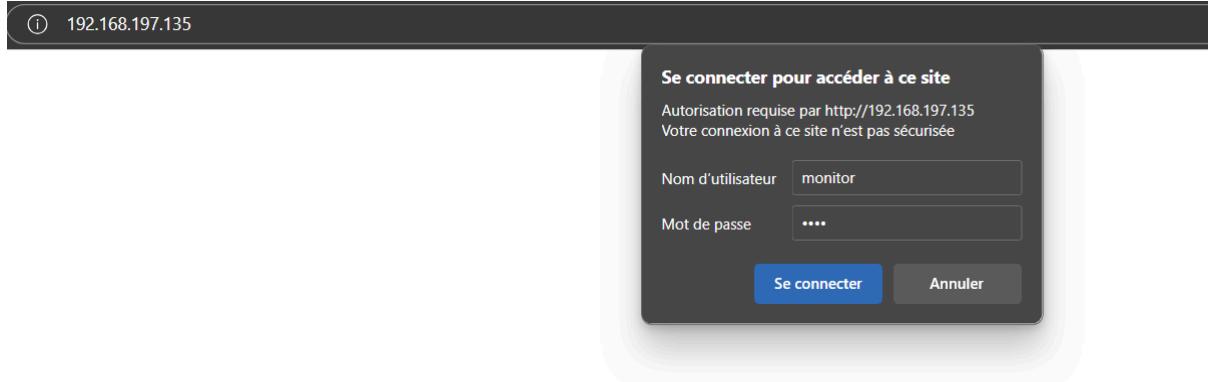
    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    <Directory "/var/www/html">
        AuthType Basic
        AuthName "Restricted Access"
        AuthUserFile /etc/apache2/.htpasswd
        Require valid-user
    </Directory>
    # For most configuration files from conf-available/, which are
```

- Nous vérifions la configuration à l'aide de la commande suivante :

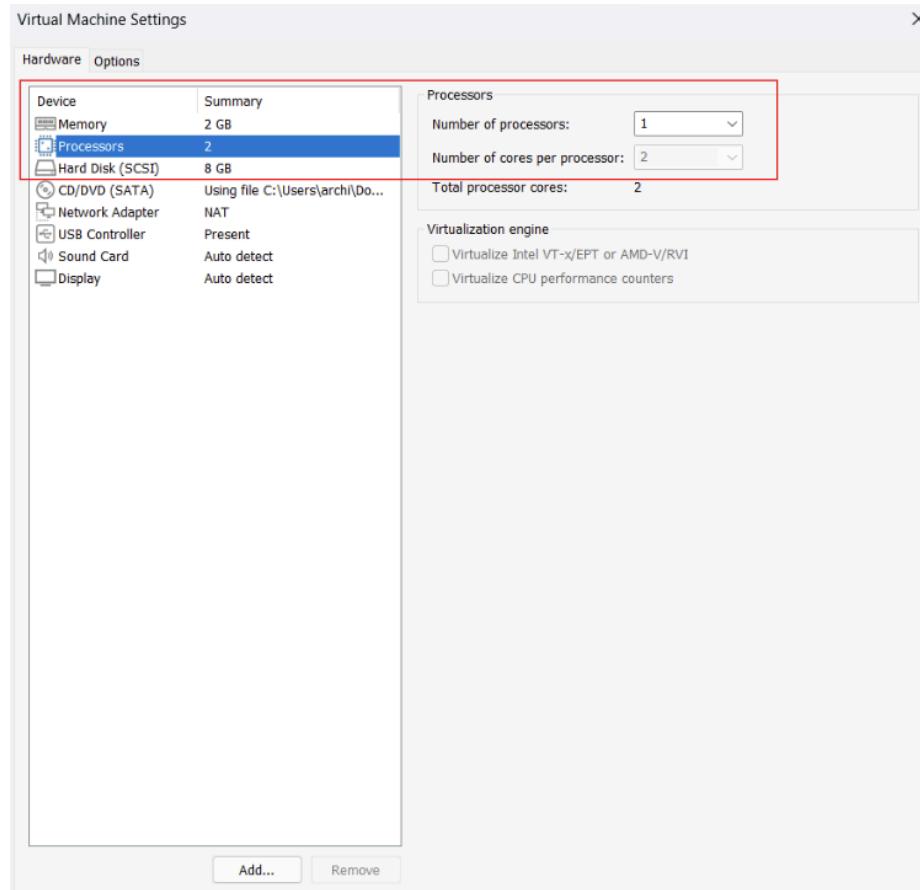
```
monitor@Jweb:/etc/apache2/sites-available$ sudo apachectl configtest
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Syntax OK
```

- N'oubliez pas de redémarrer le service Apache2 une fois la configuration vérifiée. Ensuite, en retournant sur notre navigateur, la page devrait s'afficher de la manière suivante.



Serveur SQL : 2 Go RAM, 2 vCPU, 8 Go disque

- On commence par faire la configuration matérielle(Hardware) de notre VM ; en lui attribuant 2 Go de RAM pour que ce soit plus rapide, 1 vCPU et un disque de 8 Go.



- Avant de commencer on met à jour nos paquets et après on installe MariaDb :

```
monitor@Jmariadb:/etc/apt$ sudo apt update
Atteint :1 http://deb.debian.org/debian bookworm InRelease
Atteint :2 http://deb.debian.org/debian bookworm-updates InRelease
Atteint :3 http://security.debian.org/debian-security bookworm-security InRelease
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Tous les paquets sont à jour.
monitor@Jmariadb:/etc/apt$ sudo apt upgrade -y
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Calcul de la mise à jour... Fait
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

La Plateforme

```
monitor@Jmariadb:/etc/apt$ sudo apt install mariadb-server -y
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  galera=4 gawk libcgifast-perl libcgipm-perl libclone-perl libconfig-inifiles-perl libdaxctl1 libdbd-mariadb-perl
  libdbi-perl libencode-locale-perl libfcgi-bin libfcgi-perl libfcgioldbl libgpmlibhtmlparser-perl
  libhtml-tagset-perl libhtml-template-perl libhttpdate-perl libhttpmessage-perl libio-html-perl
  liblwpmediatypes-perl liblzo2-2 libmariadb3 libmpfr6 libncurses6 libndctl6 libnumal libpmem1 libregexp-ipv6-perl
  libsigsegv2 libsnappy1v5 libtermreadkey-perl libtimedate-perl liburi-perl liburing2 mariadb-client
  mariadb-client-core mariadb-common mariadb-plugin-provider-bzip2 mariadb-plugin-provider-lz4
  mariadb-plugin-provider-lzma mariadb-plugin-provider-lzo mariadb-plugin-provider-snappy mariadb-server-core
  mysql-common psmisc pv rsync socat
Paquets suggérés :
  gawk-doc libldbm-perl libnetdaemon-perl libsqlstatement-perl gpm libdatadump-perl libipcsharedcache-perl
  libbusinessisbn-perl libwwwperl mailx mariadb-test netcatopenbsd docbase python3braceexpand
Les NOUVEAUX paquets suivants seront installés :
  galera=4 gawk libcgifast-perl libcgipm-perl libclone-perl libconfiginifiles-perl libdaxctl1 libdbd-mariadb-perl
  libdbi-perl libencode-locale-perl libfcgi-bin libfcgi-perl libfcgioldbl libgpmlibhtmlparser-perl
  libhtml-tagset-perl libhtmltemplate-perl libhttpdate-perl libhttpmessage-perl libiohtml-perl
  liblwpmediatypes-perl liblzo2-2 libmariadb3 libmpfr6 libncurses6 libndctl6 libnumal libpmem1 libregexp-ipv6-perl
  libsigsegv2 libsnappy1v5 libtermreadkey-perl libtimedate-perl liburi-perl liburing2 mariadb-client
  mariadb-client-core mariadb-common mariadbpluginproviderbzip2 mariadbpluginproviderlz4
  mariadbpluginproviderlzma mariadbpluginproviderlzo mariadbpluginprovidersnappy mariadbserver
  mariadbservercore mysqlcommon psmisc pv rsync socat
0 mis à jour, 50 nouvellement installés, 0 à enlever et 0 non mis à jour.
Il est nécessaire de prendre 20,5 Mo dans les archives.
```

- Démarrer MariaDB et configurer son démarrage automatique au démarrage du système.

```
monitor@Jmariadb:/etc/apt$ sudo systemctl start mariadb
monitor@Jmariadb:/etc/apt$ sudo systemctl status mariadb
● mariadb.service - MariaDB 10.11.6 database server
   Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; preset: enabled)
   Active: active (running) since Tue 2024-09-24 16:28:32 CEST; 39s ago
     Docs: man:mariadb(8)
           https://mariadb.com/kb/en/library/systemd/
   Main PID: 3365 (mariadb)
     Status: "Taking your SQL requests now..."
      Tasks: 13 (limit: 2273)
    Memory: 178.8M
      CPU: 834ms
     CGroup: /system.slice/mariadb.service
             └─3365 /usr/sbin/mariadb

sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Note] InnoDB: log sequence number 45582; transaction ▶
sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/my▶
sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Note] Plugin 'FEEDBACK' is disabled.
sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Warning] You need to use --log-bin to make --expire-l▶
sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Note] Server socket created on IP: '127.0.0.1'.
sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Note] InnoDB: Buffer pool(s) load completed at 240924▶
sept. 24 16:28:32 Jmariadb mariadb[3365]: 2024-09-24 16:28:32 0 [Note] /usr/sbin/mariadb: ready for connections.
sept. 24 16:28:32 Jmariadb mariadb[3365]: Version: '10.11.6-MariaDB-0+deb12u1' socket: '/run/mysqld/mysqld.sock' port:▶
sept. 24 16:28:32 Jmariadb systemd[1]: Started mariadb.service - MariaDB 10.11.6 database server.
sept. 24 16:28:32 Jmariadb /etc/mysql/debian-start[3382]: Upgrading MySQL tables if necessary.
Lines 1-23/23 (END)
```

- Se connecter à MariaDB en tant que root, puis créer l'utilisateur "monitor".

```
monitor@Jmariadb:/etc/apt$ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```



La Plateforme

- Création d'un utilisateur 'monitor' avec accès distant :

```
MariaDB [(none)]> CREATE USER 'monitor'@'%' IDENTIFIED BY 'root';
Query OK, 0 rows affected (0,006 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'monitor'@'%';
Query OK, 0 rows affected (0,003 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,001 sec)

MariaDB [(none)]> exit
Bye
```

- Modifier le fichier de configuration pour autoriser les connexions distantes

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

on remplace bind-address = 127.0.0.1 par : bind-address = 0.0.0.0

```
GNU nano 7.2                                     50-server.cnf

# Broken reverse DNS slows down connections considerably and name resolve is
# safe to skip if there are no "host by domain name" access grants
#skip-name-resolve

# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0

#
# * Fine Tuning
#

#key_buffer_size        = 128M
#max_allowed_packet    = 1G
#thread_stack          = 192K
#thread_cache_size     = 8
# This replaces the startup script and checks MyISAM tables if needed
# the first time they are touched
#myisam_recover_options = BACKUP
#max_connections        = 100
#table_cache            = 64

#
# * Logging and Replication
#
```



La Plateforme

- On teste la connexion distante depuis une autre machine (par exemple, depuis ta machine hôte Windows)

```
PS C:\Users\archi> mysql -u monitor -p -h 192.168.197.136
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 31
Server version: 5.5.5-10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

- Entre chaque modification on n'oublie pas de restart notre service



2. VM dédiée aux scripts Python :

- Debian sans interface graphique, avec Python, MySQL/MariaDB (client uniquement), FTP et outils d'envoi de mails.

- On commence par retourner dans notre VM serveur, nous avons sélectionné la VM nommée "Jftp". Nous allons accéder à cette VM et modifier le fichier `/etc/ssh/sshd_config` pour rétablir l'accès par mot de passe, car sans cela, il sera impossible de configurer la nouvelle clé.

```
GNU nano 7.2                                         sshd_config
# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile      .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
KbdInteractiveAuthentication no
```

La Plateforme

- On génère ensuite les clés public et privé

```
monitor@Jpython:/etc/ssh$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/monitor/.ssh/id_rsa):
/home/monitor/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/monitor/.ssh/id_rsa
Your public key has been saved in /home/monitor/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:1a3Moh67nbS2Vfv3hN0VaUXWHTuj/v7KEaBVeZpnnl4 monitor@Jpython
The key's randomart image is:
+---[RSA 4096]---+
|          .+B|
|         . . .B|
|        . o   o @|
|       o o . o .= *|
|      . + S . .+o|
|     . . . . .o +E|
|    o . . . . = o|
|   . =.+ . .o.o|
|  +o=. . . .+=o|
+---[SHA256]---+
```

Clés publique et privée : différences et utilités

Les clés publiques et privées sont fondamentales dans le chiffrement asymétrique, utilisé pour sécuriser les communications, comme avec SSH.

Clé publique

- Fonction : Chiffre les données.
- Partage : Publique et peut être partagée sans risque.
- Utilité dans SSH : Elle est stockée sur le serveur (dans `~/.ssh/authorized_keys`) et sert à vérifier l'identité du client lors d'une connexion.

Clé privée

- Fonction : Déchiffre les données chiffrées et permet de signer numériquement.
- Sécurité : Doit rester secrète ; sa divulgation permettrait à un tiers d'accéder à vos systèmes.
- Utilité dans SSH : Conservée sur le client (dans `~/.ssh/id_rsa`), elle permet de répondre correctement aux défis du serveur, prouvant ainsi l'identité de l'utilisateur.



Processus SSH

1. Clé publique sur le serveur : ajoutée dans `~/.ssh/authorized_keys`.
2. Clé privée sur le client : gardée secrète.
3. Connexion : le serveur envoie un défi, et le client utilise sa clé privée pour y répondre. Si la réponse est correcte, l'accès est accordé.

En résumé

- Clé publique : chiffrer et partager.
- Clé privée : pour déchiffrer et prouver son identité, doit rester secrète.

- On passe à l'installation de Python :

Pour commencer, nous mettons à jour nos paquets et installons les dépendances nécessaires à l'installation et à la compilation de Python ainsi que des bibliothèques et outils associés.

```
monitor@Jpython:~$ sudo apt install -y build-essential libssl-dev libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm libgdbm-dev liblzma-dev python3-openssl git
[sudo] Mot de passe de monitor :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
build-essential est déjà la version la plus récente (12.9).
```

Ensuite, nous téléchargeons la dernière version de Python en utilisant la commande suivante.

```
monitor@Jpython:~$ wget https://www.python.org/ftp/python/3.11.4/Python-3.11.4.tgz
--2024-09-25 13:14:19--  https://www.python.org/ftp/python/3.11.4/Python-3.11.4.tgz
Résolution de www.python.org (www.python.org)... 199.232.80.223, 2a04:4e42:7d::223
Connexion à www.python.org (www.python.org)|199.232.80.223|:443... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Taille : 26526163 (25M) [application/octet-stream]
Sauvegarde en : « Python-3.11.4.tgz.1 »

Python-3.11.4.tgz.1          100%[=====] 25,30M 17,5MB/s   ds 1,4s
2024-09-25 13:14:21 (17,5 MB/s) - « Python-3.11.4.tgz.1 » sauvegardé [26526163/26526163]
```

Ensuite, nous décompressons notre fichier avec la commande suivante :

```
tar -xvf Python-3.11.4.tgz
```

On teste ensuite si tout fonctionne :

```
monitor@Jpython:~/Python-3.11.4$ python3
Python 3.11.2 (main, Aug 26 2024, 07:20:54) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Salut")
Salut
>>>
```

La Plateforme

Pour sftp Pour sftp, il suffit juste de vérifier si openssh client est bien installé, et on se connecte avec l'adresse ip du serveur sftp que l'on a configuré sur la vm serveur.

```
monitor@Jpython:~$ sftp monitor@192.168.197.134
Connected to 192.168.197.134.
sftp> _
```

- On passe maintenant à la création de notre environnement virtuel :

```
monitor@Jpython:~$ python3 -m venv mon_environnement/
monitor@Jpython:~$ source mon_environnement/bin/activate
(mon_environnement) monitor@Jpython:~$
```

3. Scripts de gestion des serveurs :

3.1 SSH :

- **ssh_login.py : Connexion SSH et exécution de commandes**

- J'ai créé un répertoire "scripts" :

```
monitor@Jpython:~/scripts$ source mon_environnement/bin/activate
(mon_environnement) monitor@Jpython:~/scripts$ ls
mon_environnement ssh_login.py
(mon_environnement) monitor@Jpython:~/scripts$
```

Nous avons utilisé la bibliothèque Paramiko pour gérer les connexions SSH. Le script se connecte à une machine distante (192.168.76.139 dans notre exemple) et exécute une commande (df -h pour afficher l'espace disque). Le retour de la commande est affiché dans le terminal.

```
monitor@Jpython:~$ cat ssh_login.py
import paramiko

# Informations de connexion
hostname = '192.168.76.140' # IP de ton serveur (FTP, Web ou SQL)
username = 'monitor'
key_file = '/home/monitor/id_rsa' # Chemin vers ta clé privée

# Création du client SSH
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

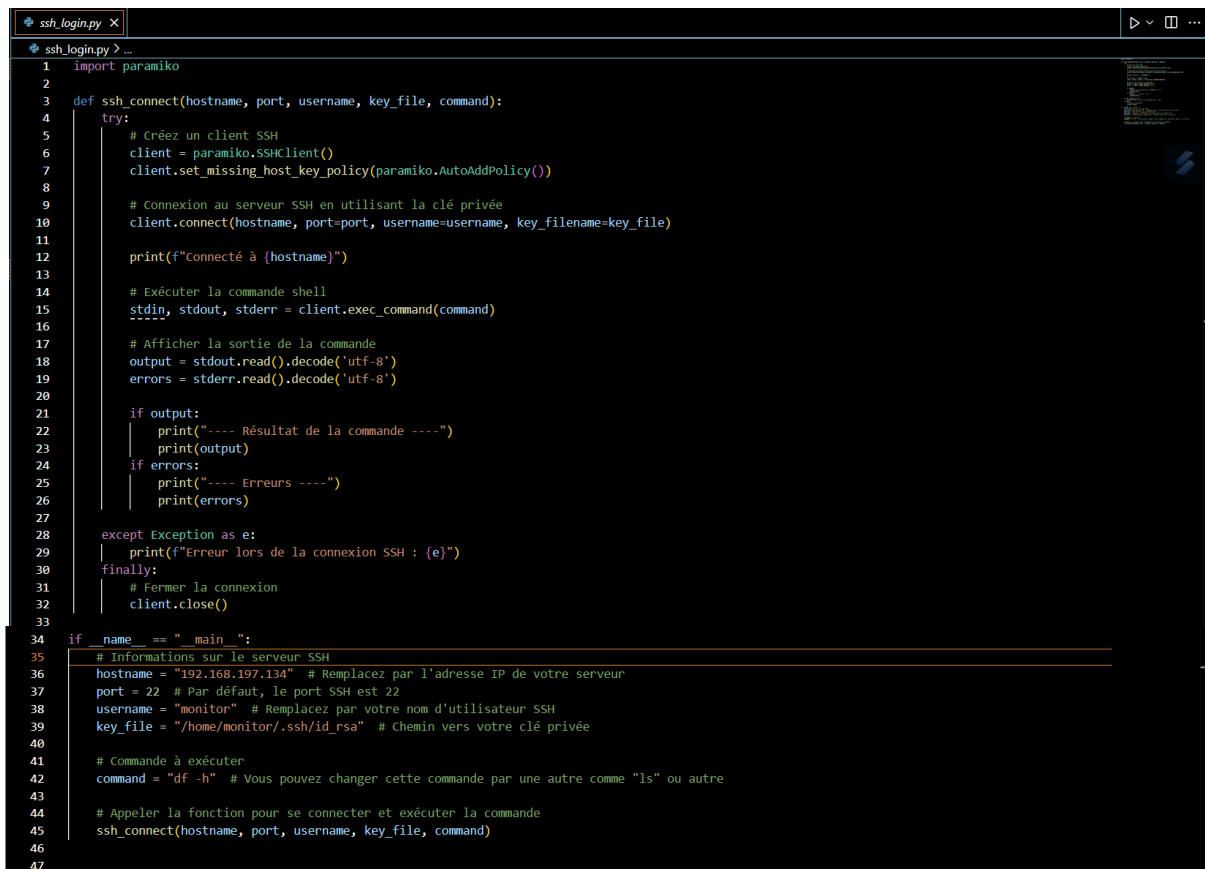
try:
    # Connexion au serveur via clé SSH
    ssh.connect(hostname, username=username, key_filename=key_file)
    print(f"Connexion réussie à {hostname}")

    # Exécution de la commande shell
    stdin, stdout, stderr = ssh.exec_command('df -h')
    print("Résultat de la commande :")
    print(stdout.read().decode())

except Exception as e:
    print(f"Erreur de connexion ou d'exécution : {e}")

finally:
    ssh.close()
```

La Plateforme



```
ssh_login.py > ...
1 import paramiko
2
3 def ssh_connect(hostname, port, username, key_file, command):
4     try:
5         # Créez un client SSH
6         client = paramiko.SSHClient()
7         client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
8
9         # Connexion au serveur SSH en utilisant la clé privée
10        client.connect(hostname, port=port, username=username, key_filename=key_file)
11
12        print(f"Connecté à {hostname}")
13
14        # Exécuter la commande shell
15        stdin, stdout, stderr = client.exec_command(command)
16
17        # Afficher la sortie de la commande
18        output = stdout.read().decode('utf-8')
19        errors = stderr.read().decode('utf-8')
20
21        if output:
22            print("---- Résultat de la commande ----")
23            print(output)
24        if errors:
25            print("---- Erreurs ----")
26            print(errors)
27
28    except Exception as e:
29        print(f"Erreur lors de la connexion SSH : {e}")
30    finally:
31        # Fermer la connexion
32        client.close()
33
34 if __name__ == "__main__":
35     # Informations sur le serveur SSH
36     hostname = "192.168.197.134" # Remplacez par l'adresse IP de votre serveur
37     port = 22 # Par défaut, le port SSH est 22
38     username = "monitor" # Remplacez par votre nom d'utilisateur SSH
39     key_file = "/home/monitor/.ssh/id_rsa" # Chemin vers votre clé privée
40
41     # Commande à exécuter
42     command = "df -h" # Vous pouvez changer cette commande par une autre comme "ls" ou autre
43
44     # Appeler la fonction pour se connecter et exécuter la commande
45     ssh_connect(hostname, port, username, key_file, command)
46
47
```

Connexion SSH (ligne 3-12) : La fonction `ssh_connect()` prend en entrée le nom d'hôte, le port, le nom d'utilisateur, la clé privée, et la commande à exécuter. Elle crée un client SSH avec Paramiko, configure la politique pour accepter les clés d'hôtes non reconnues, puis se connecte au serveur.

Exécution de la commande (ligne 14-21) : Après la connexion SSH, la commande shell est exécutée sur le serveur distant. Le résultat de la commande et les erreurs sont lus respectivement via `stdout` et `stderr`.

Affichage des résultats (ligne 22-27) : Si des résultats sont produits, ils sont affichés. De même, si des erreurs sont présentes, elles sont affichées.

Gestion des exceptions (ligne 28-32) : Si une erreur survient pendant la connexion SSH, un message d'erreur est imprimé. Quoi qu'il arrive, la connexion SSH est fermée grâce à l'instruction `finally`.

Exécution principale (ligne 35-45) : En bas du script, les informations du serveur (IP, port, utilisateur, chemin vers la clé) sont définies, ainsi que la commande à exécuter (par exemple, `df -h` pour vérifier l'espace disque). La fonction `ssh_connect()` est ensuite appelée avec ces paramètres.

- **`stdout`** : C'est le flux par lequel un programme envoie ses sorties normales, c'est-à-dire les résultats de l'exécution des commandes qui se sont bien passées. Par exemple, dans un script qui imprime des données, ces données passent par `stdout`.

La Plateforme

- **stderr** : C'est le flux utilisé pour les sorties d'erreurs. Lorsqu'une erreur survient pendant l'exécution d'un programme ou d'une commande, l'erreur est envoyée à **stderr**. Cela permet de séparer les résultats normaux (qui vont à **stdout**) des messages d'erreur.

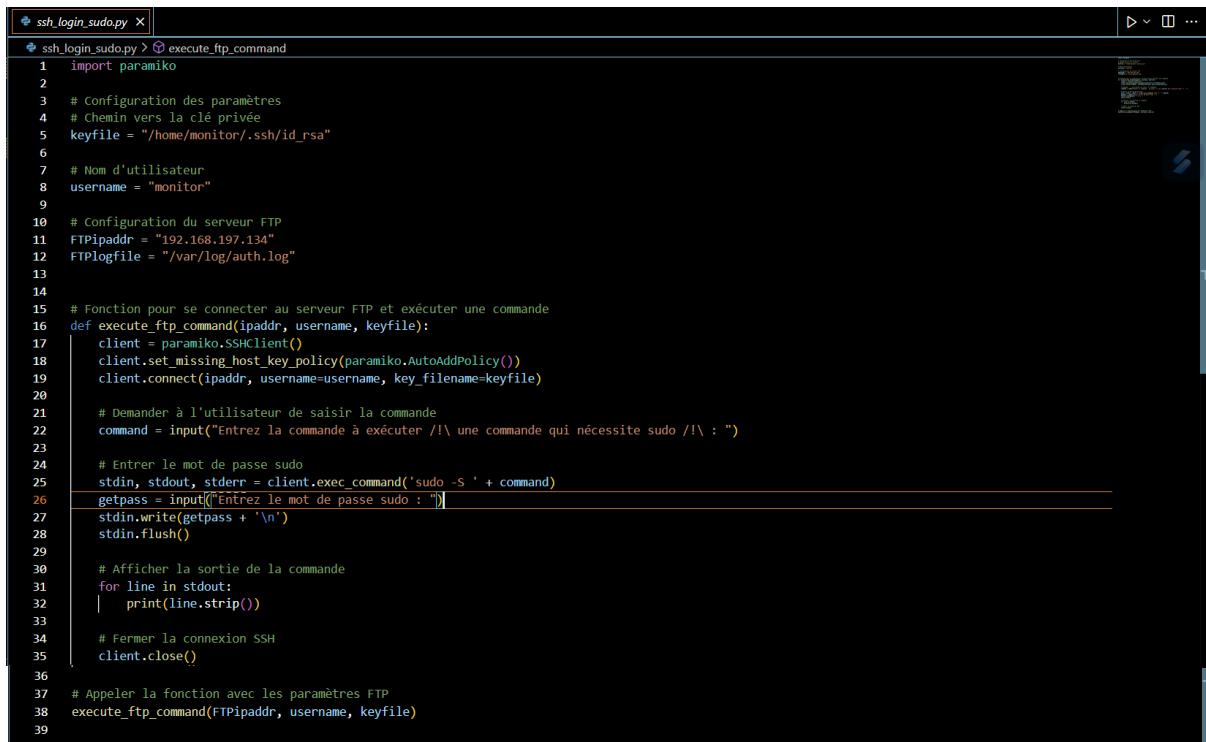
En résumé, **stdout** est utilisé pour afficher les résultats normaux d'une commande, tandis que **stderr** est utilisé pour afficher les erreurs rencontrées.

```
monitor@jypython:~$ python3 ssh_login.py
Connexion réussie à 192.168.76.140
Résultat de la commande :
Sys. de fichiers Taille Utilisé Dispo Uti% Monté sur
udev           944M      0  944M   0% /dev
tmpfs          194M    772K  193M   1% /run
/dev/sda1       19G    2,1G  16G  12% /
tmpfs          967M      0  967M   0% /dev/shm
tmpfs          5,0M      0  5,0M   0% /run/lock
tmpfs          194M      0  194M   0% /run/user/1000
```

```
(mon_environnement) monitor@Jpython:~/scripts$ python ssh_login.py
Connecté à 192.168.197.134
---- Résultat de la commande ----
Sys. de fichiers Taille Utilisé Dispo Uti% Monté sur
udev           948M      0  948M   0% /dev
tmpfs          194M    740K  193M   1% /run
/dev/sda1       6,9G    1,7G  4,8G  27% /
tmpfs          967M      0  967M   0% /dev/shm
tmpfs          5,0M      0  5,0M   0% /run/lock
tmpfs          194M      0  194M   0% /run/user/1000
```

La Plateforme

- `ssh_login_sudo.py` : Connexion SSH avec commandes sudo



```
ssh_login_sudo.py > execute_ftp_command
1 import paramiko
2
3 # Configuration des paramètres
4 # Chemin vers la clé privée
5 keyfile = "/home/monitor/.ssh/id_rsa"
6
7 # Nom d'utilisateur
8 username = "monitor"
9
10 # Configuration du serveur FTP
11 FTPIpaddr = "192.168.197.134"
12 FTPLogfile = "/var/log/auth.log"
13
14
15 # Fonction pour se connecter au serveur FTP et exécuter une commande
16 def execute_ftp_command(ipaddr, username, keyfile):
17     client = paramiko.SSHClient()
18     client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
19     client.connect(ipaddr, username=username, key_filename=keyfile)
20
21     # Demander à l'utilisateur de saisir la commande
22     command = input("Entrez la commande à exécuter /!\ une commande qui nécessite sudo /!\ : ")
23
24     # Entrer le mot de passe sudo
25     stdin, stdout, stderr = client.exec_command('sudo -S ' + command)
26     getpass = input("Entrez le mot de passe sudo : ")
27     stdin.write(getpass + '\n')
28     stdin.flush()
29
30     # Afficher la sortie de la commande
31     for line in stdout:
32         print(line.strip())
33
34     # Fermer la connexion SSH
35     client.close()
36
37 # Appeler la fonction avec les paramètres FTP
38 execute_ftp_command(FTPIpaddr, username, keyfile)
39
```

- **Lignes 1-13 :** Le script importe `paramiko` pour la connexion SSH. Il définit les variables de configuration : le chemin vers la clé privée SSH, le nom d'utilisateur, et l'adresse IP du serveur FTP ainsi que le chemin vers le fichier de log FTP.
- **Lignes 16-20 :** La fonction `execute_ftp_command` crée un client SSH avec Paramiko, accepte automatiquement les nouvelles clés hôtes, et se connecte au serveur avec les paramètres fournis.
- **Lignes 22-25 :** Le script demande à l'utilisateur une commande à exécuter sur le serveur distant nécessitant des droits sudo et lit ensuite le mot de passe sudo pour exécuter cette commande avec `sudo -S`.
- **Lignes 27-31 :** Le script lit les lignes de sortie standard (stdout) de la commande exécutée et les affiche dans le terminal.
- **Ligne 33 :** Après avoir exécuté la commande et affiché les résultats, la connexion SSH est fermée.
- **Lignes 39-41 :** Cette section appelle la fonction `execute_ftp_command` avec les paramètres configurés, ce qui déclenche la connexion SSH et l'exécution de la commande sur le serveur distant.

Pour vérifier que notre script fonctionne correctement il suffit d'aller checker directement le serveur FTP si la commande a bien été exécutée. :

La Plateforme

```
● (mon_environnement) monitor@Jftp:~/scripts$ /home/monitor/scripts/mon_environnement/bin/python /home/monitor/scripts/ssh_login_sudo.py  
Entrez la commande à exécuter !\ une commande qui nécessite sudo !\ : touch test  
Entrez le mot de passe sudo : root
```

```
monitor@Jftp:~$ ls  
test
```



La Plateforme

3.2 MariaDB/MySQL :

- ssh_mysql.py : Vérification des accès au serveur

```
mysqldpy > ...
import paramiko
import getpass

# Fonction pour se connecter à la base de données MariaDB via SSH avec sudo et exécuter une requête SQL
def execute_sudo_mysql_query(hostname, username, private_key_path, db_host, db_user, db_password, query, sudo_password):
    try:
        # Connexion SSH
        key = paramiko.RSAKey.from_private_key_file(private_key_path)
        ssh_client = paramiko.SSHClient()
        ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh_client.connect(hostname=hostname, username=username, pkey=key)
        print(f"Connexion SSH réussie à {hostname}")

        # Exécution de la commande MariaDB avec sudo
        command = f"echo '{sudo_password}' | sudo -S mysql -u {db_user} -p{db_password} -e \"{query}\""
        stdin, stdout, stderr = ssh_client.exec_command(command)

        # Récupération des erreurs et des résultats
        error_output = stderr.read().decode().strip()
        if "incorrect" in error_output or "Désolé" in error_output:
            print("Mot de passe sudo incorrect.")
        else:
            result = stdout.read().decode().strip()
            if result:
                print(f"Résultat de la requête SQL :\n{result}")
            else:
                print("Aucun résultat retourné par la requête SQL.")

        # Fermeture de la connexion SSH
        ssh_client.close()

    except Exception as e:
        print(f"Erreur SSH ou autre : {e}")

# Paramètres de la connexion SSH
hostname = "192.168.76.140" # Adresse IP de la VM MariaDB
username = "monitor" # Nom d'utilisateur pour SSH
private_key_path = "/home/monitor/id_rsa" # Chemin vers la clé privée

# Paramètres de la connexion MariaDB
db_host = "192.168.76.140" # MariaDB est généralement installé sur localhost
db_user = "monitor" # Utilisateur MariaDB
db_password = "root" # Mot de passe pour l'utilisateur MariaDB

# Requête SQL à exécuter
query = "SHOW DATABASES;"

# Demande du mot de passe sudo
sudo_password = getpass.getpass("Entrez le mot de passe sudo pour l'utilisateur 'monitor' : ")

# Exécution du script
execute_sudo_mysql_query(hostname, username, private_key_path, db_host, db_user, db_password, query, sudo_password)
```

Ce script est conçu pour se connecter à un serveur MariaDB via SSH, exécuter une commande SQL en utilisant des privilèges sudo, et afficher les résultats de la requête ou les erreurs.

Connexion SSH (lignes 7-14) : La fonction `execute_sudo_mysql_query()` se connecte au serveur SSH en utilisant Paramiko, la clé privée, et le nom d'utilisateur spécifié. La connexion SSH est établie avec succès après vérification de la clé hôte.

La Plateforme

Exécution de la commande MariaDB avec sudo (lignes 17-21) : Une commande MySQL est exécutée sur le serveur distant en utilisant sudo, après avoir passé le mot de passe sudo.

Affichage des résultats (lignes 24-29) : Si des erreurs surviennent, elles sont affichées. Si la requête SQL produit des résultats, ceux-ci sont affichés. Sinon, un message indiquant qu'aucun résultat n'a été retourné est affiché.

Gestion des exceptions (lignes 32-34) : Si une erreur se produit lors de la connexion SSH ou de l'exécution de la commande, elle est affichée.

Exécution principale (lignes 37-47) : Les paramètres du serveur SSH, de la base de données, et la requête SQL sont définis. La fonction `execute_sudo_mysql_query()` est ensuite appelée avec ces paramètres pour exécuter la requête SQL via SSH.

```
(myenv) monitor@jython:~$ python3 ssh_mysql.py
Entrez le mot de passe sudo pour l'utilisateur 'monitor' :
Connexion SSH réussie à 192.168.76.140
Résultat de la requête SQL :
Database
DB_access
information_schema
logs_db
mysql
performance_schema
sys
(myenv) monitor@jython:~$ |
```



La Plateforme

- ssh_mysql_error.py : Récupération des erreurs de connexion

Script : config.py

```
❸ config.py > ...  
1  #chemin vers la clé privée  
2  keyfile = "/home/monitor/.ssh/id_rsa"  
3  
4  #Nom d'utilisateur  
5  username = "monitor"  
6  
7  #Configuration du serveur FTP & WEB  
8  FTPipaddr = "192.168.76.139"  
9  WEBipaddr = "192.168.76.138"  
10 #Configuration du serveur mariadb  
11 DBipaddr = "192.168.76.140"  
12 DBpassword = "root"  
13 DBname = "DB_access"  
14 #SMTP  
15 smtp_server = "smtp.gmail.com"  
16 smtp_port = 587  
17 smtp_user = "dest.userjr@gmail.com"  
18 smtp_password = "fjzu gjmd gtci yrccs"  
19 mail_dest ="jordan.reinaldo@laplateforme.io"  
20 #sudo  
21 sudo_password = "root"
```



La Plateforme

Script : ssh_mysql_error.py :

```
ssh_mysql_error.py > ...
import re
import pymysql
import paramiko
from datetime import datetime
import config

# Configuration de la connexion à la base de données
db = pymysql.connect(host=config.DBipaddr, user=config.username, password=config.DBpassword, database=config.DBname)

# Configuration de la connexion SSH
ssh_host = config.DBipaddr # L'adresse IP ou le nom d'hôte du serveur distant
ssh_user = config.username # Le nom d'utilisateur SSH
ssh_password = config.keyfile # Le mot de passe SSH
log_file_path = '/var/log/mysql/mysql.log' # Chemin vers le fichier de log sur le serveur distant

# Connexion SSH
ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(ssh_host, username=ssh_user, password=ssh_password)

# Lecture du fichier de log sur le serveur distant
stdin, stdout, stderr = ssh_client.exec_command(f'cat {log_file_path}')
log_content = stdout.read().decode('utf-8')

# Traitement du contenu du fichier de log pour "Access Denied"
connections = []
lines = log_content.splitlines()
i = 0

while i < len(lines) - 1:
    current_line = lines[i]
    next_line = lines[i + 1]

    # Vérifie si la ligne courante contient une connexion et la suivante un "Access denied"
    if "Connect" in current_line and "Access denied" in next_line:
        connect_match = re.match(r"(\d{6}) (\d{2}:\d{2}:\d{2})\s+\d+\s+Connect\s+(\w+)\@([\w.-]+)\.", current_line)
        if connect_match:
            date_str, time_str, username, ip_address = connect_match.groups()
            # Convertir le format de date en YYYY-MM-DD HH:MM:SS
            attempt_time = datetime.strptime(f"{date_str} {time_str}", "%Y-%m-%d %H:%M:%S").strftime("%Y-%m-%d %H:%M:%S")
            connections.append((username, attempt_time, ip_address, 'failed'))

    # Passe à la prochaine paire de lignes
    i += 1

# Fonction pour insérer les tentatives de connexion échouées dans la base de données
def insert_login_attempts(connections):
    with db.cursor() as cursor:
        for connection in connections:
            cursor.execute("INSERT INTO access_logs (username, attempt_time, ip_address, status) VALUES (%s, %s, %s, %s)", connection)
            print(f"Insertion: {connection}") # Debug: Montre chaque connexion à insérer
    db.commit()
    print(f"{len(connections)} enregistrements ont été insérés avec succès.")

# Insérer les connexions dans la base de données
insert_login_attempts(connections)

# Fermeture de la connexion
ssh_client.close()
db.close()
```

Le script ci-dessus effectue un travail complet d'extraction et d'insertion des logs de connexions échouées depuis un serveur distant vers une base de données, assurant ainsi une traçabilité des événements de sécurité.



La Plateforme

Connexion à la base de données (lignes 7-8) :

Il se connecte à la base MariaDB/MySQL en utilisant `pymysql.connect` avec les informations de connexion (hôte, utilisateur, mot de passe, base de données).

Configuration SSH (lignes 11-18) :

Il configure la connexion SSH à un serveur distant, utilise `paramiko` pour ouvrir une connexion SSH, et exécute la commande pour lire les fichiers de logs sur le serveur distant.

Traitement du fichier de log (lignes 21-26) :

Le contenu du fichier log est analysé, ligne par ligne, pour détecter les tentatives de connexion échouées ("Access Denied"). Il stocke les connexions pertinentes dans une liste `connections`.

Insertion des tentatives de connexion dans la base de données (lignes 41-47) :

Chaque tentative échouée est insérée dans la table `access_logs` avec les détails de l'utilisateur, le temps de tentative, l'adresse IP et l'état.

Fermeture des connexions (lignes 50-52) :

Enfin, il ferme la connexion SSH ainsi que la connexion à la base de données après avoir inséré les informations.

```
(myenv) monitor@jypython:~$ python3 ssh_mysql_error.py
Insertion: ('fake_user', '2024-09-25 16:12:44', 'localhost', 'failed')
Insertion: ('test2fake', '2024-09-26 12:49:38', 'localhost', 'failed')
Insertion: ('monitor', '2024-09-27 13:29:45', 'localhost', 'failed')
Insertion: ('monitor', '2024-09-30 15:20:12', 'localhost', 'failed')
Insertion: ('rijaseigneurkebabtoutpuissant', '2024-09-30 15:45:50', 'localhost', 'failed')
Insertion: ('test', '2024-09-30 17:56:58', 'localhost', 'failed')
Insertion: ('test', '2024-09-30 17:57:10', 'localhost', 'failed')
Insertion: ('fake_user', '2024-09-30 18:00:37', 'jymariadb', 'failed')
Insertion: ('fake_user', '2024-09-30 18:02:15', '192.168.76.141', 'failed')
9 enregistrements ont été insérés avec succès.
(myenv) monitor@jypython:~$ |
```

37	fake_user	2024-09-25	16:12:44	localhost	failed
38	test2fake	2024-09-26	12:49:38	localhost	failed
39	monitor	2024-09-27	13:29:45	localhost	failed
40	monitor	2024-09-30	15:20:12	localhost	failed
41	rijaseigneurkebabtoutpuissant	2024-09-30	15:45:50	localhost	failed
42	test	2024-09-30	17:56:58	localhost	failed
43	test	2024-09-30	17:57:10	localhost	failed
44	fake_user	2024-09-30	18:00:37	jymariadb	failed
45	fake_user	2024-09-30	18:02:15	192.168.76.141	failed



La Plateforme

3.3 FTP :

- ssh_ftp_error.py : Extraction des logs d'erreurs FTP

```
❶ ssh_ftp_error.py > ...
1  import re
2  import pymysql
3  import paramiko
4  from datetime import datetime
5  import config
6
7  # Configuration de la connexion à la base de données
8  db = pymysql.connect(host=config.DBipaddr, user=config.username, password=config.DBpassword, database=config.DBname)
9
10 # Configuration de la connexion SSH pour accéder au serveur FTP
11 ssh_host = config.FTPipaddr # L'adresse IP du serveur FTP
12 ssh_user = config.username # Le nom d'utilisateur SSH
13 ssh_keyfile = config.keyfile # Chemin vers la clé privée
14 log_file_path = '/var/log/vsftpd.log.1' # Chemin vers le fichier de log de vsftpd sur le serveur FTP
15
16 # Connexion SSH au serveur FTP
17 ssh_client = paramiko.SSHClient()
18 ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
19 ssh_client.connect(ssh_host, username=ssh_user, key_filename=ssh_keyfile)
20
21 # Lecture du fichier de log sur le serveur distant
22 stdin, stdout, stderr = ssh_client.exec_command(f"cat {log_file_path}")
23 log_content = stdout.read().decode('utf-8')
24
25 # Traitement du contenu du fichier de log pour les échecs de connexion FTP
26 connections = []
27 lines = log_content.splitlines()
28
29 for line in lines:
30     # Vérifie si la ligne contient un échec de connexion FTP
31     if "FAIL LOGIN" in line:
32         log_match = re.match(r"(.*)\s+[\pid\s+\d+\s+\[(\w-+)\]\s+FAIL LOGIN:\s+Client\s+::ffff:([\d\.\.]+)\s+", line)
33         if log_match:
34             date_str, username, ip_address = log_match.groups()
35             # Convertir la date en format YYYY-MM-DD HH:MM:SS
36             attempt_time = datetime.strptime(date_str, "%a %b %d %H:%M:%S %Y").strftime("%Y-%m-%d %H:%M:%S")
37             connections.append((username, attempt_time, ip_address, 'failed'))
38
39 # Fonction pour insérer les tentatives de connexion échouées dans la base de données
40 def insert_ftp_login_attempts(connections):
41     with db.cursor() as cursor:
42         for connection in connections:
43             cursor.execute("INSERT INTO ftp_access_logs (username, attempt_time, ip_address, status) VALUES (%s, %s, %s, %s)", connection)
44             print(f"Insertion: {connection}") # Debug: Montre chaque connexion à insérer
45     db.commit()
46     print(f"{len(connections)} enregistrements ont été insérés avec succès.")
47
48 # Insérer les connexions dans la base de données
49 insert_ftp_login_attempts(connections)
50
51 # Fermeture des connexions
52 ssh_client.close()
53 db.close()
```

Ce script est conçu pour extraire et stocker les tentatives de connexion échouées depuis un serveur FTP distant. Voici sa structure :

1. Connexion à la base de données (lignes 6-8) : Le script se connecte à une base de données MySQL en utilisant les identifiants fournis.
2. Connexion SSH (lignes 10-19) : Une connexion SSH est établie avec le serveur FTP, permettant l'accès au fichier de log.
3. Lecture des logs (lignes 21-23) : Le contenu du fichier de log FTP est lu et décodé.
4. Traitement des logs (lignes 25-40) : Chaque ligne est analysée pour détecter les échecs de connexion via l'expression régulière. Les informations sur l'échec (date, IP, utilisateur) sont extraites et stockées.
5. Insertion dans la base de données (lignes 42-52) : Les tentatives de connexion échouées sont insérées dans la base de données.
6. Fermeture des connexions (lignes 53-54) : Les connexions SSH et à la base de données sont fermées pour terminer proprement le script.

La Plateforme

```
(myenv) monitor@jupyter:~$ python3 ssh_ftp_error.py
Insertion: ('test', '2024-09-30 16:31:16', '192.168.76.139', 'failed')
Insertion: ('fake', '2024-09-30 16:48:46', '192.168.76.141', 'failed')
2 enregistrements ont été insérés avec succès.
```

```
MariaDB [DB_access]> select * from ftp_access_logs;
+----+-----+-----+-----+-----+
| id | username | attempt_time | ip_address | status |
+----+-----+-----+-----+-----+
| 1 | test     | 2024-09-30 16:31:16 | 192.168.76.139 | failed
| 2 | test     | 2024-09-30 16:31:16 | 192.168.76.139 | failed
| 3 | fake     | 2024-09-30 16:48:46 | 192.168.76.141 | failed
```



3.4 Web :

- ssh_web_error.py : Récupération des erreurs d'accès web

```
❶ ssh_web_error.py > ...
1  import re
2  import pymysql
3  import paramiko
4  from datetime import datetime
5  import config
6
7  # Configuration de la connexion à la base de données
8  db = pymysql.connect(host=config.DBipaddr, user=config.username, password=config.DBpassword, database=config.DBname)
9
10 # Configuration de la connexion SSH pour accéder au serveur Web
11 ssh_host = config.WEBipaddr # L'adresse IP du serveur Web
12 ssh_user = config.username # Le nom d'utilisateur SSH
13 ssh_keyfile = config.keyfile # Chemin vers la clé privée
14 log_file_path = '/var/log/apache2/error.log' # Chemin vers le fichier de log du serveur Apache
15
16 # Connexion SSH au serveur Web
17 ssh_client = paramiko.SSHClient()
18 ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
19 ssh_client.connect(ssh_host, username=ssh_user, key_filename=ssh_keyfile)
20
21 # Lecture du fichier de log sur le serveur distant
22 stdin, stdout, stderr = ssh_client.exec_command(f"cat {log_file_path}")
23 log_content = stdout.read().decode('utf-8')
24
25 # Traitement du contenu du fichier de log pour les erreurs d'accès
26 connections = []
27 lines = log_content.splitlines()
28
29 for line in lines:
30     print(f"Analyzing log line: {line}") # Debug: Affiche chaque ligne analysée
31     # Vérifie si la ligne contient une erreur d'accès liée à l'authentification
32     if "auth_basic:error" in line:
33         print("Line contains 'auth_basic:error'") # Debug: indique que la ligne contient l'erreur attendue
34         # Expression régulière simplifiée pour capturer les informations clés
35         log_match = re.match(r"\[(.*?\]\.\*\[\!client ([\d\.\.]+)\:\d+\] AH01618: user (\w+) not found\.*", line)
36         if log_match:
37             print("Regex matched the line successfully") # Debug: indique que la regex a trouvé une correspondance
38             date_str, ip_address, username = log_match.groups()
39             print(f"Matched groups - Date: {date_str}, IP: {ip_address}, Username: {username}") # Debug: Affiche les valeurs extraites
40
41             # Convertir la date en format YYYY-MM-DD HH:MM:SS
42             try:
43                 attempt_time = datetime.strptime(date_str, "%a %b %d %H:%M:%S %Y").strftime("%Y-%m-%d %H:%M:%S")
44             except ValueError:
45                 attempt_time = datetime.strptime(date_str, "%a %b %d %H:%M:%S %Y").strftime("%Y-%m-%d %H:%M:%S")
46             connections.append((username, attempt_time, ip_address, 'failed'))
47             print(f"Captured failed login attempt: {username}, {attempt_time}, {ip_address}") # Debug: Affiche les connexions capturées
48         else:
49             print("Regex did not match the line") # Debug: indique que la regex n'a pas trouvé de correspondance
50
51     # Fonction pour insérer les tentatives de connexion échouées dans la base de données
52     def insert_web_login_attempts(connections):
53         with db.cursor() as cursor:
54             for connection in connections:
55                 cursor.execute("INSERT INTO web_access_logs (username, attempt_time, ip_address, status) VALUES (%s, %s, %s, %s)", connection)
56                 print(f"Insertion: {connection}") # Debug: Montre chaque connexion à insérer
57             db.commit()
58             print(f"{len(connections)} enregistrements ont été insérés avec succès.")
59
60     # Insérer les connexions dans la base de données
61     insert_web_login_attempts(connections)
62
63     # Fermeture des connexions
64     ssh_client.close()
65     db.close()
```

La Plateforme

Ce script est conçu pour extraire et enregistrer dans une base de données les erreurs d'accès du serveur web Apache via SSH. Voici son fonctionnement détaillé :

- Connexion à la base de données (ligne 7-9) : Le script se connecte à une base de données MySQL/MariaDB via `pymysql`.
- Connexion SSH (ligne 11-16) : Il établit une connexion SSH au serveur web Apache avec Paramiko pour récupérer les logs d'accès.
- Récupération des logs (ligne 20-22) : Le script exécute la commande `cat` sur le fichier de logs Apache et lit son contenu.
- Traitement des logs (ligne 24-52) : Il analyse ligne par ligne le contenu des logs pour identifier les erreurs "user not found" liées à l'authentification, en utilisant des expressions régulières pour extraire les informations pertinentes (date, adresse IP, utilisateur).
- Insertion dans la base de données (ligne 54-58) : Les tentatives d'accès échouées sont insérées dans une table de la base de données, accompagnées de la date, de l'heure, de l'adresse IP, du nom d'utilisateur, et du statut.
- Fermeture des connexions (ligne 60-61) : Le script ferme la connexion SSH et la connexion à la base de données.

```
(myenv) monitor@mypython:~/ssh_web_error.py
Analyzing log line: [Sun Oct 06 18:06:48 424227 2024] [mpm_event:notice] [pid 694:tid 694] AH00489: Apache/2.4.62 (Debian) configured -- resuming normal operations
Analyzing log line: [Sun Oct 06 18:06:48 424278 2024] [core:notice] [pid 694:tid 694] AH00094: Command line: '/usr/sbin/apache2'
Analyzing log line: [Sun Oct 06 18:21:23.001426 2024] [auth_basic:error] [pid 907:tid 924] [client 192.168.76.1:50483] AH01618: user youcefseigneurkebab not found: /
Line contains 'auth_basic:error'
Regex matched the line successfully
Matched groups - Date: Sun Oct 06 18:21:23.001426 2024, IP: 192.168.76.1, Username: youcefseigneurkebab
Captured failed login attempt: youcefseigneurkebab, 2024-10-06 18:21:23, 192.168.76.1
Insertion: ('youcefseigneurkebab', '2024-10-06 18:21:23', '192.168.76.1', 'failed')
1 enregistrement ont été insérés avec succès.
```

```
MariaDB [DB_access]> select * from web_access_logs;
+----+-----+-----+-----+-----+
| id | username          | attempt_time      | ip_address        | status   |
+----+-----+-----+-----+-----+
| 1  | test              | 2024-09-30 17:04:59 | 192.168.76.1     | failed   |
| 2  | test              | 2024-09-30 17:04:59 | 192.168.76.1     | failed   |
| 3  | fake              | 2024-09-30 17:34:21 | 192.168.76.1     | failed   |
| 4  | test              | 2024-09-30 17:04:59 | 192.168.76.1     | failed   |
| 5  | fake              | 2024-09-30 17:34:21 | 192.168.76.1     | failed   |
| 6  | rija              | 2024-10-02 13:53:09 | 192.168.76.1     | failed   |
| 7  | youcefseigneurkebab | 2024-10-06 18:21:23 | 192.168.76.1     | failed   |
+----+-----+-----+-----+-----+
7 rows in set (0,001 sec)
```



4. Notifications et sauvegardes :

4.1 Envoi de mails :

- ssh_serveur_mail.py : Rapport quotidien des connexions échouées

```
1  ✓ import pymysql
2   import smtplib
3   from email.mime.text import MIMEText
4   from datetime import datetime, timedelta
5   import config
6
7   # Connexion à la base de données MySQL
8   ✓ db = pymysql.connect(
9       host=config.DBipaddr,
10      user=config.username,
11      password=config.DBpassword,
12      database=config.DBname
13  )
14
15  # Calcul de la date d'hier
16  yesterday = (datetime.now() - timedelta(days=1)).strftime('%Y-%m-%d')
17
18  # Fonction pour récupérer les tentatives de connexion échouées par machine
19  ✓ def get_failed_logins():
20    ✓ failed_logins = {
21        "web": [],
22        "ftp": [],
23        "mariadb": []
24    }
25
26  ✓ with db.cursor() as cursor:
27      # Récupérer les logs de la machine web
28      ✓ query_web = """
29          SELECT username, attempt_time, ip_address, status
30          FROM web_access_logs
31          WHERE DATE(attempt_time) = %s;
32          """
33
34      cursor.execute(query_web, (yesterday,))
35      failed_logins["web"] = cursor.fetchall()
36
37  ✓      # Récupérer les logs de la machine ftp
38      ✓ query_ftp = """
39          SELECT username, attempt_time, ip_address, status
40          FROM ftp_access_logs
41          WHERE DATE(attempt_time) = %s;
42          """
43
44      cursor.execute(query_ftp, (yesterday,))
45      failed_logins["ftp"] = cursor.fetchall()
46
47  ✓      # Récupérer les logs de la machine mariadb
48      ✓ query_mariadb = """
49          SELECT username, attempt_time, ip_address, status
50          FROM access_logs
51          WHERE DATE(attempt_time) = %s;
52          """
53
54      cursor.execute(query_mariadb, (yesterday,))
55      failed_logins["mariadb"] = cursor.fetchall()
56
57  return failed_logins
```

La Plateforme

```
55
56     # Fonction pour envoyer un e-mail avec le rapport
57     def send_email(report):
58         # Création de l'e-mail
59         msg = MIMEText(report)
60         msg['Subject'] = f"Rapport des tentatives de connexion échouées du {yesterday}"
61         msg['From'] = config.smtp_user
62         msg['To'] = config.mail_dest
63
64         # Envoi de l'e-mail via SMTP
65         try:
66             with smtplib.SMTP(config.smtp_server, config.smtp_port) as server:
67                 server.starttls() # Sécuriser la connexion
68                 server.login(config.smtp_user, config.smtp_password)
69                 server.sendmail(config.smtp_user, config.mail_dest, msg.as_string())
70                 print("E-mail envoyé avec succès.")
71         except Exception as e:
72             print(f"Erreur lors de l'envoi de l'e-mail : {e}")
73
74     # Récupérer les tentatives échouées de la veille par machine
75     failed_logins = get_failed_logins()
76
77     # Générer le rapport par machine
78     report = ""
79     machines = ["web", "ftp", "mariadb"]
80     for machine in machines:
81         logins = failed_logins[machine]
82
83         if logins:
84             report += f"Tentatives de connexion échouées sur machine {machine} du {yesterday} :\n\n"
85             report += "Utilisateur\tDate/Heure\tAdresse IP\tStatut\n"
86             report += "-" * 50 + "\n"
87             for login in logins:
88                 report += f"\t{login[0]}\t{login[1]}\t{login[2]}\t{login[3]}\n"
89             report += "\n"
90         else:
91             report += f"Aucune tentative de connexion échouée sur la machine {machine} du {yesterday}.\n\n"
92
93     # Envoi de l'e-mail avec le rapport
94     send_email(report)
95
96     # Fermeture de la connexion à la base de données
97     db.close()
```

Le but de ce script est de récupérer les tentatives de connexion échouées de trois serveurs (web, FTP et MariaDB) pour la veille, de générer un rapport détaillé des échecs de connexion, puis d'envoyer ce rapport par email à l'administrateur système.

Il collecte les tentatives échouées depuis les journaux d'accès de chaque machine, compile les résultats et les présente dans un format lisible pour l'administrateur, incluant les informations comme l'utilisateur, l'heure, l'adresse IP, et le statut de la tentative.



La Plateforme

Ce script se divise en plusieurs parties clés :

1. Connexion à la base de données MySQL (lignes 6-12) : Le script établit une connexion avec la base de données MySQL en utilisant `pymysql` avec les identifiants stockés dans un fichier de configuration.
2. Calcul de la date d'hier (ligne 14) : Il calcule la date de la veille pour filtrer les tentatives de connexion échouées qui ont eu lieu uniquement la veille.
3. Fonction `get_failed_logins()` (lignes 19-53) : Cette fonction extrait les tentatives de connexion échouées des logs de trois serveurs : web, FTP, et MariaDB. Pour chaque machine, elle exécute une requête SQL afin de récupérer les tentatives échouées dans les 24 heures précédentes et stocke les résultats dans un dictionnaire `failed_logins`.
4. Fonction `send_email(report)` (lignes 56-73) : Elle envoie un email contenant un rapport des tentatives de connexion échouées par machine. L'email est généré à partir d'un objet `MIMEText` et envoyé via le serveur SMTP configuré.
5. Récupération des connexions échouées (lignes 75-77) : Cette partie du code appelle la fonction `get_failed_logins()` pour obtenir les données des tentatives de connexion échouées de chaque machine pour la veille.
6. Génération du rapport (lignes 79-92) : Le script parcourt chaque machine (web, FTP, mariadb) et génère un rapport détaillant les tentatives de connexion échouées pour chaque serveur. Si aucune tentative n'est trouvée pour une machine, un message indiquant l'absence de tentatives est ajouté au rapport.
7. Envoi du rapport (ligne 94) : Le rapport généré est envoyé par email grâce à la fonction `send_email()`.
8. Fermeture de la connexion à la base de données (ligne 96) : Une fois le processus

```
(myenv) monitor@jypython:~$ python3 ssh_serveur_mail.py
E-mail envoyé avec succès.
```

dest.userjr@gmail.com
À moi ▾
Tentatives de connexion échouées sur machine web du 2024-10-02 :

Utilisateur Date/Heure Adresse IP Statut

rija 2024-10-02 13:53:09 192.168.76.1 failed

Aucune tentative de connexion échouée sur la machine ftp du 2024-10-02.
Aucune tentative de connexion échouée sur la machine mariadb du 2024-10-02.

jeu. 3 oct. 12:53 (il y a 3 jours)



La Plateforme

4.2 Sauvegarde :

- ssh_cron_backup.py : Sauvegarde des bases de données toutes les 3 heures, avec rétention de 7 sauvegardes

```
#!/usr/bin/python3
# ssh_cron_backup.py > ...
1 import os
2 import time
3 from datetime import datetime
4 import config
5
6 # Dossier où les sauvegardes seront stockées
7 backup_dir = "/home/monitor/backups"
8 os.makedirs(backup_dir, exist_ok=True)
9
10 # Nom de la sauvegarde basé sur l'horodatage actuel
11 timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
12 backup_file = f"{backup_dir}/backup_{timestamp}.sql"
13
14 # Commande de sauvegarde avec mysqldump
15 dump_command = f"mysqldump -h {config.DBipaddr} -u {config.username} -p{config.DBpassword} {config.DBname} > {backup_file}"
16
17 # Exécuter la commande de sauvegarde
18 os.system(dump_command)
19
20 # Garder uniquement les 7 dernières sauvegardes
21 backups = sorted([f for f in os.listdir(backup_dir) if f.startswith("backup_")], reverse=True)
22
23 # Supprimer les sauvegardes les plus anciennes au-delà des 7 dernières
24 for old_backup in backups[7:]:
25     old_backup_path = os.path.join(backup_dir, old_backup)
26     os.remove(old_backup_path)
27     print(f"Ancienne sauvegarde supprimée : {old_backup_path}")
28
29 print(f"Sauvegarde effectuée avec succès : {backup_file}")
30
```

Ce script est conçu pour effectuer des sauvegardes régulières de la base de données, conserver uniquement les sept dernières sauvegardes et supprimer les plus anciennes. Il génère une nouvelle sauvegarde chaque fois qu'il est exécuté, en créant un fichier horodaté pour éviter tout conflit de noms. L'objectif principal est d'automatiser le processus de sauvegarde tout en évitant que le stockage ne soit saturé par des sauvegardes trop anciennes. Ce script est donc utile pour la gestion et la protection des données de manière automatisée.

Lignes 5-6 : Création du dossier où seront stockées les sauvegardes. Si le dossier existe déjà, il n'est pas recréé.

Lignes 9-10 : Le script génère un nom de fichier pour la sauvegarde basée sur l'horodatage actuel (par exemple, backup_2024-10-03_12-00-00.sql).

Ligne 13 : Construction de la commande mysqldump pour effectuer la sauvegarde de la base de données. Cette commande utilise les identifiants de la base de données (hôte, utilisateur, mot de passe, nom de la base de données).

Ligne 16 : Exécution de la commande mysqldump pour créer la sauvegarde dans le fichier précédemment généré.

Ligne 19 : Le script récupère la liste des sauvegardes présentes dans le dossier, les trie dans l'ordre inverse (de la plus récente à la plus ancienne).

La Plateforme

Lignes 22-27 : Le script conserve uniquement les sept dernières sauvegardes. Si plus de sept sauvegardes sont présentes, les plus anciennes sont supprimées pour libérer de l'espace.

Ligne 29 : Affichage d'un message de succès une fois la sauvegarde effectuée.

```
(myenv) monitor@jypython:~$ python3 ssh_cron_backup.py
Sauvegarde effectuée avec succès : /home/monitor/backups/backup_2024-10-06_18-32-54.sql
```

```
GNU nano 7.2
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
0 */3 * * * /home/monitor/myenv/bin/python /home/monitor/ssh_cron_backup.py >> /home/monitor/backup.log 2>&1
*/5 * * * * /home/monitor/myenv/bin/python /home/monitor/ssh_system_mail.py >> /home/monitor/system_mail.log 2>&1
```

```
myenv) monitor@jypython:~/backups$ ls
backup_2024-10-02_15-49-47.sql  backup_2024-10-02_18-00-01.sql  backup_2024-10-03_15-00-02.sql  backup_2024-10-03_18-00-01.sql  backup_2024-10-06_18-00-01.sql  backup_2024-10-06_18-32-54.sql
```



5. Surveillance des ressources systèmes :

5.1 Monitoring :

- ssh_system_status.py : Suivi des ressources (RAM, CPU, disque)

```
import paramiko
import pymysql
from datetime import datetime, timedelta
import config

# Connexion à la base de données MySQL
db = pymysql.connect(
    host=config.DBipaddr,
    user=config.username,
    password=config.DBpassword,
    database=config.DBname
)

# Fonction pour récupérer l'état des ressources système via SSH
def get_system_status(ssh_host):
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(ssh_host, username=config.username, key_filename=config.keyfile)

    # Commandes pour récupérer l'utilisation de la RAM, du CPU et du DISK
    stdin, stdout, stderr = ssh_client.exec_command("free -m | awk 'NR==2{printf \"%s/%sMB (%.2f%%)\", $3,$2,$3*100/$2 }'")
    ram_usage = stdout.read().decode().strip()

    stdin, stdout, stderr = ssh_client.exec_command("top -bn1 | grep 'Cpu(s)' | sed 's/.*, *\\([0-9.]*)\\* id.*\\l/ | awk '{print 100 - $1}'")
    cpu_usage = stdout.read().decode().strip() + "%"

    stdin, stdout, stderr = ssh_client.exec_command("df -h --total | grep 'total' | awk '{print $3\"/\"$2 \" (" $5 \"")\"}'")
    disk_usage = stdout.read().decode().strip()

    ssh_client.close()

    return ram_usage, cpu_usage, disk_usage

# Fonction pour insérer l'état des ressources dans la base de données
def insert_system_status(ssh_host, ram_usage, cpu_usage, disk_usage):
    with db.cursor() as cursor:
        query = """
        INSERT INTO system_status (hostname, ram_usage, cpu_usage, disk_usage, timestamp)
        VALUES (%s, %s, %s, %s, NOW())
        """
        cursor.execute(query, (ssh_host, ram_usage, cpu_usage, disk_usage))
    db.commit()

# Fonction pour supprimer les données plus anciennes que 72 heures
def delete_old_status():
    with db.cursor() as cursor:
        query = "DELETE FROM system_status WHERE timestamp < NOW() - INTERVAL 72 HOUR"
        cursor.execute(query)
    db.commit()

# Liste des serveurs à surveiller
servers = [config.WEBipaddr, config.FTPipaddr, config.DBipaddr]

# Récupérer et stocker l'état des ressources pour chaque serveur
for server in servers:
    ram, cpu, disk = get_system_status(server)
    insert_system_status(server, ram, cpu, disk)
    print(f"Status inséré pour {server} : RAM {ram}, CPU {cpu}, DISK {disk}")

# Supprimer les données plus anciennes que 72 heures
delete_old_status()

# Fermeture de la connexion à la base de données
db.close()
```

Ce script sert donc à surveiller l'état des ressources systèmes (RAM, CPU, disque) de plusieurs serveurs en récupérant et stockant les données dans une base de données MySQL, avec une gestion des anciennes données.

La Plateforme

Connexion à la base de données MySQL (lignes 5-11) :

- Utilise `pymysql` pour se connecter à une base de données MySQL en utilisant les informations d'identification définies dans le fichier de configuration.

Récupération de l'état des ressources système via SSH (lignes 13-25) :

- La fonction `get_system_status()` se connecte via SSH à un serveur distant, exécute des commandes shell pour obtenir l'utilisation de la RAM, du CPU et du disque, et renvoie ces valeurs.

Insertion des données dans la base de données (lignes 27-34) :

- La fonction `insert_system_status()` insère les données sur l'état du système dans la table `system_status` de la base de données.

Suppression des données anciennes (lignes 36-41) :

- La fonction `delete_old_status()` supprime les enregistrements plus anciens que 72 heures dans la table de la base de données pour éviter une accumulation excessive.

Récupération et stockage des données des serveurs (lignes 43-54) :

- Le script parcourt une liste de serveurs définie dans `config`, appelle `get_system_status()` pour chaque serveur, puis insère les données dans la base avec `insert_system_status()`.
- Enfin, il supprime les données plus anciennes que 72 heures à l'aide de `delete_old_status()` et clôture la connexion à la base.

```
(myenv) monitor@jypython:~$ python3 ssh_system_status.py
Status inséré pour 192.168.76.138 : RAM 335/1932MB (17,34%), CPU 97%, DISK 1,8G/21G (9%)
Status inséré pour 192.168.76.139 : RAM 333/1932MB (17,24%), CPU 100%, DISK 1,8G/21G (9%)
Status inséré pour 192.168.76.140 : RAM 534/1932MB (27,64%), CPU 100%, DISK 2,1G/21G (11%)
```



La Plateforme

MariaDB [DB_access]> select * from system_status;						
id	hostname	ram_usage	cpu_usage	disk_usage	timestamp	
315	192.168.76.140	568/1932MB (29.40%)	98%	11%	2024-10-03 18:40:11	
316	192.168.76.138	358/1932MB (18.53%)	96%	9%	2024-10-03 18:45:06	
317	192.168.76.139	350/1932MB (18.12%)	100%	9%	2024-10-03 18:45:09	
318	192.168.76.140	572/1932MB (29.61%)	94%	11%	2024-10-03 18:45:12	
319	192.168.76.138	358/1932MB (18.53%)	95%	9%	2024-10-03 18:50:05	
320	192.168.76.139	350/1932MB (18.12%)	91%	9%	2024-10-03 18:50:08	
321	192.168.76.140	572/1932MB (29.61%)	91%	11%	2024-10-03 18:50:12	
322	192.168.76.138	358/1932MB (18.53%)	92%	9%	2024-10-03 18:55:05	
323	192.168.76.139	350/1932MB (18.12%)	100%	9%	2024-10-03 18:55:07	
324	192.168.76.140	576/1932MB (29.81%)	100%	11%	2024-10-03 18:55:10	
325	192.168.76.138	322/1932MB (16.67%)	95%	9%	2024-10-06 18:10:04	
326	192.168.76.139	308/1932MB (15.94%)	100%	9%	2024-10-06 18:10:07	
327	192.168.76.140	554/1932MB (28.67%)	100%	11%	2024-10-06 18:10:09	
328	192.168.76.138	339/1932MB (17.55%)	100%	9%	2024-10-06 18:15:04	
329	192.168.76.139	328/1932MB (16.98%)	95%	9%	2024-10-06 18:15:07	
330	192.168.76.140	555/1932MB (28.73%)	100%	11%	2024-10-06 18:15:09	
331	192.168.76.138	358/1932MB (18.53%)	100%	9%	2024-10-06 18:20:04	
332	192.168.76.139	329/1932MB (17.03%)	95%	9%	2024-10-06 18:20:07	
333	192.168.76.140	559/1932MB (28.93%)	100%	11%	2024-10-06 18:20:09	
334	192.168.76.138	359/1932MB (18.58%)	98%	9%	2024-10-06 18:25:04	
335	192.168.76.139	331/1932MB (17.13%)	100%	9%	2024-10-06 18:25:06	
336	192.168.76.140	539/1932MB (27.90%)	97%	11%	2024-10-06 18:25:09	
337	192.168.76.138	361/1932MB (18.69%)	97%	9%	2024-10-06 18:30:04	
338	192.168.76.139	331/1932MB (17.13%)	100%	9%	2024-10-06 18:30:07	
339	192.168.76.140	541/1932MB (28.00%)	100%	11%	2024-10-06 18:30:09	
340	192.168.76.138	361/1932MB (18.69%)	100%	9%	2024-10-06 18:35:04	
341	192.168.76.139	333/1932MB (17.24%)	93%	9%	2024-10-06 18:35:07	
342	192.168.76.140	519/1932MB (26.86%)	97%	11%	2024-10-06 18:35:09	
343	192.168.76.138	335/1932MB (17.34%)	97%	1,8G/21G (9%)	2024-10-06 18:39:59	
344	192.168.76.139	333/1932MB (17.24%)	100%	1,8G/21G (9%)	2024-10-06 18:40:02	
345	192.168.76.138	339/1932MB (17.55%)	100%	9%	2024-10-06 18:40:04	
346	192.168.76.140	534/1932MB (27.64%)	100%	2,1G/21G (11%)	2024-10-06 18:40:05	
347	192.168.76.139	333/1932MB (17.24%)	97%	9%	2024-10-06 18:40:06	
348	192.168.76.140	536/1932MB (27.74%)	97%	11%	2024-10-06 18:40:09	



La Plateforme

MariaDB [DB_access]> select * from system_status;						
id hostname ram_usage cpu_usage disk_usage timestamp						
316	192.168.76.138	358/1932MB (18,53%)	96%	9%	2024-10-03 18:45:06	
317	192.168.76.139	350/1932MB (18,12%)	100%	9%	2024-10-03 18:45:09	
318	192.168.76.140	572/1932MB (29,61%)	94%	11%	2024-10-03 18:45:12	
319	192.168.76.138	358/1932MB (18,53%)	95%	9%	2024-10-03 18:50:05	
320	192.168.76.139	350/1932MB (18,12%)	91%	9%	2024-10-03 18:50:08	
321	192.168.76.140	572/1932MB (29,61%)	91%	11%	2024-10-03 18:50:12	
322	192.168.76.138	358/1932MB (18,53%)	92%	9%	2024-10-03 18:55:05	
323	192.168.76.139	350/1932MB (18,12%)	100%	9%	2024-10-03 18:55:07	
324	192.168.76.140	576/1932MB (29,81%)	100%	11%	2024-10-03 18:55:10	
325	192.168.76.138	322/1932MB (16,67%)	95%	9%	2024-10-06 18:10:04	
326	192.168.76.139	308/1932MB (15,94%)	100%	9%	2024-10-06 18:10:07	
327	192.168.76.140	554/1932MB (28,67%)	100%	11%	2024-10-06 18:10:09	
328	192.168.76.138	339/1932MB (17,55%)	100%	9%	2024-10-06 18:15:04	
329	192.168.76.139	328/1932MB (16,98%)	95%	9%	2024-10-06 18:15:07	
330	192.168.76.140	555/1932MB (28,73%)	100%	11%	2024-10-06 18:15:09	
331	192.168.76.138	358/1932MB (18,53%)	100%	9%	2024-10-06 18:20:04	
332	192.168.76.139	329/1932MB (17,03%)	95%	9%	2024-10-06 18:20:07	
333	192.168.76.140	559/1932MB (28,93%)	100%	11%	2024-10-06 18:20:09	
334	192.168.76.138	359/1932MB (18,58%)	98%	9%	2024-10-06 18:25:04	
335	192.168.76.139	331/1932MB (17,13%)	100%	9%	2024-10-06 18:25:06	
336	192.168.76.140	539/1932MB (27,90%)	97%	11%	2024-10-06 18:25:09	
337	192.168.76.138	361/1932MB (18,69%)	97%	9%	2024-10-06 18:30:04	
338	192.168.76.139	331/1932MB (17,13%)	100%	9%	2024-10-06 18:30:07	
339	192.168.76.140	541/1932MB (28,00%)	100%	11%	2024-10-06 18:30:09	
340	192.168.76.138	361/1932MB (18,69%)	100%	9%	2024-10-06 18:35:04	
341	192.168.76.139	333/1932MB (17,24%)	93%	9%	2024-10-06 18:35:07	
342	192.168.76.140	519/1932MB (26,86%)	97%	11%	2024-10-06 18:35:09	
343	192.168.76.138	335/1932MB (17,34%)	97%	1,8G/21G (9%)	2024-10-06 18:39:59	
344	192.168.76.139	333/1932MB (17,24%)	100%	1,8G/21G (9%)	2024-10-06 18:40:02	
345	192.168.76.138	339/1932MB (17,55%)	100%	9%	2024-10-06 18:40:04	
346	192.168.76.140	534/1932MB (27,64%)	100%	2,1G/21G (11%)	2024-10-06 18:40:05	
347	192.168.76.139	333/1932MB (17,24%)	97%	9%	2024-10-06 18:40:06	
348	192.168.76.140	536/1932MB (27,74%)	97%	11%	2024-10-06 18:40:09	
349	192.168.76.138	341/1932MB (17,65%)	97%	1,8G/21G (9%)	2024-10-06 18:43:38	
350	192.168.76.139	333/1932MB (17,24%)	100%	1,8G/21G (9%)	2024-10-06 18:43:41	
351	192.168.76.140	536/1932MB (27,74%)	100%	2,1G/21G (11%)	2024-10-06 18:43:43	



La Plateforme

- `ssh_system_mail.py` : Alerte par e-mail si dépassement des seuils critiques

```
1 import paramiko
2 import pymysql
3 import smtplib
4 from email.mime.text import MIMEText
5 from datetime import datetime, timedelta
6 import config
7 import os
8
9 # Seuils d'alerte
10 CPU_THRESHOLD = 70 # En pourcentage
11 RAM_THRESHOLD = 80 # En pourcentage
12 DISK_THRESHOLD = 90 # En pourcentage
13
14 # Fichier pour suivre l'heure du dernier envoi d'email
15 LAST_EMAIL_FILE = '/home/monitor/last_email_sent.txt'
16
17 # Connexion à la base de données MySQL
18 db = pymysql.connect(
19     host=config.DBipaddr,
20     user=config.username,
21     password=config.DBpassword,
22     database=config.DBname
23 )
24
25 # Fonction pour récupérer l'état des ressources système via SSH
26 def get_system_status(ssh_host):
27     ssh_client = paramiko.SSHClient()
28     ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
29     ssh_client.connect(ssh_host, username=config.username, key_filename=config.keyfile)
30
31     # Récupération des informations RAM, CPU, et DISK
32     stdin, stdout, stderr = ssh_client.exec_command("free -m | awk 'NR==2{printf \"%s/%sMB (%.2f%%)\", $3,$2,$3*100/$2 }'")
33     ram_usage = stdout.read().decode().strip()
34     ram_percent = float(ram_usage.split("(")[1].split("%")[0].replace(",","."))
35     # Remplacer la virgule par un point
36
37     stdin, stdout, stderr = ssh_client.exec_command("top -bn1 | grep 'Cpu(s)' | sed 's/.*/ *\\([0-9.]*)\\%* id.*\\\\1/ | awk '{print 100 - $1}'")
38     cpu_usage = stdout.read().decode().strip() + "%"
39     cpu_percent = float(cpu_usage.strip("%").replace(",","."))
34     # Remplacer la virgule par un point
40
41     stdin, stdout, stderr = ssh_client.exec_command("df -h --total | grep 'total' | awk '{print $5}'")
42     disk_usage = stdout.read().decode().strip()
43     disk_percent = float(disk_usage.strip("%").replace(",","."))
44     # Remplacer la virgule par un point
45
46     ssh_client.close()
47     return ram_usage, ram_percent, cpu_usage, cpu_percent, disk_usage, disk_percent
48
49 # Fonction pour insérer l'état des ressources dans la base de données
50 def insert_system_status(ssh_host, ram_usage, cpu_usage, disk_usage):
51     with db.cursor() as cursor:
52         query = """
53             INSERT INTO system_status (hostname, ram_usage, cpu_usage, disk_usage, timestamp)
54             VALUES (%s, %s, %s, %s, NOW())
55         """
56
57         cursor.execute(query, (ssh_host, ram_usage, cpu_usage, disk_usage))
58     db.commit()
59
60 # Fonction pour envoyer un e-mail d'alerte récapitulatif
61 def send_alert_email(alerts):
62     if not alerts:
63         return # Pas d'alerte, pas d'e-mail
64
65     # Vérification de l'heure du dernier envoi d'email
66     if os.path.exists(LAST_EMAIL_FILE):
67         with open(LAST_EMAIL_FILE, 'r') as f:
68             last_email_time = datetime.strptime(f.read().strip(), "%Y-%m-%d %H:%M:%S")
69             # Si moins d'une heure s'est écoulée depuis le dernier email, ne pas envoyer
70             if datetime.now() - last_email_time < timedelta(hours=1):
71                 print(f"Un email a déjà été envoyé dans l'heure écoulée. Pas d'envoi.")
72                 return
```



La Plateforme

```
70
71     # Construction du contenu de l'e-mail
72     message_content = "Récapitulatif des alertes de système :\n\n"
73     message_content += "\n".join(alerts)
74
75     msg = MIMEText(message_content)
76     msg['Subject'] = "Récapitulatif des alertes système"
77     msg['From'] = config.smtp_user
78     msg['To'] = config.mail_dest
79
80     try:
81         with smtplib.SMTP(config.smtp_server, config.smtp_port) as server:
82             server.starttls()
83             server.login(config.smtp_user, config.smtp_password)
84             server.sendmail(config.smtp_user, config.mail_dest, msg.as_string())
85             print("E-mail récapitulatif envoyé à l'administrateur.")
86
87         # Enregistrer l'heure actuelle dans le fichier après l'envoi de l'email
88         with open(LAST_EMAIL_FILE, 'w') as f:
89             f.write(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
90
91     except Exception as e:
92         print(f"Erreur lors de l'envoi de l'e-mail : {e}")
93
94     # Fonction pour supprimer les données plus anciennes que 72 heures
95     def delete_old_status():
96         with db.cursor() as cursor:
97             query = "DELETE FROM system_status WHERE timestamp < NOW() - INTERVAL 72 HOUR"
98             cursor.execute(query)
99             db.commit()
100
101    # Liste des serveurs à surveiller
102    servers = [config.WEBipaddr, config.FTPipaddr, config.DBipaddr]
103
104    # Récupérer et stocker l'état des ressources pour chaque serveur
105    alerts = []
106    for server in servers:
107        ram_usage, ram_percent, cpu_usage, cpu_percent, disk_usage, disk_percent = get_system_status(server)
108
109        # Insérer l'état des ressources dans la base de données
110        insert_system_status(server, ram_usage, cpu_usage, disk_usage)
111
112        # Vérification des seuils et ajout d'alertes si nécessaire
113        if cpu_percent > CPU_THRESHOLD:
114            alerts.append(f"CPU dépasse le seuil sur {server} : {cpu_percent}% (seuil : {CPU_THRESHOLD}%)")
115        if ram_percent > RAM_THRESHOLD:
116            alerts.append(f"RAM dépasse le seuil sur {server} : {ram_percent}% (seuil : {RAM_THRESHOLD}%)")
117        if disk_percent > DISK_THRESHOLD:
118            alerts.append(f"DISK dépasse le seuil sur {server} : {disk_percent}% (seuil : {DISK_THRESHOLD}%)")
119
120    # Envoyer un email si des alertes ont été détectées
121    send_alert_email(alerts)
122
123    # Supprimer les données plus anciennes que 72 heures
124    delete_old_status()
125
126    # Fermeture de la connexion à la base de données
127    db.close()
```

Ce script surveille les ressources système (RAM, CPU, disque) de plusieurs serveurs via SSH, enregistre les informations dans une base de données et envoie un email récapitulatif en cas de dépassement de seuils prédéfinis. Il vérifie également si un email a déjà été envoyé dans l'heure pour éviter les doublons. Enfin, il supprime les données vieilles de plus de 72 heures pour maintenir une base de données propre.



La Plateforme

1. Seuils d'alerte : (lignes 4-6) Définit des limites d'utilisation pour la RAM, le CPU et le disque.
2. Connexion à la base de données : (ligne 11-14) Connexion à MySQL pour stocker les données de surveillance.
3. Récupération des ressources système :
 - Fonction `get_system_status` (ligne 17-38) : Connecte à un serveur via SSH, exécute des commandes pour récupérer l'utilisation de la RAM, du CPU et du disque, et renvoie ces valeurs.
4. Insertion des données :
 - Fonction `insert_system_status` (ligne 41-48) : Insère l'utilisation des ressources du serveur dans la base de données avec un horodatage.
5. Envoi d'un email récapitulatif :
 - Fonction `send_alert_email` (ligne 50-92) : Envoie un email récapitulatif si les seuils sont dépassés. Elle vérifie également si un email a déjà été envoyé récemment pour éviter les doublons.
6. Suppression des anciennes données :
 - Fonction `delete_old_status` (ligne 94-98) : Supprime les entrées de la base de données datant de plus de 72 heures.
7. Surveillance des serveurs : (ligne 100-125)
 - Récupère l'état des ressources pour chaque serveur, vérifie si elles dépassent les seuils et enregistre les données dans la base de données. Si des seuils sont franchis, une alerte est ajoutée à la liste.
8. Finalisation : (ligne 126) Ferme la connexion à la base de données.

```
GNU nano 7.2
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
0 */3 * * * /home/monitor/myenv/bin/python /home/monitor/ssh_cron_backup.py >> /home/monitor/backup.log 2>&1
*/5 * * * * /home/monitor/myenv/bin/python /home/monitor/ssh_system_mail.py >> /home/monitor/system_mail.log 2>&1
```

Récapitulatif des alertes système Externes Boîte de réception x

dest.userjr@gmail.com

À moi ▾

18:10 (il y a 42 minutes)

Récapitulatif des alertes de système :

CPU dépasse le seuil sur 192.168.76.138 : 95.0% (seuil : 70%)
CPU dépasse le seuil sur 192.168.76.139 : 100.0% (seuil : 70%)
CPU dépasse le seuil sur 192.168.76.140 : 100.0% (seuil : 70%)



5.2 Limitation des alertes :

- Modification pour limiter les alertes à une par heure

```
(myenv) monitor@jupyter:~$ cat last_email_sent.txt  
2024-10-06 18:10:13(myenv) monitor@jupyter:~$ |
```

```
(myenv) monitor@jupyter:~$ python3 ssh_system_mail.py  
Un email a déjà été envoyé dans l'heure écoulée. Pas d'envoi.  
(myenv) monitor@jupyter:~$ |
```

La Plateforme

6. Mises à jour et automatisation :

6.1 Mises à jour :

- ssh_update.py : Vérification et application des mises à jour (Alcasar), avec notification en cas de redémarrage requis

```
❶ ssh_update.py > ⏺ send_recap_email
1  import paramiko
2  import smtplib
3  from email.mime.text import MIMEText
4  import config
5  import time
6
7  # Fichier pour suivre l'envoi des emails
8  LAST_EMAIL_FILE = '/home/monitor/last_update_email_sent.txt'
9  LOG_FILE = '/home/monitor/update_log.txt' # Fichier log pour capturer les erreurs
10
11 # Liste des serveurs à mettre à jour
12 servers = [config.WEBipaddr, config.FTPipaddr, config.DBipaddr]
13
14 # Fonction pour envoyer un email récapitulatif
15 def send_recap_email(summary):
16     msg = MIMEText(summary)
17     msg['Subject'] = "Rapport de mise à jour des serveurs"
18     msg['From'] = config.smtp_user
19     msg['To'] = config.mail_dest
20
21     try:
22         with smtplib.SMTP(config.smtp_server, config.smtp_port) as server:
23             server.starttls()
24             server.login(config.smtp_user, config.smtp_password)
25             server.sendmail(config.smtp_user, config.mail_dest, msg.as_string())
26             print("Email récapitulatif envoyé.")
27     except Exception as e:
28         print(f"Erreur lors de l'envoi de l'email : {e}")
29
30 # Fonction pour écrire dans le fichier de log
31 def write_log(message):
32     with open(LOG_FILE, 'a') as log_file:
33         log_file.write(f"{time.strftime('%Y-%m-%d %H:%M:%S')} - {message}\n")
34
35 # Fonction pour mettre à jour et vérifier les serveurs
36 def update_and_check_reboot(server):
37     ssh_client = paramiko.SSHClient()
38     ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```



La Plateforme

```
40 v try:
41     ssh_client.connect(server, username=config.username, key_filename=config.keyfile)
42
43     print(f"Mise à jour en cours pour le serveur {server}...")
44
45     # Préparer la commande avec sudo -S pour passer le mot de passe en stdin
46     update_command = f"echo {config.sudo_password} | sudo -S apt update && echo {config.sudo_password} | sudo -S apt upgrade -y"
47
48     stdin, stdout, stderr = ssh_client.exec_command(update_command, timeout=600) # Limiter à 10 minutes
49     stdout.channel.recv_exit_status() # Attendre que la commande se termine
50
51     output = stdout.read().decode()
52     error_output = stderr.read().decode()
53     print(output)
54     if error_output:
55         print(error_output)
56         write_log(f"Erreur sur {server} : {error_output}")
57
58     # Vérification si un redémarrage est nécessaire
59     stdin, stdout, stderr = ssh_client.exec_command("if [ -f /var/run/reboot-required ]; then echo 'reboot required'; else echo 'no reboot'; fi", timeout=60)
60     reboot_required = stdout.read().decode().strip()
61
62     ssh_client.close()
63     return reboot_required == "reboot required"
64
65 v except Exception as e:
66     write_log(f"Erreur de connexion ou d'exécution sur {server} : {str(e)}")
67     return False
68
69 # Connexion à chaque serveur pour effectuer les mises à jour
70 summary = ""
71 v for server in servers:
72     reboot_needed = update_and_check_reboot(server)
73     if reboot_needed:
74         summary += f"Le serveur {server} nécessite un redémarrage.\n"
75     else:
76         summary += f"Mise à jour sans redémarrage pour {server}.\n"
77
78 # Envoyer un mail récapitulatif à la fin
79 send_recap_email(summary)
80 print("Mises à jour terminées pour tous les serveurs.")
81
```

Ce script sert principalement à surveiller l'état des ressources système (RAM, CPU, disque) sur plusieurs serveurs, à alerter en cas de dépassement des seuils prédéfinis, et à maintenir des informations à jour dans une base de données. Voici un résumé de son utilité :

1. Surveillance des serveurs : Il récupère l'état des ressources via SSH pour chaque serveur surveillé.
2. Stockage des données : Les données récupérées sont stockées dans une base de données MySQL pour une analyse ultérieure.
3. Alerte par email : Si les ressources dépassent des seuils (pour la RAM, le CPU ou l'espace disque), le script envoie un email récapitulatif à l'administrateur.
4. Maintenance des données : Les enregistrements de la base de données vieux de plus de 72 heures sont supprimés pour éviter l'encombrement.



La Plateforme

Définition des seuils d'alerte (lignes 4-6) : Les seuils pour le CPU, la RAM et le disque sont définis, et si ces limites sont dépassées, une alerte est déclenchée.

Connexion à la base de données (lignes 9-12) : Cette fonction initialise une connexion à la base de données MySQL où les informations des ressources seront stockées.

Récupération des ressources système via SSH (lignes 16-34) : La fonction `get_system_status()` récupère l'utilisation de la RAM, du CPU, et du disque en se connectant aux serveurs via SSH, en utilisant Paramiko pour exécuter les commandes appropriées. Elle retourne les valeurs récupérées pour ces ressources.

Insertion des données dans la base de données (lignes 36-43) : La fonction `insert_system_status()` insère les valeurs des ressources collectées (RAM, CPU, disque) ainsi que leur horodatage dans une table MySQL.

Envoi d'un email récapitulatif (lignes 45-93) : La fonction `send_alert_email()` est responsable de l'envoi d'un email si des alertes sont déclenchées. Elle vérifie également si un email a déjà été envoyé dans l'heure et, si c'est le cas, empêche un nouvel envoi.

Suppression des données plus anciennes (lignes 94-98) : La fonction `delete_old_status()` supprime les enregistrements de la base de données vieux de plus de 72 heures pour maintenir les données à jour.

Vérification des serveurs (lignes 99-123) : Cette boucle itère sur la liste des serveurs définie, récupère l'état des ressources via `get_system_status()`, insère ces informations dans la base de données et vérifie si une alerte doit être déclenchée en comparant les valeurs aux seuils définis.

Fermeture de la connexion (ligne 127) : La connexion à la base de données est fermée une fois toutes les opérations terminées.



La Plateforme

```
(myenv) monitor@jypython:~$ python3 ssh_update.py
Mise à jour en cours pour le serveur 192.168.76.138...
Atteint :1 http://security.debian.org/debian-security bookworm-security InRelease
Atteint :2 http://deb.debian.org/debian bookworm InRelease
Atteint :3 http://deb.debian.org/debian bookworm-updates InRelease
Lecture des listes de paquets...
Construction de l'arbre des dépendances...
Lecture des informations d'état...
Tous les paquets sont à jour.
Lecture des listes de paquets...
Construction de l'arbre des dépendances...
Lecture des informations d'état...
Calcul de la mise à jour...
Le paquet suivant a été installé automatiquement et n'est plus nécessaire :
    linux-image-6.1.0-10-amd64
Veuillez utiliser « sudo apt autoremove » pour le supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.

[sudo] Mot de passe de monitor :
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Mise à jour en cours pour le serveur 192.168.76.139...
Atteint :1 http://security.debian.org/debian-security bookworm-security InRelease
Atteint :2 http://deb.debian.org/debian bookworm InRelease
Atteint :3 http://deb.debian.org/debian bookworm-updates InRelease
Lecture des listes de paquets...
Construction de l'arbre des dépendances...
Lecture des informations d'état...
Tous les paquets sont à jour.
Lecture des listes de paquets...
Construction de l'arbre des dépendances...
Lecture des informations d'état...
Calcul de la mise à jour...
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
    libldap-2.4-2 libssl1.1 linux-image-6.1.0-10-amd64
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.

[sudo] Mot de passe de monitor :
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Mise à jour en cours pour le serveur 192.168.76.140...
Atteint :1 http://security.debian.org/debian-security bookworm-security InRelease
Atteint :2 http://deb.debian.org/debian bookworm InRelease
Atteint :3 http://deb.debian.org/debian bookworm-updates InRelease
Lecture des listes de paquets...
Construction de l'arbre des dépendances...
Lecture des informations d'état...
Tous les paquets sont à jour.
Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 2005 (apt)...
Lecture des listes de paquets...
Construction de l'arbre des dépendances...
Lecture des informations d'état...
Calcul de la mise à jour...
Le paquet suivant a été installé automatiquement et n'est plus nécessaire :
    linux-image-6.1.0-10-amd64
Veuillez utiliser « sudo apt autoremove » pour le supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

Rapport de mise à jour des serveurs Externes Boîte de réception x

dest.userjr@gmail.com

À moi ▾

19:59 (il y a 1 minute)

Mise à jour sans redémarrage pour 192.168.76.138.
Mise à jour sans redémarrage pour 192.168.76.139.
Mise à jour sans redémarrage pour 192.168.76.140.



6.2 Automatisation Google Chat :

- Envoi de rapports périodiques à l'équipe via un espace Google Chat

Création spacegoogle : A FAIRE

Script :

```
from json import dumps
from httplib2 import Http
import paramiko
import config

def get_system_status(server):
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(server, username=config.username, key_filename=config.keyfile)

    stdin, stdout, stderr = ssh_client.exec_command("free -m | awk 'NR==2{printf \"%s/%sMB (%.2F%%)\", $3,$2,$3*100/$2 }'")
    ram_usage = stdout.read().decode().strip()

    stdin, stdout, stderr = ssh_client.exec_command("df -h --total | grep 'total' | awk '{print $3\"/\"$2\"B ($\"$5\")\"}'")
    disk_usage = stdout.read().decode().strip()

    stdin, stdout, stderr = ssh_client.exec_command("top -bn1 | grep 'Cpu(s)' | sed 's/.*/ *\\([0-9.]*)%* id.*\\(1/1/| awk '{print 100 - $1\"%\"}'")
    cpu_usage = stdout.read().decode().strip()

    ssh_client.close()

    return f"RAM: {ram_usage}, Disk: {disk_usage}, CPU: {cpu_usage}"

def send_message_to_google_chat(message):
    url = "https://chat.googleapis.com/v1/spaces/AAAAiKbsh_Y/messages?key=AIzaSy0dI0hCZtE6vySjMn-WEfRq3CPzqKqqshI&token=1IoSnMqR-tYMF7V_--hL2csElAkP6aS600-_MyBH6VU"
    app_message = {"text": message}
    message_headers = {"Content-Type": "application/json; charset=UTF-8"}
    http_obj = Http()
    response = http_obj.request(
        uri=url,
        method="POST",
        headers=message_headers,
        body=dumps(app_message),
    )
    print(response)

if __name__ == "__main__":
    servers = [config.WEBipaddr, config.FTPipaddr, config.DBipaddr]
    message = "État des serveurs :\n"
    for server in servers:
        status = get_system_status(server)
        message += f"Serveur {server} : {status}\n"

    send_message_to_google_chat(message)
```

Ce script sert à récupérer les informations sur l'état des ressources (RAM, CPU et disque) des serveurs distants via SSH, puis à envoyer ces informations sous forme de message dans un espace Google Chat.

La Plateforme

Importation des modules (lignes 1-4) : Ce script utilise les modules `paramiko`, `json`, `httplib2`, et un fichier de configuration pour récupérer les informations système et les envoyer à Google Chat.

Récupération de l'état des ressources système (lignes 6-21) : La fonction `get_system_status(server)` se connecte à un serveur via SSH (Paramiko) et exécute des commandes Linux pour obtenir l'état de la RAM, du disque et du CPU. Les résultats sont renvoyés sous forme de texte.

Envoi du message à Google Chat (lignes 23-39) : La fonction `send_message_to_google_chat(message)` envoie les données système sous forme de message à un espace Google Chat en utilisant une requête HTTP POST.

Exécution principale (lignes 41-50) : Le script récupère l'état des ressources des serveurs spécifiés dans `config` et les envoie à l'espace Google Chat via la fonction `send_message_to_google_chat`.

```
myenv) ➜  testjob15 python googlechat.py
HTTPConnectionPool(host='chat.googleapis.com'): "GET /v1/spaces/AAAHiHoch-1m0:history?&view=summary" 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: 1099
Date: Wed, 06 Oct 2021 17:07:56 GMT
Server: Google Frontend
Set-Cookie: CNPSS-dynamic-integration=CjAQ.OLQsVqTAjwLX.NvUqSlip1tCgN-mpL5tLh6kAQmZq0LgQUwt91p2Cts9vIufuBPKJDTCTJwRF5SGc9eyfH0V_4QWAdk6MME; expires=Wed, 16-Oct-2021 17:07:56 GMT; path=/; Secure; HttpOnly; Alt-Svc: h3=":443"; max-age=2592000; h3-2bin=":443"; max=2592000; transfer-encoding: chunked; status: 200; content-length: 1089; content-encoding: gzip}
b'[\n  "name": "spaces/AAAHiHoch/_messages/B13H.9cVdLj813H.kv40u",\n  "text": "\u00d7xCPat des serveurs :\u00d7\n  \"server\": 192.168.76.138,\n  \"name\": \"spaces/AAAHiHoch-1m0\"\n]'\n
```

