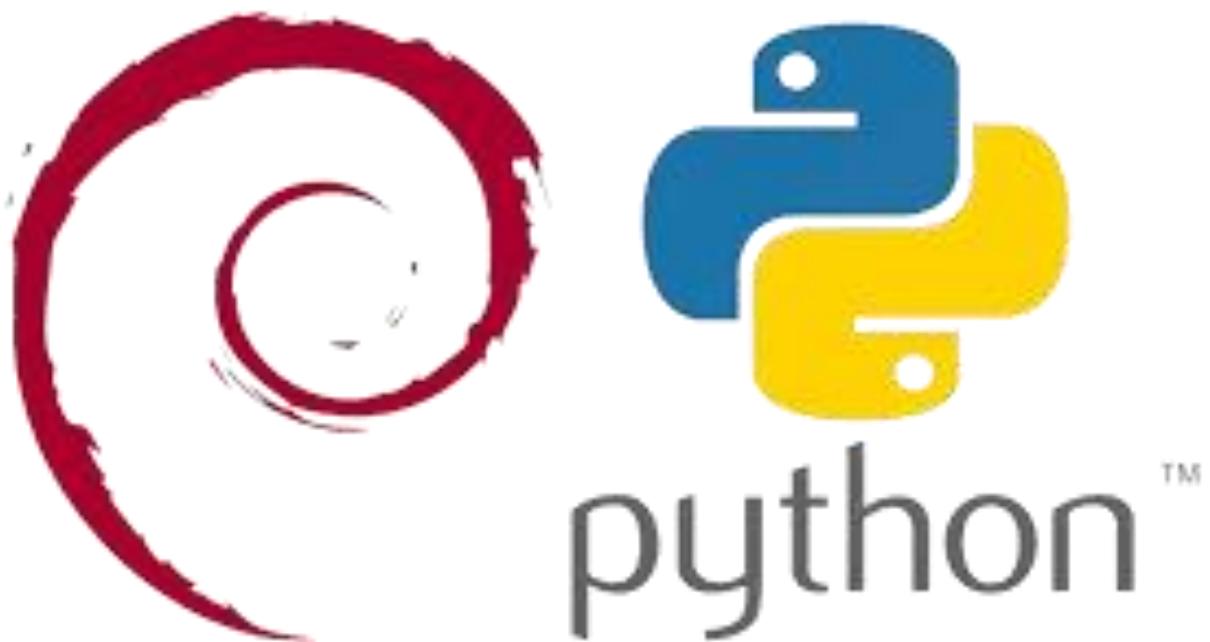


# Centralisation des protocoles de gestion d'erreurs



SOMMAIRE centralisation des protocoles de gestion d'erreurs :

## **1. Création des Environnements Virtuels : JOB01 LUCAS**

- 1.1. Création de 3 VM Debian (FTP, Web, SQL)
- 1.2. Configuration des accès SSH avec le compte "monitor"

## **2. VM dédiée aux Scripts Python : JOB02 LUCAS**

- 2.1. Création d'une VM Debian avec Python, MySQL/MariaDB (client), FTP, et outils d'envoi de mails
- 2.2. Placement de cette VM dans le réseau de la plateforme

## **3. Scripting SSH : JOB03 & JOB04 LUCAS**

- 3.1. `ssh_login.py` : Connexion SSH et exécution de commandes shell
- 3.2. `ssh_login_sudo.py` : Connexion SSH et exécution de commandes en mode sudo

## **4. Scripting MariaDB/MySQL : JOB05 LUCAS & JOB06 MANON**

- 4.1. `ssh_mysql.py` : Connexion au serveur MariaDB/MySQL et vérification des accès
- 4.2. `ssh_mysql_error.py` : Récupération des tentatives d'accès avec mot de passe incorrect et stockage dans la base SQL

## **5. Scripting FTP : JOB07 MANON**

- 5.1. `ssh_ftp_error.py` : Récupération des logs d'erreurs FTP et stockage dans la base SQL

## **6. Scripting Web : JOB08 MANON**

- 6.1. `ssh_web_error.py` : Récupération des logs d'erreurs d'accès Web et stockage dans la base SQL

## **7. Envoi de Mails : JOB09 MANON**

- 7.1. `ssh_serveur_mail.py` : Envoi de mails quotidiens à l'administrateur système avec les tentatives de connexion échouées

## **8. Sauvegarde des Bases de Données : JOB10 MANON**

- 8.1. `ssh_cron_backup.py` : Sauvegarde locale de la base de données avec planification toutes les 3 heures

## **9. Surveillance des Ressources Systèmes : JOB11 & JOB12 & JOB13 RIJA**

- 9.1. `ssh_system_status.py` : Récupération et stockage de l'état des ressources systèmes (RAM/CPU/DISK)
- 9.2. `ssh_system_mail.py` : Envoi d'alertes si dépassement des seuils d'utilisation des ressources
- 9.3. Ne pas envoyer plus d'un mail par heure à l'administrateur

## **10. Gestion des Mises à Jour des Serveurs : JOB14 RIJA**

- 10.1. `ssh_update.py` : Connexion à Alcasar pour vérifier et appliquer les mises à jour des serveurs

## 11. Automatisation via Google Chat : JOB15 RIJA

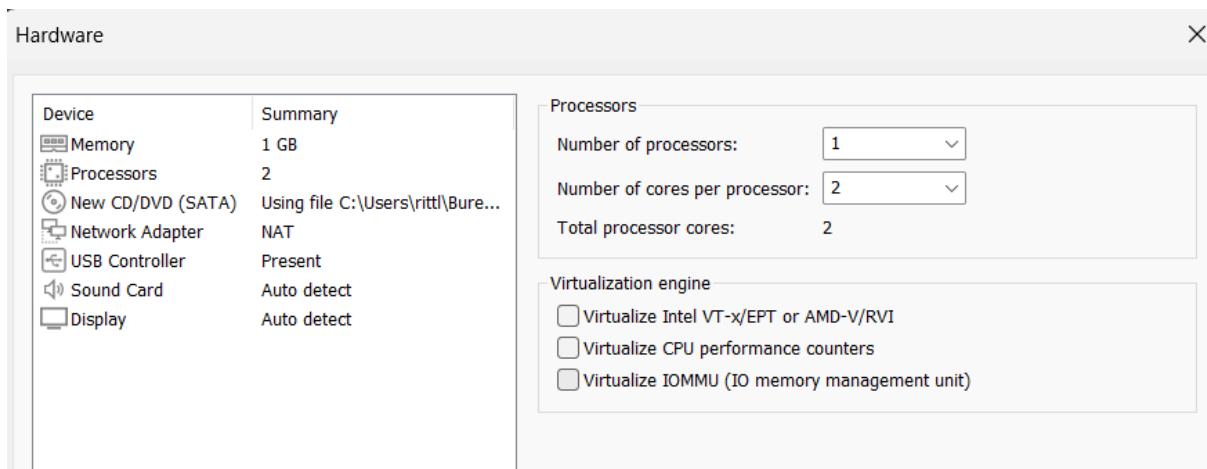
### 11.1. Création d'un espace Google Chat pour l'équipe et envoi de rapports périodiques via un script

## 1 Création des environnements virtuels :

### 1.1. Création des 3 VM Debian pour FTP, Web et SQL

#### VM serveur FTP :

Pour respecter la consigne, nous avons choisi d'appeler la machine : "lrmServeur\_ftp" et le compte utilisateur doit être "monitor" et il doit avoir le droit sudo. Mettre 2 g ram trop lent sinon



Pourquoi configurer la vm avec 1 seul processeur et 2 coeurs :

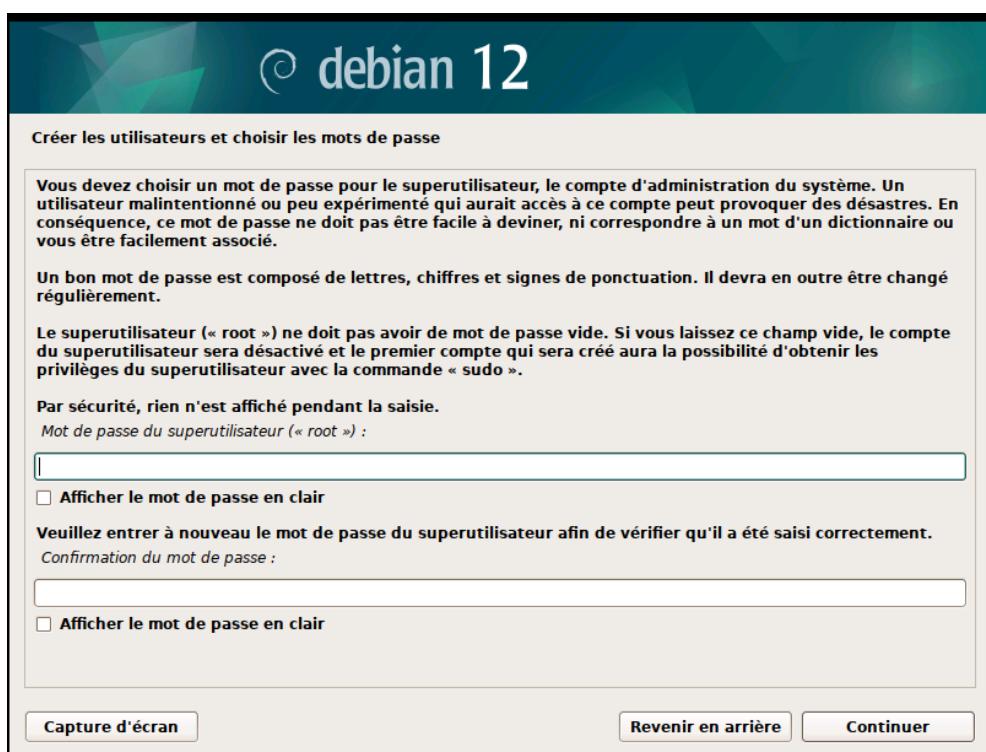
1. **Utilisation optimale des coeurs** : Le fait d'avoir 2 coeurs dans un seul processeur permet à la machine d'exécuter plus efficacement plusieurs tâches simultanément (multi-threading), tout en restant dans les limites des ressources mémoire et disque. Il s'agit d'une configuration idéale pour des charges de travail légères à modérées.
2. **Performance équilibrée** : Avec 2 Go de RAM, la machine virtuelle a une quantité limitée de mémoire. Utiliser un seul processeur avec 2 coeurs permet de mieux gérer les ressources disponibles sans surcharger la machine avec des processus multi-coeurs qui pourraient consommer plus de RAM. Cela assure une bonne répartition des ressources entre le processeur et la mémoire.

3. **Éviter la surcharge de l'hôte** : Si tu exécutes cette machine virtuelle sur un hôte avec d'autres machines ou des applications exigeantes, limiter le processeur à 1 (mais avec 2 cœurs) réduit l'impact de la machine virtuelle sur l'ensemble du système. Cela permet à l'hôte de mieux gérer ses ressources et d'éviter la saturation.

Cette configuration (1 processeur avec 2 cœurs, 1 Go de RAM et 8 Go de stockage) est adaptée pour une machine virtuelle destinée à des tâches légères. Cela permet une bonne gestion des ressources tout en évitant de surcharger l'hôte ou la machine virtuelle elle-même.

! On ne définit pas le mdp de “root” dans la configuration pourquoi ?

Le compte root n'a pas de mot de passe défini par défaut si on le définit pas pendant l'installation du système sur **Debian**. À la place, l'utilisateur créé lors de l'installation reçoit des priviléges sudo.



Vérification interdiction accès ROOT:

```
monitor@lrmServeurFtp:~$ su
Mot de passe :
su: Échec de l'authentification
```

L'utilisateur "monitor" aura le rôle d'administrateur , c'est lui qui peut gérer et surveiller les trois VM que l'on a déployé : le serveur FTP, le serveur Web et le serveur SQL. )

### Pourquoi un compte spécifique ?

En créant un compte spécifique "monitor", nous renforçons la sécurité de nos serveurs. Tout d'abord, en limitant l'accès root, nous réduisons les risques d'une compromission du système, puisque seul le compte "monitor" dispose des priviléges nécessaires pour effectuer les tâches de maintenance.

### Choix de notre protocole pour notre serveur FTP

Critère	SFTP	vsftpd
Type	Protocole (au-dessus de SSH)	Serveur FTP (implémente FTP et FTPS)
Sécurité	Sécurisé par défaut (chiffrement via SSH)	Sécurisé si configuré pour FTPS (SSL/TLS)
Port	Utilise le port 22 (SSH)	Utilise les ports 21 (FTP) et 20
Fonction	Transfert de fichiers sécurisé	Serveur FTP sécurisé
Protocole	SFTP (protocole distinct)	FTP/FTPS (protocole standard)
Cas d'usage	Sécurité maximale, transferts entre serveurs	Compatibilité FTP, transferts à grande échelle

Vu que nous devons utiliser le chiffrement via ssh ce qui est pertinent car meilleur sécurité que SSL/TLS, nous allons donc installer et configurer SFTP.

Pour se faire, on vérifie que openssh soit bien installé ou on installe les paquets.

⚠️ On pense bien à mettre son adresse ip en statique et on vérifie bien dans notre fichier hosts que notre serveur soit déclaré.

```
GNU nano 7.2                                         /etc/network/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 192.168.217.131
    netmask 255.255.255.0
    gateway 192.168.217.2
```

```
GNU nano 7.2                                         /etc/hosts
127.0.0.1      localhost
127.0.1.1      lrmServeurFtp
192.168.217.131 lrmServeurFtp

# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

⚠️ on pense à redémarrer le service network.

ok maintenant on crée notre groupe pour nos utilisateurs sftp puis on crée le répertoire.

⚠️ on le met à la racine "/" pour une question de facilité après on peut le mettre où on le souhaite mais je trouve plus simple de le placer ici.

Création du groupe sftp pour le service sftp :

```
sudo groupadd sftp
```

```
sudo adduser sftp_user (user, entrer ...)
```

```
monitor@lrmServeurFtp:~$ sudo groupadd sftp_user
monitor@lrmServeurFtp:~$ mkdir /sftp
mkdir: impossible de créer le répertoire « /sftp »: Permission non accordée
monitor@lrmServeurFtp:~$ sudo 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
```

Cette configuration dans le fichier SSH configure un environnement SFTP sécurisé pour les utilisateurs du groupe sftp. Voici les points essentiels :

- **Subsystem sftp internal-sftp** : Utilise le sous-système SFTP interne pour gérer les transferts de fichiers, sans exécuter de processus externe, ce qui renforce la sécurité.
- **Match Group sftp** : Applique des règles spécifiques aux utilisateurs du groupe sftp.
- **X11Forwarding no** : Désactive le transfert graphique X11 pour ces utilisateurs, car ils n'en ont pas besoin pour SFTP.
- **AllowTcpForwarding no** : Désactive le tunneling TCP, limitant les risques de redirection de connexions non autorisées.
- **ForceCommand internal-sftp** : Implique que les utilisateurs du groupe sftp peuvent uniquement utiliser SFTP et non un accès shell. Cela empêche toute tentative d'exécuter des commandes SSH.

```
monitor@lrmServeurFtp:~$ sftp monitor@192.168.217.131
The authenticity of host '192.168.217.131 (192.168.217.131)' can't be established.
ED25519 key fingerprint is SHA256:U+xdUojt+qqwB08P3vL08Yh8Kqm/Rp7bgdbuFepIKAU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.217.131' (ED25519) to the list of known hosts.
monitor@192.168.217.131's password:
Connected to 192.168.217.131.
sftp> exit
```

## Vm serveur WEB avec une authentification basic pour monitor :

Comme tout bon serveur, on le passe en static

```
GNU nano 7.2                                     /etc/network/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 192.168.217.132/24
    gateway 192.168.217.2
```

On vérifie que les paquets correspondant à apache2 soit là soit bien installé

```
monitor@lrmServeurWeb:~$ sudo systemctl restart networking
monitor@lrmServeurWeb:~$ sudo apt install apache2
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
apache2 est déjà la version la plus récente (2.4.62-1~deb12u1).
apache2 passé en « installé manuellement ».
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
monitor@lrmServeurWeb:~$ sudo apt install apache2
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
apache2 est déjà la version la plus récente (2.4.62-1~deb12u1).
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
monitor@lrmServeurWeb:~$
```

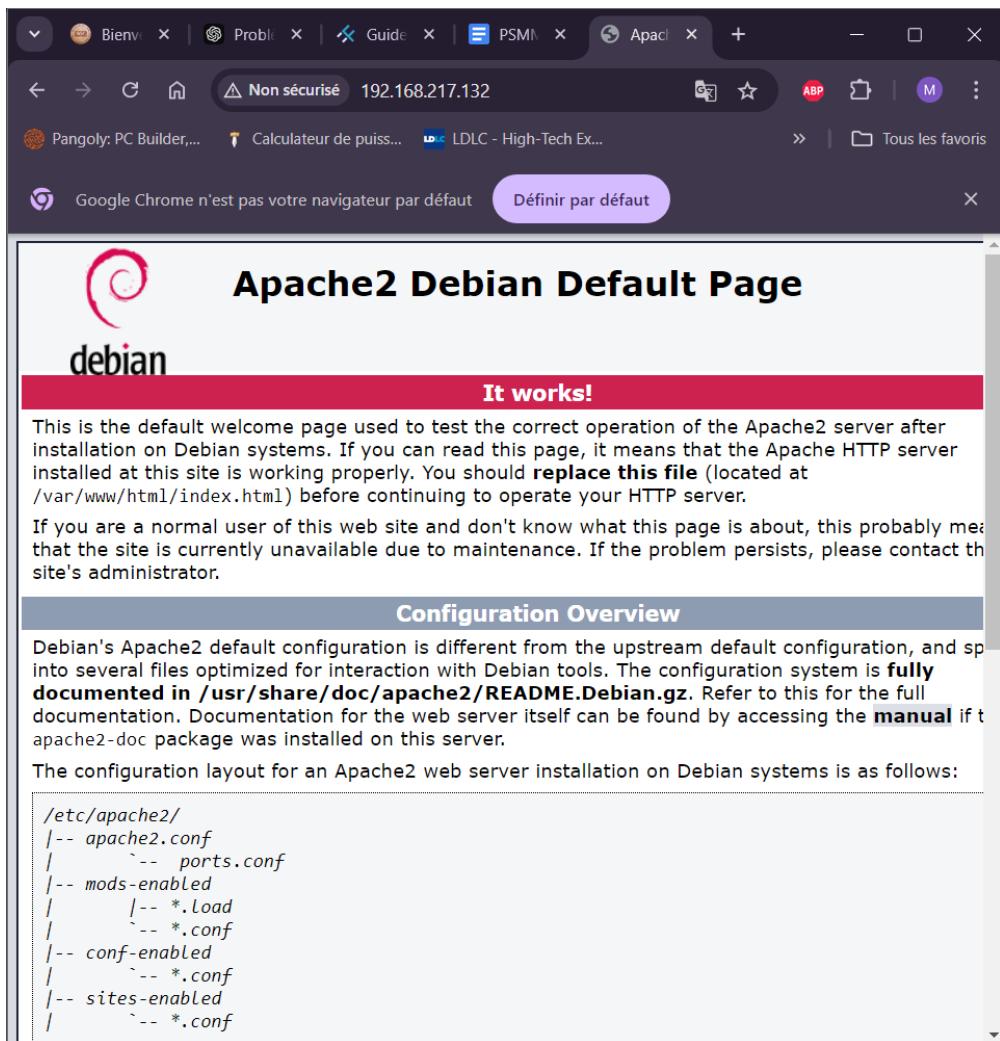
```
monitor@lrmServeurWeb:~$ sudo apt-get install apache2-utils
[sudo] Mot de passe de monitor :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
apache2-utils est déjà la version la plus récente (2.4.62-1~deb12u1).
0 mis à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

On vérifie que le service fonctionne pour passer à la configuration

```
monitor@lrmServeurWeb:~$ sudo systemctl start apache2
monitor@lrmServeurWeb:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enabled)
  Active: active (running) since Mon 2024-09-23 09:55:25 CEST; 42min ago
    Docs: https://httpd.apache.org/docs/2.4/
   Process: 717 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
 Main PID: 792 (apache2)
    Tasks: 55 (limit: 2264)
   Memory: 14.0M
      CPU: 224ms
     CGroup: /system.slice/apache2.service
             └─792 /usr/sbin/apache2 -k start
                 ├─795 /usr/sbin/apache2 -k start
                 ├─796 /usr/sbin/apache2 -k start

sept. 23 09:55:25 lrmServeurWeb systemd[1]: Starting apache2.service - The Apache HTTP Server.
sept. 23 09:55:25 lrmServeurWeb apachectl[769]: AH00557: apache2: apr_sockaddr_info_get() fail
sept. 23 09:55:25 lrmServeurWeb apachectl[769]: AH00558: apache2: Could not reliably determine
sept. 23 09:55:25 lrmServeurWeb systemd[1]: Started apache2.service - The Apache HTTP Server. sudo
```

On se rend sur le navigateur de notre hôte puisque nous sommes en nat et on test que apache s'ouvre correctement.



## AUTHENTIFICATION BASIQUE :

Maintenant on va créer *l'authentification basique* pour notre utilisateur “**MONITOR**” pour qu'il puisse se connecter à notre site web de manière sécurisé.

Tout d'abord on installe si ce n'est pas déjà fait apache2-utils qui permet d'avoir un ensemble d'outils qui facilite la gestion et l'administration d'Apache.

Cette commande, nous permet d'ajouter un utilisateur spécifique “monitor” pour nous ainsi que son mdp à la base de données d'authentification.

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

Si l'on consulte le contenu du dossier, on peut voir le nom d'utilisateur et le mot de passe crypté de chaque enregistrement :

```
cat /etc/apache2/.htpasswd
```

```
monitor@lrmWEB:~$ cat /etc/apache2/.htpasswd
monitor:$apr1$CUD3vFqt$PHm0.j7PZ6lRF/mnrCMQ2/
```

Après on se rend dans notre fichier de configuration par défaut d'apache pour **définir les règles d'authentification** pour le répertoire spécifié `/var/www/html`

```
GNU nano 7.2                                         /etc/apache2/sites-available/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 lrmserveurweb@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>

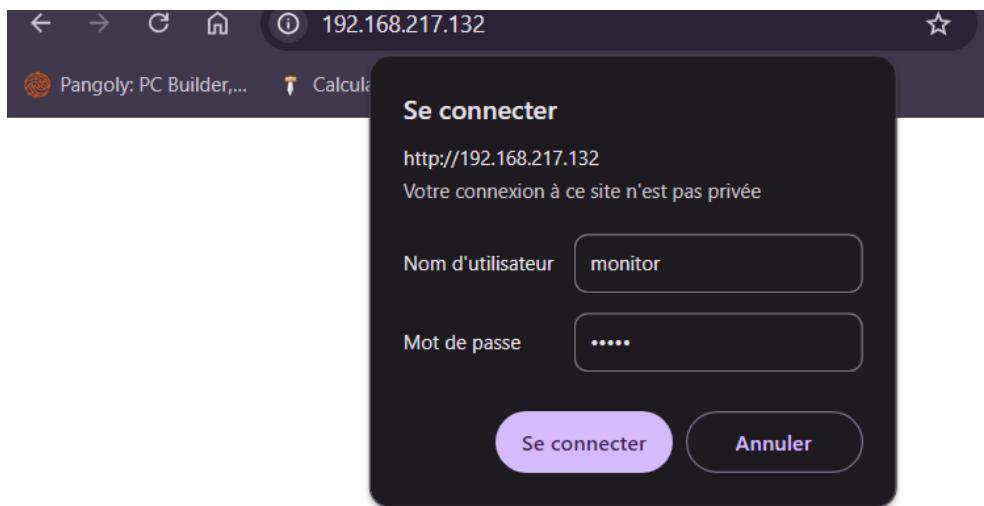
    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>
```

On vérifie la configuration avec la commande suivante:

```
monitor@lrmServeurWeb:~$ 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 sup
press this message
Syntax OK
```

⚠ On pense à redémarrer le service apache2 une fois la configuration vérifié

Puis on retourne sur notre navigateur la page doit s'ouvrir de cette manière



#### CONFIGURATION DU DNS (optionnel) :

Avant de commencer la configuration du DNS, nous allons paramétrer le “Virtual Network Editor” dans VMware de tel sorte que les autres VM interrogeant le serveur DNS en premier lors d'une requête DNS.





**NAT Settings**

Network:	vmnet8
Subnet IP:	192.168.72.0
Subnet mask:	255.255.255.0
Gateway IP:	192.168.72.2

**Virtual Network Editor**

Name	Type	External Connection	Host Connection	DHCP	Subnet Address
VMnet1	Host-only	-	Connected	-	192.168.93.0
<b>VMnet8</b>	<b>NAT</b>	<b>NAT</b>	<b>Connected</b>	<b>Enabled</b>	<b>192.168.72.0</b>

**VMnet Information**

- Bridged (connect VMs directly to the external network)
 

Bridged to:

Automatic Settings...
- NAT (shared host's IP address with VMs)
 

NAT Settings...
- Host-only (connect VMs internally in a private network)

Connect a host virtual adapter to this network  
 Host virtual adapter name: VMware Network Adapter VMnet8

Use local DHCP service to distribute IP address to VMs  
 DHCP Settings...

Subnet IP: 192.168.72.0 Subnet mask: 255.255.255.0

Administrator privileges are required to modify the network configuration. Change Settings

Buttons: Restore Defaults, Import..., Export..., OK, Cancel, Apply, Help

**Domain Name Server (DNS)**

Auto detect available DNS servers

**Virtual Network Editor**

Name	Type	External Connection	Host Connection	DHCP	Subnet Address
VMnet0	Bridged	Auto-bridging	-	-	-
VMnet1	Host-only	-	Connected	-	192.168.93.0
VMnet8	NAT	NAT	Connected	Enabled	192.168.72.0

**VMnet Information**

- Bridged (connect VMs directly to the external network)
 

Bridged to:

Automatic Settings...
- NAT (shared host's IP address with VMs)
 

NAT Settings...
- Host-only (connect VMs internally in a private network)

Connect a host virtual adapter to this network  
 Host virtual adapter name: VMware Network Adapter VMnet8

Use local DHCP service to distribute IP address to VMs  
 DHCP Settings...

Subnet IP: 192.168.72.0 Subnet mask: 255.255.255.0

Buttons: Restore Defaults, Import..., Export..., OK, Cancel, Apply, Help

On peut maintenant s'attaquer à l'installation BIND9. Taper la commande suivante :

```
monitor@lrmServeurWeb:~$ sudo apt install bind9
```

Les fichiers de configurations de BIND9 se situent dans le répertoire “/etc/bind”. Dans ce répertoire, nous allons créer un dossier nommé “zones” dans lequel sera stocké nos fichiers de zone direct et inverse.

```
monitor@lrmServeurWeb:/etc/bind$ ls
bind.keys    db.empty          named.conf.local   zones.rfc1918
db.0         db.local          named.conf.options
db.127       named.conf        rndc.key
db.255       named.conf.default-zones zones
```

### Etape 1: Configuration de BIND

Ouvrir le fichier de configuration “**named.conf.options**” et ajouter les lignes suivantes.

```
acl "trusted" {
    192.168.72.0/24;
    localhost;
};

options {
    directory "/var/cache/bind";

    recursion yes;
    allow-query { trusted; };

    forwarders {
        8.8.8.8;
        8.8.4.4;
    };

    dnssec-validation auto;

    listen-on { any; };
    listen-on-v6 { any; };
};
```

#### Bloc ACL (Access Control List) :

- **acl "trusted"** : Ceci définit une liste d'accès appelée "trusted" (de confiance), contenant des adresses IP qui sont autorisées à interagir avec votre serveur DNS.
- **192.168.72.0/24** : Cette ligne autorise toutes les machines situées dans le sous-réseau 192.168.72.0/24 à interroger votre serveur DNS (ceci inclut toutes les adresses IP allant de 192.168.72.1 à 192.168.72.254).
- **localhost** : Cette ligne autorise le serveur DNS lui-même (localhost, soit 127.0.0.1) à interroger Bind9.

#### Bloc options :

- **directory "/var/cache/bind"** : Définit le répertoire dans lequel Bind stocke ses fichiers de cache. Ce répertoire contient les fichiers temporaires que Bind utilise pour stocker des informations sur les requêtes résolues.
- **recursion yes** : Active la récursivité sur le serveur DNS. Cela signifie que si le serveur ne connaît pas la réponse à une requête, il ira interroger d'autres serveurs DNS pour trouver la réponse.
- **allow-query { trusted; }** : Seules les adresses IP définies dans l'ACL "trusted" (192.168.72.0/24 et localhost) sont autorisées à envoyer des requêtes DNS à ce serveur.
- **forwarders { 8.8.8.8; 8.8.4.4; }** : Ce bloc indique que si le serveur DNS ne parvient pas à résoudre une requête, il la transmet aux serveurs DNS spécifiés ici (8.8.8.8 et 8.8.4.4, qui sont

des serveurs DNS publics de Google). Cela permet au serveur de déléguer la résolution des domaines qu'il ne connaît pas à un autre serveur.

- **dnssec-validation auto** : Active la validation DNSSEC. DNSSEC (Domain Name System Security Extensions) ajoute une couche de sécurité à la résolution DNS en s'assurant que les réponses proviennent bien de la source légitime et n'ont pas été altérées. L'option auto permet à Bind9 d'activer la validation pour les zones qui la supportent.
- **listen-on { any; };** : Cela signifie que le serveur DNS Bind écoutera les requêtes sur toutes les interfaces réseau IPv4 disponibles. Si vous avez plusieurs cartes réseau, Bind acceptera les requêtes venant de n'importe laquelle d'entre elles.
- **listen-on-v6 { any; };** : Similaire à la directive précédente, mais pour les adresses IPv6. Bind écoutera toutes les interfaces compatibles IPv6.

## Etape 2: Définir les zones

Maintenant, modifiez le fichier de forward “**named.conf.local**” et définissez les zones comme suit :

```
zone "lrmserveur.lan" {
    type master;
    file "/etc/bind/zones/db.lrmserveur.lan";
};

zone "72.168.192.in-addr.arpa" {
    type master;
    file "/etc/bind/zones/db.reverse.lrmserveur.lan";
};
```

- **zone "lrmserveur.lan"** : Cette ligne définit une **zone DNS directe** pour le domaine **lrmserveur.lan**. Une zone est une portion de l'espace de noms DNS qui est gérée par votre serveur. Dans ce cas, il s'agit de la zone principale où les noms de domaine seront traduits en adresses IP.
- **type master** : Cela indique que ce serveur est le **serveur maître** pour cette zone. Cela signifie que le serveur DNS Bind est responsable de la gestion de cette zone et qu'il possède les fichiers de zone originaux (le fichier où les enregistrements DNS pour le domaine sont stockés). Si d'autres serveurs DNS existent (serveurs esclaves), ils se synchronisent à partir de ce serveur maître.
- **file "/etc/bind/zones/db.lrmserveur.lan"** : Cette ligne spécifie le **fichier de zone** où les enregistrements DNS pour **lrmserveur.lan** sont stockés. Ce fichier contient les enregistrements tels que les A records (qui font correspondre les noms de domaine aux adresses IP), les CNAME, les MX (mail exchange), etc.

Ensuite, dans le **répertoire “zones”**, créez les fichiers de zones “**db.lrmserveur.lan**” et “**db.inverse.lrmserveur.lan**”.

*Tips : des fichiers de zones prédéfinies sont déjà présents dans le répertoire bind ; “db.local” pour le fichier de zone direct et “db.127” pour le reverse. Vous pouvez les copier et les coller dans le répertoire “zones” en veillant à mettre leurs nouveaux noms.*

Paramétrage du fichier de zone “db.lrmserveur.lan”:

```
; BIND data file for lrmserveur.com
;
$TTL 604800
@ IN SOA lrmServeurWeb.lrmserveur.lan. admin.lrmserveur.lan. (
                    3           ; Serial
                    604800    ; Refresh
                    86400     ; Retry
                    2419200   ; Expire
                    604800 ); Negative Cache TTL
;
@       IN      NS  lrmServeurWeb.lrmserveur.lan.

lrmServeurWeb    IN      A      192.168.72.139
@      IN      A      192.168.72.139
ns1      IN      A      192.168.72.139
www      IN      A      192.168.72.139
lrmserveurweb.lan      IN      CNAME      www.lrmserveur.lan.
```

J'ai rajouté dans ce fichier un enregistrement CNAME :

“**lrmserveurweb.lan IN CNAME [www.lrmserveur.lan](#).**”

Cet enregistrement indique un alias, ici, **lrmserveurweb.lan** est un alias pour [www.lrmserveur.lan](#).  
Cela signifie que toute requête pour lrmserveurweb.lan sera redirigée vers [www.lrmserveur.lan](#).

Paramétrage du fichier “db.reverse.lrmserveur.lan”:

```
; BIND reverse data file for local loopback interface

$TTL    604800
@       IN      SOA     lrmServeurWeb.lrmserveur.lan. admin.lrmserveur
.lan. (
          3           ; Serial
          604800        ; Refresh
          86400         ; Retry
          2419200        ; Expire
          604800 )       ; Negative Cache TTL
;
@       IN      NS      lrmServeurWeb.lrmserveur.lan.
139     IN      PTR      www.lrmserveur.lan.
```

### Etape 3: Configurer la résolution de noms d'hôtes

Le fichier **/etc/hosts** est un fichier de configuration utilisé pour la résolution de noms d'hôtes (hostname) en adresses IP sur les systèmes Unix/Linux. Il permet d'associer manuellement des adresses IP à des noms de domaine ou à des noms d'hôtes locaux.

*⚠ La configuration du fichier **/etc/hosts** n'est pas obligatoire dans toutes les situations. Je l'ai fait pour limiter au maximum les problèmes sur la résolution des noms de domaine.*

```
127.0.0.1      localhost
127.0.1.1      lrmServeurWeb
192.168.72.139 lrmServeurWeb.lrmserveur.lan      lrmServeurWeb
# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

#### 127.0.1.1 lrmServeurWeb

- **127.0.1.1** est une adresse de **loopback locale** différente utilisée dans certaines configurations (notamment sur les systèmes Ubuntu et Debian) pour des raisons de compatibilité réseau.
- **lrmServeurWeb** est ici associé à cette adresse de loopback locale. Cela signifie que lorsque la machine essaie de se connecter à **lrmServeurWeb**, elle se connecte localement à l'adresse **127.0.1.1**, mais ce n'est pas lié à une adresse réseau externe.

#### 192.168.72.139 lrmServeurWeb.lrmserveur.lan lrmServeurWeb

- **192.168.72.139** est l'adresse IP du serveur sur le réseau local.

- **lrmServeurWeb.lrmserveur.lan** est le nom de domaine complet (FQDN, Fully Qualified Domain Name) associé à cette adresse IP.
- **lrmServeurWeb** est le nom d'hôte court (hostname) de la machine. Cela signifie que lorsque la machine essaie de se connecter à **lrmServeurWeb** ou **lrmServeurWeb.lrmserveur.lan**, elle se connecte à l'adresse IP **192.168.72.139**.

#### Etape 4: Tests

Depuis un serveur client situé dans le même sous-réseau.

Tapez la commande “**nslookup www.lrmserveur.lan**”, si vous avez ça d'afficher alors le DNS est bien configuré.

```
root@Client:~# nslookup www.lrmserveur.lan
Server:          192.168.72.2
Address:        192.168.72.2#53
```

```
Non-authoritative answer:
Name:    www.lrmserveur.lan
Address: 192.168.72.139
```

Nous pouvons aussi taper “<http://www.lrmserveur.lan>” depuis la barre de recherche de notre client :

The screenshot shows a web browser window with the URL [www.lrmserveur.lan](http://www.lrmserveur.lan) in the address bar. The page content is the Apache2 Debian Default Page. It includes the Debian logo, the title "Apache2 Debian Default Page", and a red banner with the text "It works!". Below the banner, there is explanatory text about the default welcome page and instructions to replace the file. A "Configuration Overview" section is also present.

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.htm`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

**Configuration Overview**

Debian's Apache2 default configuration is different from the upstream default configuration, and is split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   '-- ports.conf
|-- mods-enabled
|   '-- *.load
|   '-- *.conf
|-- conf-enabled
|   '-- *.conf
|-- sites-enabled
|   '-- *.conf
```

## INSTALLATION & CONFIG Vm Serveur SQL :

Pour ce serveur, nous avons choisi de prendre MariaDB, pour les raisons suivantes:

Voici un tableau comparant les 2 SGBD utilisés le plus souvent mariadb et mySQL:

Critère	MariaDB	MySQL
<b>Performance</b>	Optimisé pour de meilleures performances (moteurs de stockage avancés, meilleure gestion de la concurrence)	Performance stable mais moins de moteurs avancés
<b>Innovation</b>	Développement rapide, nombreuses nouvelles fonctionnalités (colonnes invisibles, vues temporelles, JSON étendu)	Moins d'innovations, développement plus lent
<b>Stabilité</b>	Très bon mais dépend des nouvelles fonctionnalités communautaires	Très stable, prouvé dans les environnements critiques
<b>Support commercial</b>	Principalement communautaire (open source complet)	Support officiel par Oracle (solutions commerciales, garanties)
<b>Moteurs de stockage</b>	Supporte plus de moteurs (Aria, XtraDB, TokuDB, etc.)	Principalement InnoDB
<b>Clustering / Haute disponibilité</b>	RéPLICATION synchrone native avec Galera Cluster	MySQL Group Replication (réPLICATION semi-synchrone)
<b>Utilisation en entreprise</b>	Idéal pour les projets nécessitant performance et innovation open source	Idéal pour les entreprises nécessitant stabilité et support commercial

## Configuration de la vm SQL sous MariaDB

New Virtual Machine Wizard X

**Ready to Create Virtual Machine**

Click Finish to create the virtual machine and start installing Debian 12.x 64-bit.

The virtual machine will be created with the following settings:

Name:	ServerSQL
Location:	C:\Users\savio\Documents\Virtual Machines\ServerSQL
Version:	Workstation 17.5.x
Operating System:	Debian 12.x 64-bit
Hard Disk:	8 GB, Split
Memory:	2048 MB
Network Adapter:	NAT
Other Devices:	2 CPU cores, CD/DVD, USB Controller, Sound Card

[Customize Hardware...](#)

Power on this virtual machine after creation

[< Back](#) [Finish](#) [Cancel](#)

Configuration des fichiers réseau :

```
sudo nano /etc/network/interfaces
```

```
sudo nano /etc/hosts
```

```
GNU nano 7.2
# /etc/network/interfaces_
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 192.168.52.128/24
    gateway 192.168.52.2
    netmask 255.255.255.0
sudo systemctl restart networking
```

```
GNU nano 7.2
# /etc/hosts
127.0.0.1      localhost
127.0.1.1      lmrSQL
192.168.52.128 lmrSQL

# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
```

Installation du service MariaDB :

```
sudo apt update && sudo apt install mariadb-server -y
```

```
sudo systemctl status mariadb
```

```
monitor@lmrSQL:~$ 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 Mon 2024-09-23 15:10:24 CEST; 18s ago
     Docs: man:mariadb(8)
           https://mariadb.com/kb/en/library/systemd/
 Main PID: 4421 (mariadb)
   Status: "Taking your SQL requests now..."
    Tasks: 11 (limit: 2264)
   Memory: 108.3M
      CPU: 1.155s
     CGroup: /system.slice/mariadb.service
             └─4421 /usr/sbin/mariadb
```

Une fois mariadb on va le configurer avec `my_secure_installation` permet de renforcer la sécurité du serveur contre les attaques courantes.

**Définir un mot de passe root sécurisé** : Si ce n'est pas déjà fait, vous pouvez définir ou renforcer le mot de passe de l'utilisateur `root`. Un mot de passe fort est essentiel pour protéger l'accès administrateur à la base de données.

**Supprimer les utilisateurs anonymes :** Par défaut, MySQL/MariaDB permet les connexions anonymes, ce qui peut présenter un risque de sécurité. Il est recommandé de supprimer ces comptes pour éviter que des utilisateurs non authentifiés accèdent à la base de données.

**Désactiver la connexion root à distance :** Cette option permet de restreindre l'accès à l'utilisateur `root` uniquement à partir du serveur local, ce qui réduit le risque d'attaques externes sur cet utilisateur privilégié.

**Supprimer la base de données de test :** Par défaut, MySQL/MariaDB installe une base de données de test accessible à tout utilisateur. Il est recommandé de la supprimer pour éviter qu'elle soit exploitée par des attaquants potentiels.

**Recharger les tables de privilèges :** Cette étape s'assure que tous les changements de sécurité prennent effet immédiatement en rechargeant les tables de privilèges.

```
sudo mysql_secure_installation
```

```
monitor@lmrSQL:~$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] n
... skipping.

You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] n
... skipping.

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them. This is intended only for testing, and to make the installation
go a bit smoother. You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
... Success!

Normally, root should only be allowed to connect from 'localhost'. This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] y
... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access. This is also intended only for testing, and should be removed
before moving into a production environment.
```

```
Remove test database and access to it? [Y/n] y
- Dropping test database...
... Success!
- Removing privileges on test database...
... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
... Success!

Cleaning up...

All done! If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

Lors de l'exécution de cette commande, il faudra répondre à plusieurs questions pour sécuriser l'installation.

Lorsque l'on exécute la commande mysql\_secure\_installation, on a plusieurs questions pour renforcer la sécurité de l'installation de MariaDB.

Voici les réponses recommandées :

**Set root password? [Y/n]**

Réponse recommandée : **Y**

**Explication :** Il est important de configurer un mot de passe pour l'utilisateur root, car par défaut il peut ne pas en avoir, ce qui est un risque de sécurité.

**New password:**

Tape un mot de passe sécurisé (et souviens-toi de ce mot de passe).

**Explication :** Ce mot de passe sera celui utilisé pour te connecter en tant qu'utilisateur root à MySQL/MariaDB. Choisis un mot de passe fort.

Re-enter new password:

Retape le même mot de passe que précédemment.

Explication : Cela confirme que tu n'as pas fait d'erreur lors de la saisie du mot de passe.

**Remove anonymous users? [Y/n]**

Réponse recommandée : **Y**

**Explication :** Supprimer les utilisateurs anonymes est une bonne pratique de sécurité. Cela empêche quelqu'un de se connecter sans fournir de nom d'utilisateur.

**Disallow root login remotely? [Y/n]**

Réponse recommandée : **Y**

Explication : Interdire les connexions à distance pour l'utilisateur root est important pour la sécurité. Cela empêche quelqu'un de l'extérieur de se connecter en tant que root depuis une machine distante.

**Remove test database and access to it? [Y/n]**

Réponse recommandée : **Y**

**Explication :** La base de données de test est utilisée pour des fins de démonstration et peut être un point d'entrée pour des attaques si elle n'est pas sécurisée. Il est conseillé de la supprimer si tu ne l'utilises pas.

**Reload privilege tables now? [Y/n]**

Réponse recommandée : **Y**

**Explication :** Cette commande recharge les tables de privilèges pour que les modifications que l'on vient d'appliquer (comme la suppression des utilisateurs anonymes et des bases de données de test) prennent effet immédiatement.

**Accès à la base de données MariaDB :**

**sudo mysql -u root -p**

```
monitor@lmrSQL:~$ sudo mysql -u root -p
[sudo] Mot de passe de monitor :
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 37
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)]>
```

Création de l'utilisateur monitor sur la base de données :

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'monitor'@'localhost' IDENTIFIED BY 'monitorsql';
Query OK, 0 rows affected (0,002 sec)
```

```
MariaDB [(none)]> SELECT user, host FROM mysql.user WHERE user = 'monitor';
+-----+-----+
| User   | Host    |
+-----+-----+
| monitor | localhost |
+-----+-----+
1 row in set (0,002 sec)
```

## DEUXIÈME MÉTHODE CONFIG MARIADB :

On commence par installer notre paquet pour mariadb sans oublier de faire la mise à jour du paquet

```
monitor@lrmServeurSql:~$ sudo apt install mariadb-server -y
[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
mariadb-server est déjà la version la plus récente (1:10.11.6-0+deb12u1).
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

On se connecte en root pour ajouter notre utilisateur “monitor” a mariadb

```
monitor@lrmServeurSql:~$ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 33
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.
```

La première commande permet d'ajouter notre utilisateur “monitor” a la base de données Mariadb en lui octroyant l'ensemble des privilèges sur toutes les bases de données et table du serveur. Ce qui

permet à l'utilisateur "monitor" d'effectuer des actions d'administration complète.

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

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

MariaDB [(none)]> SELECT user, host FROM mysql.user WHERE user = 'monitor';
+-----+-----+
| User | Host |
+-----+-----+
| monitor | localhost |
+-----+-----+
1 row in set (0,001 sec)
```



On vérifie si l'utilisateur a bien été créé avec cette commande

On test de se connecter avec monitor.

```
monitor@lrmServeurSql:~$ sudo mysql -u monitor -p
[sudo] Mot de passe de monitor :
Désolé, essayez de nouveau.
[sudo] Mot de passe de monitor :
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 34
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.
```

## 1.2 Configuration des accès SSH avec le compte "monitor"

### Authentification par clé SSH c'est quoi ?

Cette étape est importante pour la suite de notre projet car l'authentification par clé SSH est indispensable pour les scripts automatisés dans des environnements multi-VM, car elle permet de sécuriser les connexions et d'automatiser les tâches sans intervention humaine. Elle garantit une authentification forte et évite les failles liées aux mots de passe

Elle utilise un mécanisme de vérification bidirectionnelle :

Le client est authentifié par le serveur, et le serveur est également authentifié par le client. Cela réduit les risques de connexion à un serveur non légitime.

Enfin, l'association d'une clé SSH à chaque membre du groupe permet de tracer les actions effectuées sur les serveurs, facilitant ainsi la gestion et la responsabilité.

On commence sur notre vm :

Créer un répertoire .ssh avec l'utilisateur monitor et donner lui les droits suivants :

```
monitor@lmrFTP:~$ mkdir ~/.ssh
monitor@lmrFTP:~$ ls -la ~/.ssh
ls: impossible d'accéder à '/home/monitor/.ssh': Aucun fichier ou dossier de ce type
monitor@lmrFTP:~$ chmod 700 ~/.ssh
monitor@lmrFTP:~$ ls -la ~/.ssh
total 8
drwxr-xr-x 2 monitor monitor 4096 19 sept. 16:34 .
drwx----- 3 monitor monitor 4096 19 sept. 16:34 ..
monitor@lmrFTP:~$ chmod 700 ~/.ssh
monitor@lmrFTP:~$ ls -la ~/.ssh
total 8
drwxr-xr-x 2 monitor monitor 4096 19 sept. 16:34 .
drwx----- 3 monitor monitor 4096 19 sept. 16:34 ..
```

Créer un fichier authorized\_keys qui stockera nos clés :

```
monitor@lmrFTP:~$ sudo touch ~/.ssh/authorized_keys
[sudo] Mot de passe de monitor :
monitor@lmrFTP:~$ ls -la ~/.ssh/authorized_keys
-rw-r--r-- 1 root root 0 19 sept. 16:49 /home/monitor/.ssh/authorized_keys
monitor@lmrFTP:~$ chmod 600 ~/.ssh/authorized_keys
chmod: modification des droits de '/home/monitor/.ssh/authorized_keys': Opération non permise
monitor@lmrFTP:~$ sudo chmod 600 ~/.ssh/authorized_keys
monitor@lmrFTP:~$ ls -la ~/.ssh/authorized_keys
-rw----- 1 root root 0 19 sept. 16:49 /home/monitor/.ssh/authorized_keys
```

Si jamais pour être sûr que l'utilisateur monitor soit le propriétaire :

```
chown -R monitor:monitor /home/monitor/.ssh
```

⚠️ On se rend maintenant sur notre hôte, pour générer la clé ssh.

⚠ il est important de ne pas se connecter au serveur pour se faire. on vérifie juste que l'on peut se connecter avec le mdp et on sort

Maintenant on va générer les clé public et privé avec la commande suivante:

```
PS C:\Users\ritt1> ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\ritt1/.ssh/id_rsa):
C:\Users\ritt1/.ssh/id_rsa already exists.
Overwrite (y/n)? yes
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\ritt1/.ssh/id_rsa
Your public key has been saved in C:\Users\ritt1/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:baQnGEo1KkNk/6jM2uu3h9GCORUdo1MrnTtA+9gW8xQ ritt1@manon
The key's randomart image is:
+---[RSA 4096]---+
| .+ ...*.E
| o o.B.= .
|   o X.B . .
|   +. @ 0 +
|   ++.0 S +
| o+.o... +
| +. +
| o   o .
| ..+o.o
+---[SHA256]---+
```

Explication de la commande `ssh-keygen -t rsa -b 4096` :

Elle permet de générer une paire de clés SSH pour sécuriser les connexions à distance.

Elle génère 2 fichiers:

Clé privée : Stockée sur ton PC local, elle ne doit jamais être partagée.

Clé publique : Doit être copiée sur le serveur distant dans le fichier `~/.ssh/authorized_keys` pour permettre l'authentification sans mot de passe.

1. `ssh-keygen` :

- Outil utilisé pour créer et gérer des paires de clés SSH, qui permettent de se connecter à un serveur sans utiliser de mot de passe.

2. `-t rsa` (Type d'algorithme) :

- Spécifie le type d'algorithme à utiliser, ici RSA. RSA est un algorithme de chiffrement populaire basé sur la difficulté de factorisation de grands nombres premiers, utilisé pour l'authentification SSH.
- RSA est compatible avec la plupart des systèmes.

### 3. **-b 4096** (Taille de la clé) :

- Indique la taille de la clé en bits. Une clé de 4096 bits est très sécurisée et résistante aux attaques par force brute.
- Plus la clé est longue, plus elle est sécurisée, mais cela augmente légèrement le temps de calcul lors des connexions.

On vérifie que les fichiers avec les clés soit bien créés sur notre pc local

```
PS C:\Users\ritt1> cd .ssh
PS C:\Users\ritt1\.ssh> ls

Répertoire : C:\Users\ritt1\.ssh

Mode                LastWriteTime         Length  Name
----                -----          3381 id_rsa
-a----        19/09/2024      20:54           738 id_rsa.pub
```

Maintenant on va copier la clé public sur notre serveur pour ça

```
PS C:\Users\ritt1\.ssh> scp /Users/ritt1/.ssh/id_rsa.pub monitor@192.168.217
.131:/tmp/id_rsa.pub
monitor@192.168.217.131's password:
id_rsa.pub                                100%   738    212.9KB/s   00:00
PS C:\Users\ritt1\.ssh> ssh monitor@192.168.217.131
```

on se connecte au serveur avec notre mdp (une dernière fois ) et on copie la clé dans le fichier authorized\_keys.

```
monitor@lrmServeurFtp:~$ cat /tmp/id_rsa.pub >> ~/.ssh/authorized_keys
monitor@lrmServeurFtp:~$ chmod 600 ~/.ssh/authorized_keys
monitor@lrmServeurFtp:~$ chmod 700 ~/.ssh
monitor@lrmServeurFtp:~$ exit
```

on retourne dans le fichier sshd\_config on decommente "PubkeyAuthentication" et on le passe à yes pour s'authentifier avec les clés et on passe "password authentication" en no

```
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 no
#PermitEmptyPasswords no
```

on redémarre le service sshd et on retourne sur l'hôte pour tester

```
PS C:\Users\ritt1l> ssh monitor@192.168.217.131
Linux lrmServeurFtp 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 19 21:11:51 2024 from 192.168.217.1
```

## 2. VM dédiée aux scripts Python

### 2.1 Création d'une VM Debian avec Python

Nous allons créer un vm debian sans interface graphique avec python, les outils Mariadb ( client) et client ftp pour nous permettre d'envoyer des mails en python.

Pour se faire on commence par créer notre vm debian comme d'habitude que l'on nommera "lrmScript"

#### MISE EN PLACE SSH AVEC AUTHENTIFICATION PAR CLÉS DES 3 SERVEURS :

Dans la vm serveur dans l'exemple on a choisi la vm "lrmServeurFtp", on se rend dessus et on va dans le fichier /etc/ssh/sshd\_config pour remettre l'accès par mot de passe sans ça impossible de configurer la nouvelle clés.

```
#LoginGraceTime 2m
PermitRootLogin no
#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
```

⚠️ On pense bien à redémarrer le système ssh pour qu'il prenne en compte le changement.  
Maintenant nous allons faire l'authentification pour ssh en publickey sur notre vm "lrmScript"

On va maintenant générer les clés public et privé.

```
root@lrmScript:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:dDuh+IDSkslN80x/60j8qX1hd0EJm3Q5HQPHRj0rDr8 root@lrmScript
The key's randomart image is:
+---[RSA 4096]---+
|          oo=Oo|
|          . +*++|
|  o . . o o...o|
| . * * + + o ...|
| * + = S + . ..|
|  o   = .o+. . |
|      . o. oo.|
|  o . . . .|
|      +o+. E.|
+---[SHA256]---+
root@lrmScript:~#
```

⚠️ Commande pour copier clé public sur la vm “*lrmServeurFtp*”

```
root@lrmScript:~# ssh-copy-id monitor@192.168.217.131
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already present
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to enter the remote host's password:
monitor@192.168.217.131's password:

Number of key(s) added: 1

Now try logging into the machine, with:    "ssh 'monitor@192.168.217.131'"
and check to make sure that only the key(s) you wanted were added.
```

Puis on vérifie en se connectant avec *monitor* est l'adresse ip correspondant au serveur.

```
root@lrmScript:~# ssh monitor@192.168.217.131
Linux lrmServeurFtp 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Sep 24 09:24:02 2024 from 192.168.217.135
monitor@lrmServeurFtp:~$
```

## **⚠ A REFAIRE POUR SERVEUR WEB ET SQLip SANS GÉNÉRER DE NOUVELLES CLÉS**

### INSTALLATION PYTHON 3.11 SUR VM\_SCRIPT :

Pour commencer on mets à jour nos paquet et on installe les dépendances nécessaires pour installer les bibliothèques et outils pour installer et compiler python

```
root@lrmScript:/home/monitor# apt install -y build-essential libssl-dev libbz2-dev libreadline-dev libssqlite3-dev wget curl llvm libgdbm-dev liblzma-dev python3-openssl git
Lecture des listes de paquets... Fait
```

Ensute on télécharge la dernière version de python avec la commande suivante

```
root@lrmScript:/home/monitor# wget https://www.python.org/ftp/python/3.11.4/Python-3.11.4.tgz
--2024-09-24 13:24:00--  https://www.python.org/ftp/python/3.11.4/Python-3.11.4.tgz
Résolution de www.python.org (www.python.org)... 146.75.52.223, 2a04:4e42:54::223
Connexion à www.python.org (www.python.org)|146.75.52.223|: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 »

Python-3.11.4.tgz 100%[=====] 25,30M 38,8MB/s    ds 0,7s
2024-09-24 13:24:02 (38,8 MB/s) - « Python-3.11.4.tgz » sauvegardé [26526163 /26526163]
```

Après on décomprime notre fichier

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

Une fois dans le répertoire de la version de python que nous venons de télécharger, nous allons maintenant commencer le processus pour compiler notre programme et on commence par exécuter la commande suivante :

```
root@lrmScript:/home/monitor# cd Python-3.11.4
root@lrmScript:/home/monitor/Python-3.11.4# ./configure --enable-optimizations
```

**.configure** permet d'adapter le code source à ton système avant de passer à la compilation

**--enable-optimizations** permet d'obtenir une version de python qui est optimisée pour la performance si on souhaite par exemple exécuter des applications qui exigent des performances élevées.

Après on compile notre programme avec la commande suivante:

```
root@lrmScript:/home/monitor/Python-3.11.4# make -j $(nproc)
```

**make** est utilisé pour automatiser la compilation et la construction d'un programme

**-j \$(nproc)** : L'option **-j** permet à **make** d'effectuer la compilation en parallèle. **\$(nproc)** est une commande qui retourne le nombre de coeurs de processeur disponibles sur votre machine. En spécifiant cette option, vous permettez à **make** d'utiliser tous les coeurs de votre processeur pour accélérer le processus de compilation. Cela peut réduire le temps nécessaire pour construire le programme.

## Compilation séquentielle (sans -j)

Lorsque vous exécutez simplement **make**, sans l'option **-j**, le programme compile les fichiers un à la fois, en suivant l'ordre des dépendances dans le fichier **Makefile**. Cela signifie que chaque tâche (chaque fichier à compiler) est effectuée séquentiellement.

**make altinstall** Il est utilisé pour installer une nouvelle version de Python sans perturber les versions existantes sur le système. C'est particulièrement utile lorsque vous avez besoin de plusieurs versions de Python pour différents projets ou lorsque le système dépend d'une version spécifique de Python.

On peut vérifier que **python** se soit bien installé avec la commande suivante

```
root@lrmScript:/home/monitor/Python-3.11.4# python3.11 --version
Python 3.11.2
```

On teste si ça fonctionne de la manière suivante

```
root@lrmScript:/# 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 ("hello")
hello
>>>
```

Maintenant on va créer un environnement virtuel

## c'est quoi un environnement virtuel et à quoi ça sert ?

Un environnement virtuel est un espace isolé où vous pouvez installer des modules Python spécifiques à un projet, sans affecter les autres projets ou la configuration globale de Python sur votre machine. Voici pourquoi c'est utile :

- **Isolation des dépendances** : Chaque projet peut avoir ses propres versions des bibliothèques Python, évitant les conflits entre projets. Par exemple, un projet peut utiliser Django 3.2, tandis qu'un autre utilise Django 4.0.
- **Expérimentation sans risque** : Vous pouvez tester des bibliothèques ou des configurations spécifiques sans risquer de casser l'installation Python globale de votre système.
- **Portabilité** : Les environnements virtuels permettent de partager un projet avec ses dépendances spécifiques à travers des fichiers comme **requirements.txt**. Ainsi, quelqu'un d'autre peut recréer exactement les mêmes conditions sur une autre machine.

```
root@lrmScript:~# python3 -m venv mon_environnement
root@lrmScript:~# source mon_environnement/bin/activate
(mon_environnement) root@lrmScript:~# _
```

## INSTALLATION & CONFIG MARIADB CLIENT :

On installe le paquet de mariadb pour le client avec la commande suivante

```
root@lrmScript:/# apt install mariadb-client -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 :
  libconfig-inifiles-perl libdbd-mariadb-perl libdbi-perl libmariadb3
  libterm-readkey-perl mariadb-client-core mariadb-common mysql-common
Paquets suggérés :
  libclone-perl libmldb-perl libnet-daemon-perl libsql-statement-perl
Les NOUVEAUX paquets suivants seront installés :
  libconfig-inifiles-perl libdbd-mariadb-perl libdbi-perl libmariadb3
  libterm-readkey-perl mariadb-client mariadb-client-core mariadb-common
  mysql-common
0 mis à jour, 9 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

Vérifier simplement que l'on puisse se connecter au serveur via "lrmScript"

```
root@lrmScript:/# mariadb -u monitor -h 192.168.217.133 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 33
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)]>
```

## FTP CLIENT : ip a ip

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.

#### OUTIL D'ENVOI EMAIL :

Pour envoyer des Emails en python sur debian il y a plusieurs possibilités, l'une des plus populaire est “SMTPLIB” bibliothèque intégrée à python.

## 3. SCRIPTING SSH

### 3.1 ssh\_login.py : Connexion SSH et exécution d'une commande

A REVOIR Ci-dessous un script qui permet d'afficher (commande “ls”) ce qu'il y a dans le répertoire d'un des serveurs de notre choix. Notre script importe la **bibliothèque “paramiko”** qui permet au script d'interagir avec les serveurs distants en utilisant le protocole SSH. Ainsi, le programme Python peut se connecter, exécuter des commandes ou transférer des fichiers de manière sécurisée.

Il importe également le **module “sys”** qui permet de récupérer les arguments passés à notre script

depuis la ligne de commande.

```
import paramiko
import sys

def ssh_login(hostname, username, key_file, directory):
    # Créer une instance de SSHClient
    client = paramiko.SSHClient()
    # Charger les clés par défaut
    client.load_system_host_keys()
    # Ajouter automatiquement la clé de l'hôte si elle n'est pas dans les clés de système
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:
        # Connexion au serveur
        client.connect(hostname, username=username, key_filename=key_file)

        # Exécuter la commande ls pour lister le contenu du répertoire
        command = f"ls {directory}"
        stdin, stdout, stderr = client.exec_command(command)

        # Lire les résultats de la commande
        print(f"Contenu du répertoire {directory}:")
        for line in stdout:
            print(line.strip())

        # Lire les erreurs si elles existent
        for line in stderr:
            print("Erreur:", line.strip())
    except Exception as e:
        print(f"Une erreur est survenue : {e}")
    finally:
        client.close()

if __name__ == "__main__":
    if len(sys.argv) != 5:
        print("Usage: python ssh_login.py <hostname> <username> <key_file> <directory>")
        sys.exit(1)

    hostname = sys.argv[1]
    username = sys.argv[2]
    key_file = sys.argv[3]
    directory = sys.argv[4]

    ssh_login(hostname, username, key_file, directory)
```

Je crée ensuite à la racine un environnement Python que je vais appeler "myenv".

```
monitor@lrmScript:~$ python3 -m venv ~/myenv
```

Puis j'active l'environnement.

```
monitor@lrmScript:~$ source ~/myenv/bin/activate
```

Enfin j'active mon script avec la commande suivante.

```
(myenv) monitor@lrmScript:~$ python ssh_login.py 192.168.72.138 monitor ~/.ssh/id_rsa /etc/apache2/
```

- **ssh\_login.py** : nom du script
- **192.168.72.138** : adresse IP du serveur avec lequel on souhaite nous connecter, il s'agit ici du serveur FTP
- **monitor** : Il s'agit de l'utilisateur qui va se connecter au serveur

- `~/.ssh/id-rsa` : Chemin de la clé située sur le serveur local qui va nous permettre de binder la clé située sur le serveur FTP (le serveur distant)
- `/etc/apache2/` : Chemin que je souhaite afficher

Ci-dessous le résultat du script.

```
Contenu du répertoire /etc/apache2/:
apache2.conf
conf-available
conf-enabled
envvars
magic
mods-available
mods-enabled
ports.conf
sites-available
sites-enabled
```

Pour sortir de l'environnement, tapez la commande ci-dessous.

```
(myenv) monitor@lrmScript:~$ deactivate
```

## SCRIPT seconde méthode avec shebang et clé ssh dans script

C'est quoi "SHEBANG"?

Il s'agit de la première ligne `#!` suivie du chemin `/usr/bin/env`, elle va nous permettre d'exécuter le script directement avec `./nom_du_script`.

Il faudra simplement s'assurer que le script a les permissions d'exécution avec la commande suivante: `chmod +x nom_du_script.py`

## Quel intérêt d'utiliser le SHEBANG ?

Automatisation de l'interpréteur :

- Le shebang indique quel programme (interpréteur) doit être utilisé pour exécuter le script. Dans le cas de `#!/usr/bin/env python3`, cela indique au système d'utiliser **Python 3** pour interpréter et exécuter le script.

Portabilité :

- Utiliser un chemin comme `#!/usr/bin/env python3` permet également de localiser automatiquement l'interpréteur Python 3 à l'aide de la commande env, peu importe où il est installé dans le système.
- Cela garantit que le bon interpréteur sera utilisé même si son emplacement exact varie d'un système à un autre (exemple : /usr/bin/python3 sur un système, /usr/local/bin/python3 sur un autre).

```

❸ ssh_login.py > ⌂ ssh_connexion
1  #!/usr/bin/env python3
2  import paramiko
3  port = 22
4  key_file = "/home/monitor/.ssh/id_rsa"
5  # Fonction pour se connecter et exécuter une commande pour un serveur SSH
6  def ssh_connexion(hostname, username, key_file, command):
7      try:
8          # Créer un objet SSH
9          client = paramiko.SSHClient()
10
11         # Charger la clé du serveur s'il ne figure pas dans known_hosts
12         client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
13
14         # Charger la clé privée
15         private_key = paramiko.RSAKey.from_private_key_file(key_file)
16
17         # Connexion au serveur en utilisant la clé publique
18         client.connect(hostname, port, username, pkey=private_key)
19
20         # Exécution de la commande
21         stdin, stdout, stderr = client.exec_command(command)
22
23         # Lire et afficher la sortie standard (stdout)
24         output = stdout.read().decode('utf-8').strip() # .strip() pour enlever les espaces
25         # Lire et afficher les erreurs, le cas échéant (stderr)
26         errors = stderr.read().decode('utf-8').strip()
27
28         if output:
29             print(f"Résultat de la commande '{command}' sur {hostname}:\n{output}")
30         if errors:
31             print(f"Erreurs rencontrées lors de l'exécution de la commande '{command}' :\n{errors}")
32
33         # Fermer la connexion
34         client.close()
35
36     except Exception as e:
37         print(f"Une erreur est survenue: {e}")
38
39
40 if __name__ == "__main__":
41     hostname = input("Entrez l'adresse IP ou le nom du serveur : ")
42     username = input("Entrez le nom d'utilisateur SSH : ")
43     # Demander la commande à exécuter
44     command = input("Entrez la commande shell à exécuter (ex: 'df -h', 'ls') : ")
45
46     # Appeler la fonction pour se connecter et exécuter la commande
47     ssh_connexion(hostname, username, key_file, command)
48

```

## Explication du script :

`client = paramiko.SSHClient()` : Cette ligne crée un objet SSH avec la bibliothèque paramiko. Cet objet client sera utilisé pour gérer la connexion SSH au serveur distant.

- **Paramiko** : C'est une bibliothèque Python permettant d'établir des connexions SSH et d'exécuter des commandes à distance via le protocole SSH.

`client.set_missing_host_key_policy(paramiko.AutoAddPolicy()):` Cette ligne indique à paramiko quoi faire si le serveur distant (le serveur auquel on essaie de se connecter) n'est pas encore présent dans le fichier known\_hosts. Ce fichier contient les clés publiques des serveurs auxquels on s'est déjà connecté.

- Ici, avec `AutoAddPolicy()`, le script acceptera automatiquement et ajoutera la clé du serveur à la liste des hôtes de confiance sans demander une confirmation.
- Sans cette ligne, paramiko lèverait une exception et refuserait la connexion si le serveur est inconnu (c'est-à-dire s'il n'est pas déjà dans le fichier known\_hosts).

`private_key = paramiko.RSAKey.from_private_key_file(key_file)` : Cette ligne lit et charge une clé privée à partir d'un fichier spécifié par key\_file.

- Ici, `key_file` est le chemin vers le fichier de clé privée `/home/monitor/.ssh/id_rsa`.
- `RSAKey` : Il s'agit d'un type de clé utilisée dans l'authentification par clé publique SSH (au lieu d'utiliser un mot de passe). Cette méthode est généralement plus sécurisée que l'authentification par mot de passe.

`client.connect(hostname, port, username, pkey=private_key)`: Cette ligne permet d'établir une connexion ssh.

`stdin, stdout, stderr = client.exec_command(command)` : Cette ligne exécute une commande shell sur le serveur distant via SSH.

- `command` : La commande shell que tu veux exécuter (par exemple : df -h, ls, etc.).
- `stdin` : Flux d'entrée, utilisé si tu veux envoyer des données supplémentaires à la commande.
- `stdout` : Flux de sortie, c'est là où sera stockée la sortie standard de la commande (par exemple, le résultat de la commande ls).
- `stderr` : Flux d'erreurs, c'est là où sera stockée la sortie des erreurs de la commande, s'il y en a.

`stdout.read()` : Cela lit la sortie de la commande exécutée (le résultat renvoyé par la commande que tu as exécutée sur le serveur distant).

`.decode()` : La sortie est souvent sous forme de bytes (octets), donc la méthode .

Le second permet de convertir cette sortie en chaîne de caractères (string) compréhensible.

Nous lançons dans un premier temps notre script puis on rentre les informations concernant l'adresse ip du serveur,utilisateur ainsi que la commande que nous souhaitons faire.

⚠️ on pense à se mettre dans l'environnement où paramiko a été implémenté dans cette démonstration, il s'agit de mon\_environnement pour y accéder il suffit de faire la commande suivante: source nom-de-l'environnement/bin/activate

```
● (mon_environnement) root@lrmScript:~# /root/mon_environnement/bin/python /root/ssh_login.py
Entrez l'adresse IP ou le nom du serveur : 192.168.217.132
Entrez le nom d'utilisateur SSH : monitor
Entrez la commande shell à exécuter (ex: 'df -h', 'ls') : ls /etc/
Résultat de la commande 'ls /etc/' sur 192.168.217.132:
adduser.conf
adjtime
```

### 3.2 ssh\_login\_sudo.py : Connexion SSH et exécution d'une commande en mode sudo

SCRIPT A REVOIR Le script ci-dessous me permet de faire n'importe quelle commande qui nécessite “sudo”. Il importe le **module “getpass”**, qui est utilisé pour gérer la saisie sécurisée de mots de passe

d'autres informations sensibles dans la console, sans les afficher à l'écran.

```
import paramiko
import sys
import getpass

def ssh_login_sudo(hostname, username, key_file, command):
    # Créer une instance de SSHClient
    client = paramiko.SSHClient()
    # Charger les clés par défaut
    client.load_system_host_keys()
    # Ajouter automatiquement la clé de l'hôte si elle n'est pas dans les clés de système
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:
        # Connexion au serveur
        client.connect(hostname, username=username, key_filename=key_file)

        # Demander le mot de passe sudo à l'utilisateur
        sudo_password = getpass.getpass(prompt="Entrez votre mot de passe sudo: ")

        # Préparer la commande sudo
        sudo_command = f"sudo -S {command}"

        # Exécuter la commande sudo
        stdin, stdout, stderr = client.exec_command(sudo_command)

        # Fournir le mot de passe sudo
        stdin.write(sudo_password + '\n')
        stdin.flush()

        # Lire les résultats de la commande
        print(f"Résultats de la commande '{command}':")
        for line in stdout:
            print(line.strip())

        # Lire les erreurs si elles existent
        for line in stderr:
            print("Erreur:", line.strip())
    except Exception as e:
        print(f"Une erreur est survenue : {e}")
    finally:
        client.close()

if __name__ == "__main__":
    if len(sys.argv) != 5:
        print("Usage: python ssh_login_sudo.py <hostname> <username> <key_file> <command>")
        sys.exit(1)

    hostname = sys.argv[1]
    username = sys.argv[2]
    key_file = sys.argv[3]
    command = sys.argv[4]

    ssh_login_sudo(hostname, username, key_file, command)
```

Je tape la commande suivante, On part du même principe que la commande du script précédent, seule la commande à la fin de la ligne change.

En effet, je demande au script de faire la commande “mkdir ~ /DossierLrm”, ce qui me permet de créer le répertoire cité à la racine. Je dois ensuite entrer le MdP sudo et enfin un print de confirmation s'affiche.

```
(myenv) monitor@lrmScript:~$ python ssh_login_sudo.py 192.168.72.138 monitor ~/.ssh/id_rsa
"mkdir ~ /DossierLrm"
Entrez votre mot de passe sudo:
Résultats de la commande 'mkdir ~ /DossierLrm': OK
```

Je vais ensuite vérifier dans le serveur FTP si la commande a bien été exécutée.

```
monitor@lrmServeurFtp:~$ ls
DossierLrm
```

Seconde méthode avec chiffrage mdp

```
❶ ssh_login.py | ❷ ssh_login_sudo.py X
❸ ssh_login_sudo.py > ⌂ ssh_connexion
1  #!/usr/bin/env python3
2  import paramiko
3  import getpass
4  import hashlib
5
6  port = 22
7  key_file = "/home/monitor/.ssh/id_rsa"
8
9  # Fonction pour se connecter et exécuter une commande pour un serveur SSH
10 def ssh_connexion(hostname, username, key_file, command, sudo_password):
11     try:
12         # Créer un objet SSH
13         client = paramiko.SSHClient()
14
15         # Charger la clé du serveur s'il ne figure pas dans known_hosts
16         client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
17
18         # Charger la clé privée
19         private_key = paramiko.RSAKey.from_private_key_file(key_file)
20
21         # Connexion au serveur en utilisant la clé publique
22         client.connect(hostname, port, username, pkey=private_key)
23
24         # Préfixer la commande par 'sudo'
25         sudo_command = f"sudo -S {command}" # -S pour lire le mot de passe depuis stdin
26
27         # Exécution de la commande avec sudo et envoi du mot de passe via stdin
28         stdin, stdout, stderr = client.exec_command(sudo_command)
29         stdin.write(sudo_password + '\n')
30         stdin.flush()
31
32         # Lire et afficher la sortie standard (stdout)
33         output = stdout.read().decode('utf-8').strip()
34         # Lire et afficher les erreurs, le cas échéant (stderr)
35         errors = stderr.read().decode('utf-8').strip()
36
37         if output:
38             print(f"Résultat de la commande '{command}' sur {hostname}:\n{output}")
39         if errors:
40             print(f"Erreurs rencontrées lors de l'exécution de la commande '{command}' :\n{errors}")
41
42         # Fermer la connexion
43         client.close()
44
45     except Exception as e:
46         print(f"Une erreur est survenue: {e}")
47
48
49 if __name__ == "__main__":
50     hostname = input("Entrez l'adresse IP ou le nom du serveur : ")
51     username = input("Entrez le nom d'utilisateur SSH : ")
52     sudo_password = getpass.getpass("Entrez le mot de passe sudo : ")
53     hashedsudo_password = hashlib.md5(sudo_password.encode()).hexdigest()
54     print(f"Le mot de passe sudo hashé est : {hashedsudo_password}")
55
56     # Demander la commande à exécuter
57     command = input("Entrez la commande shell à exécuter (ex: 'df -h', 'ls') : ")
58
59     # Appeler la fonction pour se connecter et exécuter la commande avec sudo
60     ssh_connexion(hostname, username, key_file, command, sudo_password)
```

Dans le script, on ajoute la bibliothèque “**getpass**” qui permet de saisir un mdp de manière sécurisée sans afficher les caractères à l'écran garantissant la confidentialité des informations sensibles. Ainsi que la bibliothèque “**hashlib**” qui permet de créer des hachages sécurisés. Hashlib supporte plusieurs algorithmes comme SHA-1, SHA-256, SHA-512 et MD5.

**MD5** : Rapide mais peu sécurisé (128 bits).

**SHA-1** : Plus sécurisé que MD5, mais vulnérable aux collisions (160 bits).

**SHA-256** : Très sécurisé, utilisé dans des contextes critiques (256 bits).

**SHA-512** : Offre un niveau de sécurité élevé, adapté aux applications très sensibles (512 bits).

On lance notre script, entrons les informations nécessaires et nous testons avec une commande que seul un sudo ou root peut utiliser, nous avons choisi la commande qui nous afficher le fichier **SHADOW**

Le fichier **shadow** contient les informations des comptes utilisateurs comme le mdp hashé, la date d'expiration mdp, date de modification mdp, durée de vie max du mdp

```
(mon_environnement) root@lrmScript:~# ./ssh_login_sudo.py
Entrez l'adresse IP ou le nom du serveur : 192.168.217.133
Entrez le nom d'utilisateur SSH : monitor
Entrez le mot de passe sudo :
Le mot de passe sudo hashé est : c3dae848d72c51cf97e24014d47591cd
Entrez la commande shell à exécuter (ex: 'df -h', 'ls') : cat /etc/shadow
Résultat de la commande 'cat /etc/shadow' sur 192.168.217.133:
root:!:19985:0:99999:7:::
daemon:*:19985:0:99999:7:::
bin:*:19985:0:99999:7:::
sys:*:19985:0:99999:7:::
```

## 4. Scripting MariaDB/MySQL :

### 4.1 ssh\_mysql.py : Connexion au serveur MariaDB/MySQL et vérification d'accès

Pour commencer on installe la bibliothèque qui va nous permettre de se connecter à une base de données de mariadb ou mysql ainsi qu' exécuter des requêtes SQL donc primordial pour que notre script fonctionne.

```
(mon_environnement) root@lrmScript:~# pip install pymysql
Collecting pymysql
  Downloading PyMySQL-1.1.1-py3-none-any.whl (44 kB)
    ━━━━━━━━━━━━━━━━ 45.0/45.0 kB 2.2 MB/s eta 0:00:00
Installing collected packages: pymysql
Successfully installed pymysql-1.1.1
```

Voici le script qui va nous permettre de se connecter à notre serveur mariaDb et de lui faire une requête pour ça, nous avons choisi simplement la requête “SHOW DATABASE” pour nous montrer les bases données présente dans mariadb

```

❶ ssh_mysql.py > ...
1  #!/usr/bin/env python3
2  import paramiko
3  import pymysql
4  import getpass
5
6  portSsh = 22
7  chemin_cle = "/home/monitor/.ssh/id_rsa"
8  portMariadb = 3306
9
10 # Fonction pour établir une connexion SSH
11 def ssh_connexion(hostname, username, chemin_cle):
12     ssh = paramiko.SSHClient()
13     ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
14     cle_privee = paramiko.RSAKey.from_private_key_file(chemin_cle)
15     ssh.connect(hostname, port=portSsh, username=username, pkey=cle_privee)
16
17     print("Connexion SSH réussie à l'hôte: ", hostname)
18     return ssh
19
20 # Fonction pour établir une connexion MySQL
21 def mysql_connexion(hostname, slq_username,sql_password, portMariadb):
22     try:
23         connection = pymysql.connect(host=hostname, user=slq_username, password=sql_password, port=portMariadb)
24         print("Connexion MySQL réussie à l'hôte: ", hostname)
25         return connection
26
27     # Gestion des erreurs
28     except Exception as e:
29         print("Erreur de connexion à la base de données MySQL: ", e)
30         return None
31
32 # Fonction principale pour exécuter le programme
33 def main():
34     hostname = input("Entrez le nom d'hôte: ")
35     username = input("Entrez le nom d'utilisateur: ")
36     sql_username = input("Entrez le nom d'utilisateur MySQL: ")
37     sql_password = getpass.getpass("Entrez le mot de passe MySQL: ")
38
39     ssh = ssh_connexion(hostname, username, chemin_cle)
40     connection = mysql_connexion(hostname, sql_username, sql_password, portMariadb)
41
42     if mysql_connexion:
43         try:
44             with connection.cursor() as cursor:
45                 cursor.execute("SHOW DATABASES")
46                 for db in cursor:
47                     print(db)
48         except Exception as e:
49             print("Erreur lors de l'exécution de la requête: ", e)
50     else:
51         print("La connexion à la base de données a échoué")
52
53     ssh.close()
54     connection.close()
55
56 # Appel de la fonction principale
57 if __name__ == "__main__":
58     main()

```

On exécute le script et voici le résultat :

```
(mon_environnement) root@lrmScript:~# ./ssh_mysql.py  
Entrez le nom d'hôte: 192.168.217.133  
Entrez le nom d'utilisateur: monitor  
Entrez le nom d'utilisateur MySQL: monitor  
Entrez le mot de passe MySQL:  
Connexion SSH réussie à l'hôte: 192.168.217.133  
Connexion MySQL réussie à l'hôte: 192.168.217.133  
('information_schema',)  
('mysql',)  
('performance_schema',)  
('sys',)
```

## 4.2 ssh\_mysql\_error.py : Récupération des erreurs d'accès et stockage dans la base SQL

Avant de commencer à faire créer une base de données et un script, nous allons **activer les logs de mariadb**.

Pour ce faire nous allons **créer le répertoire /var/log/mysql**.

```
monitor@lrmServeurSql:~$ sudo mkdir -p /var/log/mysql
```

**changer le propriétaire** du répertoire afin de le donner à “mysql”.

```
monitor@lrmServeurSql:/var/log$ sudo chown mysql mysql/
```

Dans le répertoire “mysql”, créer un fichier “mysql.log” et changer ici aussi le propriétaire.

```
monitor@lrmServeurSql:/var/log/mysql$ sudo touch mysql.log
```

```
monitor@lrmServeurSql:/var/log/mysql$ sudo chown mysql mysql.log
```

Nous pouvons maintenant activer les logs de mariadb, ces logs nous permettront de voir tout ce qui se passe dans l'environnement de mariadb. Nous pourrons ainsi voir les requêtes réalisées par un utilisateur de la base de données, mais aussi les **connexions fructueuses et infructueuses**, ce sont ces derniers qui vont nous intéresser car nous allons les extraire et les stocker dans une table que nous allons créer.

Pour ce faire nous allons décommenter deux petites lignes dans le fichier **/etc/mysql/mariadb.conf.d/50-server.cnf** comme ci-dessous.

```
# Both location gets rotated by the cronjob.  
# Be aware that this log type is a performance killer.  
# Recommend only changing this at runtime for short testing periods if needed!  
general_log_file      = /var/log/mysql/mysql.log  
general_log            = 1
```

Puis on redémarre mariadb.

```
monitor@lrmServeurSql:~$ sudo systemctl restart mariadb
```

Nous allons maintenant créer notre base de données qui va accueillir notre table.

Dans mariadb, je créer une base de données qui va se nommer “DB\_access”.

```
MariaDB [(none)]> CREATE DATABASE DB_access;
```

Une fois la base de données créée, je peux faire un check pour voir s' il a bien été créé.

```
MariaDB [(none)]> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| DB_access |  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
5 rows in set (0,001 sec)
```

Pour utiliser la base de données on tape la commande suivante.

```
MariaDB [(none)]> USE DB_access;  
Reading table information for completion of table and column names  
You can turn off this feature to get a quicker startup with -A  
  
Database changed  
MariaDB [DB_access]>
```

Enfin, on va créer notre table “access\_logs”.

```
MariaDB [DB_access]> CREATE TABLE access_logs (  
    ->     id INT(11) NOT NULL AUTO_INCREMENT,  
    ->     username VARCHAR(255) NOT NULL,  
    ->     attempt_time DATETIME NOT NULL,  
    ->     ip_address VARCHAR(255) NOT NULL,  
    ->     status VARCHAR(10) DEFAULT NULL,  
    ->     PRIMARY KEY (id)  
    -> );
```

On peut taper la commande suivante pour checker si elle a bien été créée.

```
MariaDB [DB_access]> SHOW TABLES;
+-----+
| Tables_in_DB_access |
+-----+
| access_logs          |
+-----+
1 row in set (0,001 sec)
```

Nous pouvons également avoir une description détaillée des colonnes.

```
MariaDB [DB_access]> DESCRIBE access_logs;
+-----+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra        |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)    | NO   | PRI | NULL     | auto_increment |
| username   | varchar(255)| NO   |     | NULL     |               |
| attempt_time | datetime  | NO   |     | NULL     |               |
| ip_address | varchar(255)| NO   |     | NULL     |               |
| status     | varchar(10) | YES  |     | NULL     |               |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0,009 sec)
```

```

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
11 ssh_host = config.DBipaddr # L'adresse IP ou le nom d'hôte du serveur distant
12 ssh_user = config.username # Le nom d'utilisateur SSH
13 ssh_password = config.keyfile # Le mot de passe SSH
14 log_file_path = '/var/log/mysql/mysql.log' # Chemin vers le fichier de log sur le serveur distant
15
16 # Connexion SSH
17 ssh_client = paramiko.SSHClient()
18 ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
19 ssh_client.connect(ssh_host, username=ssh_user, password=ssh_password)
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 "Access Denied"
26 connections = []
27 lines = log_content.splitlines()
28 i = 0
29
30 while i < len(lines) - 1:
31     current_line = lines[i]
32     next_line = lines[i + 1]
33
34     # Vérifie si la ligne courante contient une connexion et la suivante un "Access denied"
35     if "Connect" in current_line and "Access denied" in next_line:
36         connect_match = re.match(r"(\d{6}) (\d{2}:\d{2}:\d{2})\s+\d+\s+Connect\s+(\w+)\@([\w.-]+)\.*", current_line)
37         if connect_match:
38             date_str, time_str, username, ip_address = connect_match.groups()
39             # Convertir le format de date en YYYY-MM-DD HH:MM:SS
40             attempt_time = datetime.strptime(f"{date_str} {time_str}", "%Y-%m-%d %H:%M:%S").strftime("%Y-%m-%d %H:%M:%S")
41             connections.append((username, attempt_time, ip_address, 'failed'))
42
43     # Passe à la prochaine paire de lignes
44     i += 1
45
46 # Fonction pour insérer les tentatives de connexion échouées dans la base de données
47 def insert_login_attempts(connections):
48     with db.cursor() as cursor:
49         for connection in connections:
50             cursor.execute("INSERT INTO access_logs (username, attempt_time, ip_address, status) VALUES (%s, %s, %s, %s)", connection)
51             print(f"Insertion: {connection}") # Debug: Montre chaque connexion à insérer
52     db.commit()
53     print(f"{len(connections)} enregistrements ont été insérés avec succès.")
54
55 # Insérer les connexions dans la base de données
56 insert_login_attempts(connections)
57
58 # Fermeture de la connexion
59 ssh_client.close()
60 db.close()

```

Ce script permet de récupérer les tentatives de connexions échouées qui sont dans le fichier de log `/var/log/mysql/mysql.log`, et les enregistres ensuite dans la table “**access\_logs**” qui est située dans la base de donnée “**DB\_access**”.

Ci-dessous les résultats affichés dans le terminal et la tables “access\_logs”.

```
(myenv) monitor@lrmScript:~$ /home/monitor/myenv/bin/python /home/monitor/ssh_mysql_error.py
Insertion: ('Manu', '2024-09-30 11:08:35', 'localhost', 'failed')
Insertion: ('monitor', '2024-09-30 11:13:04', 'localhost', 'failed')
Insertion: ('Rija', '2024-09-30 11:14:03', 'localhost', 'failed')
Insertion: ('monitor', '2024-09-30 11:15:46', 'localhost', 'failed')
4 enregistrements ont été insérés avec succès.
```

MariaDB [DB_access]> SELECT * FROM access_logs;					
id	username	attempt_time	ip_address	status	
1393	Manu	2024-09-30 11:08:35	localhost	failed	
1394	Manu	2024-09-30 11:08:35	localhost	failed	
1395	monitor	2024-09-30 11:13:04	localhost	failed	
1396	Manu	2024-09-30 11:08:35	localhost	failed	
1397	monitor	2024-09-30 11:13:04	localhost	failed	
1398	Rija	2024-09-30 11:14:03	localhost	failed	
1399	Manu	2024-09-30 11:08:35	localhost	failed	
1400	monitor	2024-09-30 11:13:04	localhost	failed	
1401	Rija	2024-09-30 11:14:03	localhost	failed	
1402	monitor	2024-09-30 11:15:46	localhost	failed	

methode 2 sans config et mdp chiffre

```
#!/usr/bin/env python3

import re

import pymysql

import paramiko

from datetime import datetime

import getpass


# Configuration des connexions

ssh_host = input("Entrez le nom d'hôte pour SSH: ")

ssh_user = input("Entrez le nom d'utilisateur SSH: ")

chemin_cle = "/home/monitor/.ssh/id_rsa" # Clé privée pour SSH

log_file_path = '/var/log/mysql/mysql.log' # Chemin vers le fichier de log MySQL

portSsh = 22
```

```
portMariadb = 3306

# Demander les informations pour MySQL

db_host = ssh_host # L'hôte est le même que celui du SSH
db_user = input("Entrez le nom d'utilisateur MySQL: ")
db_password = getpass.getpass("Entrez le mot de passe MySQL: ")
db_name = "erreur_log" # Base de données où sont stockées les erreurs

# Connexion SSH avec clé privée
ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(ssh_host, username=ssh_user, key_filename=chemin_cle,
port=portSsh)

# Connexion MySQL

db = pymysql.connect(host=db_host, user=db_user, password=db_password,
database=db_name, port=portMariadb)

# 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"
erreurs_acces = []

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")

        erreurs_acces.append((username, ip_address, attempt_time))

# Passe à la prochaine paire de lignes

i += 1


# Fonction pour insérer les erreurs dans la table erreurs_acces

def insert_erreurs_acces(erreurs_acces):

    with db.cursor() as cursor:

        for erreur in erreurs_acces:

            cursor.execute("INSERT INTO erreurs_acces (user, ip, date_time) VALUES (%s, %s, %s)", erreur)

            print(f"Insertion: {erreur}") # Debug: Montre chaque erreur à insérer

    db.commit()

    print(f"{len(erreurs_acces)} enregistrements ont été insérés avec succès.")


# Insérer les erreurs d'accès dans la base de données

if erreurs_acces:

    insert_erreurs_acces(erreurs_acces)

else:

    print("Aucune erreur d'accès trouvée dans les logs.")

```

```
# Fermeture des connexions  
ssh_client.close()  
db.close()
```

## 5. Scripting FTP : JOB07

### 5.1. ssh\_ftp\_error.py : Récupération des logs d'erreurs FTP et stockage dans la bd

Avant de créer le script et la table qui va accueillir les données, nous allons activer les logs de SSH. Pour ce faire nous allons installer “**rsyslog**”.

```
monitor@lrmServeurFtp:/var/log$ sudo apt install rsyslog
```

Nous allons maintenant configurer le fichier **/etc/ssh/sshd\_config** et décommenter les lignes “**SyslogFacility AUTH**” et “**LogLevel INFO**”.

```
# Logging  
SyslogFacility AUTH  
LogLevel INFO
```

Puis dans le répertoire **/var/log**, créez un fichier qui va se nommer “**auth.log**” et changer le propriétaire du fichier pour que le fichier appartienne à **sshd**.

```
monitor@lrmServeurFtp:/var/log$ sudo touch auth.log
```

```
monitor@lrmServeurFtp:/var/log$ sudo chown sshd auth.log
```

Enfin, redémarrez **sshd** et faites un “**cat**” de “**auth.log**” pour vérifier si le service écrit bien dans le fichier.

```
monitor@lrmServeurFtp:/var/log$ sudo systemctl restart sshd
```

```
monitor@lrmServeurFtp:/var/log$ cat auth.log  
2024-10-01T10:49:00.519318+02:00 lrmServeurFtp sudo: pam_unix(sudo:session):  
    session closed for user root  
2024-10-01T10:49:05.760815+02:00 lrmServeurFtp sudo: monitor : TTY=pts/0 ;  
PWD=/var/log ; USER=root ; COMMAND=/usr/bin/systemctl restart sshd  
2024-10-01T10:49:05.761458+02:00 lrmServeurFtp sudo: pam_unix(sudo:session):  
    session opened for user root(uid=0) by monitor(uid=1000)  
2024-10-01T10:49:05.769000+02:00 lrmServeurFtp sshd[1388]: Received signal 1  
5; terminating.
```

L'activation des logs est OK, youhou!

Dans le **serveur mariadb**, nous allons maintenant créer notre table de données de la même manière que nous l'avons fait dans le job précédent. Seul le nom de la table changera.

```
MariaDB [DB_access]> CREATE TABLE ftp_logs (
-> id INT(11) NOT NULL AUTO_INCREMENT,
-> username VARCHAR(255) NOT NULL,
-> attempt_time DATETIME NOT NULL,
-> ip_address VARCHAR(255) NOT NULL,
-> status VARCHAR(255) DEFAULT NULL,
-> PRIMARY KEY (id),
-> );
```

Et nous créons le script.

```
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 au serveur FTP
ssh_host = config.FTPipaddr # 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 = config.FTPlogfile # Chemin vers le fichier de log sur le serveur distant

# Configuration de la connexion SSH au serveur mariadb
ssh_host_db = config.DBipaddr # L'adresse IP ou le nom d'hôte du serveur distant
ssh_user_db = config.username # Le nom d'utilisateur SSH
ssh_password_db = config.keyfile # Le mot de passe SSH

# Connexion SSH au serveur FTP
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 FTP
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 "Failed password"
connections = []
lines = log_content.splitlines()
i = 0
```

Ci-dessous le résultat dans le terminal.

```
(myenv) monitor@lrmScript:~$ /home/monitor/myenv/bin/python /home/monitor/ssh_ftp_error.py
Insertion: ('Rija', '2024-10-01T10:53:15', '192.168.72.1', 'failed for invalid user')
Insertion: ('Yousef', '2024-10-01T14:26:10', '192.168.72.1', 'failed for invalid user')
Insertion: ('monitor', '2024-10-01T14:34:02', '192.168.72.1', 'failed for invalid password')
3 enregistrements ont été insérés avec succès.
```

Et ci-dessous le résultat dans la table.

```
MariaDB [DB_access]> SELECT * FROM ftp_logs;
+----+-----+-----+-----+-----+
| id | username | attempt_time | ip_address | status |
+----+-----+-----+-----+-----+
| 10 | Rija    | 2024-10-01 10:53:15 | 192.168.72.1 | failed for invalid user |
| 11 | Youcef   | 2024-10-01 14:26:10 | 192.168.72.1 | failed for invalid user |
| 12 | monitor   | 2024-10-01 14:34:02 | 192.168.72.1 | failed for invalid password |
+----+-----+-----+-----+
3 rows in set (0,006 sec)
```

## 6.Scripting Web : JOB08

### 6.1 ssh\_web\_error.py : Récupération des logs d'erreurs d'accès Web et stockage dans DB

Pour récupérer les logs d'erreurs de notre serveur web et les stocker dans la base de données.

Tout d'abord, on se rend sur notre serveur web. L'utilisateur “ monitor” doit avoir les permissions pour accéder au répertoire Apache2 qui contient le fichier qui nous intéresse “ **ERROR.LOG** ” que l'on voir ci dessous

```
monitor@lrmServeurWeb:/var/log/apache2$ ls
access.log      error.log      error.log.2.gz  error.log.4.gz  other_vhosts_access.log  other_vhosts_access.log.2.gz
access.log.1    error.log.1    error.log.3.gz  error.log.5.gz  other_vhosts_access.log.1
```

Faire cette commande pour donner la permission à l'utilisateur monitor d'accéder au répertoire Apache2, ainsi que les sous répertoire et fichier présent dans le répertoire spécifié.

```
monitor@lrmServeurWeb:/var/log/apache2$ sudo chown -R monitor:monitor /var/log/apache2/
```

Pour notre script, nous utiliserons les bibliothèques suivante, nous allons expliquer à quoi sert les 2 nouvelles utiliser

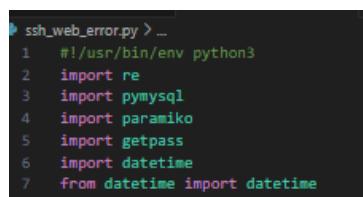
**La bibliothèque DATETIME:** permet de gérer efficacement les dates, heures et intervalles de temps en Python.

Ses classes Principales :

- **datetime.date** : Représente une date (année, mois, jour).
- **datetime.time** : Représente une heure (heures, minutes, secondes).
- **datetime.datetime** : Combine date et heure.
- **datetime.timedelta** : Représente une durée entre deux dates/heures.
- **datetime.tzinfo** : Gère les fuseaux horaires.

Ses fonctions Utiles :

- **strptime()** : Convertir une chaîne en objet datetime.
- **strftime()** : Convertir un objet datetime en chaîne.
- **combine()** : Combine une date et une heure en un objet datetime



```
❶ ssh_web_error.py > ...
1  #!/usr/bin/env python3
2  import re
3  import pymysql
4  import paramiko
5  import getpass
6  import datetime
7  from datetime import datetime
```

```

9  # Configuration des connexions
10 ssh_host = "192.168.217.132"
11 ssh_user = "monitor"
12 chemin_cle = "/home/monitor/.ssh/id_rsa"
13 log_file_path = '/var/log/apache2/error.log' # Chemin vers le fichier erreur log Apache
14 portSsh = 22
15 portMariadb = 3306
16
17 # Demander les informations pour MySQL
18 db_host = "192.168.217.133" # L'hôte de la base de données
19 db_user = "monitor"
20 db_password = getpass.getpass("Entrez le mot de passe MySQL: ")
21 db_name = "erreur_log" # Base de données où sont stockées les erreurs
22
23 # Connexion SSH avec clé privée
24 ssh_client = paramiko.SSHClient()
25 ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
26 ssh_client.connect(ssh_host, username=ssh_user, key_filename=chemin_cle, port=portSsh)
27 print("Connexion SSH réussie.")
28
29 # Connexion MySQL
30 db = pymysql.connect(host=db_host, user=db_user, password=db_password, database=db_name, port=portMariadb)
31 print("Connexion à la base de données réussie.")
32
33 # Lecture du fichier de log sur le serveur distant
34 stdin, stdout, stderr = ssh_client.exec_command(f"cat {log_file_path}")
35 log_content = stdout.read().decode('utf-8')
36
37 # Traitement du contenu du fichier de log pour les erreurs d'accès
38 erreurlog = []
39 lines = log_content.splitlines()
40
41 for line in lines:
42
43     # Vérifie si la ligne contient une erreur d'accès liée à l'authentification
44     if "AH01618: user" in line and "not found" in line:
45
46         # Expression régulière pour capturer les informations clés regex
47         log_match = re.match(r"\[(.*?\)] .* \[client ([\d\.]+):\d+] AH01618: user (\w+) not found:.+", line)
48         if log_match:
49             date_str, ip_address, username = log_match.groups()
50
51             # Convertir la date en format YYYY-MM-DD HH:MM:SS
52             try:
53                 attempt_time = datetime.strptime(date_str, "%a %b %d %H:%M:%S %Y").strftime("%Y-%m-%d %H:%M:%S")
54             except ValueError:
55                 attempt_time = datetime.strptime(date_str, "%a %b %d %H:%M:%S %Y").strftime("%Y-%m-%d %H:%M:%S")
56             erreurlog.append((username, attempt_time, ip_address, 'failed'))
57
58     # Fonction pour insérer les tentatives de connexion échouées dans la base de données
59     def insert_table_erreurs_log(lignes):
60         with db.cursor() as cursor:
61             for ligne in lignes:
62                 cursor.execute("INSERT INTO erreurs_log (username, attempt_time, ip_address, status) VALUES (%s, %s, %s, %s)", ligne)
63                 print(f"Insertion: {ligne}")
64         db.commit()
65         print(f"{len(erreurlog)} enregistrements ont été insérés avec succès.")
66
67     # Insérer les connexions dans la base de données
68     if erreurlog:
69         insert_table_erreurs_log(erreurlog)
70     else:
71         print("Aucune connexion à insérer.")
72
73     # Fermeture des connexions
74     ssh_client.close()
75     db.close()

```

```
(mon_environnement) root@lrmScript:~# /root/mon_environnement/bin/python /root/ssh_web_error.py
Entrez le mot de passe MySQL:
Connexion SSH réussie.
Connexion à la base de données réussie.
Regex matched the line successfully
Matched groups - Date: Mon Sep 30 16:02:15.679594 2024, IP: 192.168.217.1, Username: truc
Regex matched the line successfully
Matched groups - Date: Mon Sep 30 16:02:24.927595 2024, IP: 192.168.217.1, Username: muche
Regex matched the line successfully
Matched groups - Date: Mon Sep 30 16:02:30.942916 2024, IP: 192.168.217.1, Username: rija
Regex matched the line successfully
Matched groups - Date: Mon Sep 30 16:02:37.093444 2024, IP: 192.168.217.1, Username: lucas
Insertion: ('truc', '2024-09-30 16:02:15', '192.168.217.1', 'failed')
Insertion: ('muche', '2024-09-30 16:02:24', '192.168.217.1', 'failed')
Insertion: ('rija', '2024-09-30 16:02:30', '192.168.217.1', 'failed')
Insertion: ('lucas', '2024-09-30 16:02:37', '192.168.217.1', 'failed')
4 enregistrements ont été insérés avec succès.
```

```
MariaDB [erreur_log]> SELECT * FROM erreurs_log;
+----+-----+-----+-----+-----+
| id | username | attempt_time | ip_address | status |
+----+-----+-----+-----+-----+
| 1 | truc | 2024-09-30 16:02:15 | 192.168.217.1 | failed |
| 2 | muche | 2024-09-30 16:02:24 | 192.168.217.1 | failed |
| 3 | rija | 2024-09-30 16:02:30 | 192.168.217.1 | failed |
| 4 | lucas | 2024-09-30 16:02:37 | 192.168.217.1 | failed |

```

## 7. Envoi de Mails : JOB09

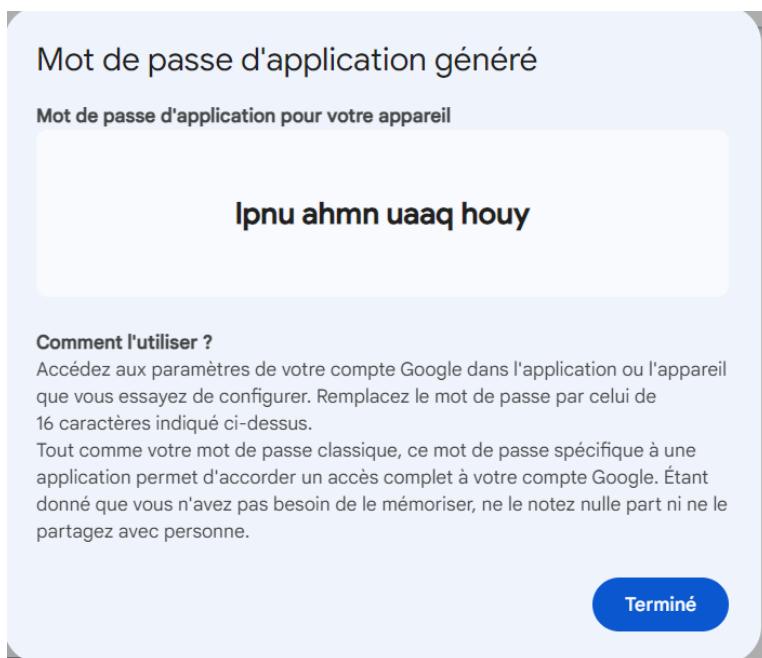
## 7.1. ssh\_serveur\_mail.py : Envoi de mails quotidiens à l'admin système avec les tentatives de connexion échouées

Nous avons besoin de 3 adresse mails, nous allons expliquer l'intérêt de chacune

**1- Adresse du serveur SMTP [smtp@gmail.com](mailto:smtp@gmail.com)** : Serveur qui transmet vos emails, il s'agit de l'adresse mail du serveur SMTP .c'est port sont Port : **587 (TLS)** ou 465 (SSL)

**2- Adresse d'authentification** : Il s'agit de l'adresse email que vous utilisez pour vous connecter au serveur SMTP. Cette adresse correspond à celle avec laquelle vous envoyez l'email.

 manon  
manon.userlrm@gmail.com



on la crée ou utilise son adresse gmail si on en a une (préférence créer )

**3- Adresse du destinataire** : Il s'agit de l'adresse de l'administrateur qui va recevoir l'email. Ce n'est pas forcément la même que celle que vous utilisez pour vous connecter ou envoyer l'email.

 monitor  
monitor.admlrm@gmail.com

son mot de passe : **root1234!**

## Mot de passe d'application généré

Mot de passe d'application pour votre appareil

**teim jdeu aqly atmz**

### Comment l'utiliser ?

Accédez aux paramètres de votre compte Google dans l'application ou l'appareil que vous essayez de configurer. Remplacez le mot de passe par celui de 16 caractères indiqué ci-dessus.

Tout comme votre mot de passe classique, ce mot de passe spécifique à une application permet d'accorder un accès complet à votre compte Google. Étant donné que vous n'avez pas besoin de le mémoriser, ne le notez nulle part ni ne le partagez avec personne.

Terminé

Maintenant il faut activer la [validation en 2 étapes](#) sur les comptes gmail afin de pouvoir générer notre mot de passe d'application qui va nous être primordial pour notre script

### Pourquoi ?

Gmail, par défaut, n'autorise pas les connexions non sécurisées. Vous devez créer un **mot de passe d'application** pour autoriser l'envoi de mails via SMTP à partir d'applications comme Python.

Pour ce faire, on se rend sur le compte google de l'adresse email que l'on vient de créer

puis on se rend dans sécurité

Google Account  Search Google Account   

Home Personal info Data & privacy **Security** People & sharing Payment

### You have security recommendations

Recommended actions found in the Security Checkup



[Protect your account](#)

### Recent security activity

New sign-in on Windows 2:42 PM · France 

[Review security activity](#)

### How you sign in to Google

Make sure you can always access your Google Account by keeping this information up to date

 [2-Step Verification](#) 2-Step Verification is off 

On va sur validation en 2 étapes, et on le configure en suivant simplement les instructions.

## You're now protected with 2-Step Verification



When signing in you'll be asked to complete the most secure second step, so make sure this info is always up to date

Ok maintenant il faut **générer le mot de passe d'application**, pour le générer il suffit de se rendre dans "moncompte" puis dans la barre de recherche dédié on tape "**mdp d'application**"

A screenshot of a search interface. At the top, there is a search bar with the text "mot". Below the search bar, the text "Résultats du compte Google" is displayed. Two search results are shown: "Gestionnaire de mots de passe Sécurité" and "Mots de passe des applications Sécurité". Both results have small icons to their left.

Comme demandé, on rentre le nom de l'application pour laquelle on veut le mdp et on appuie sur créer il le génère.

Vos mots de passe d'application

python SMTP	Date de création : 1 oct.	
-------------	---------------------------	--

Pour créer un mot de passe spécifique à une appli, indiquez son nom ci-dessous.

[Créer](#)

Mot de passe d'application généré

Mot de passe d'application pour votre appareil

**teim jdeu aqly atmz**

**Comment l'utiliser ?**  
Accédez aux paramètres de votre compte Google dans l'application ou l'appareil que vous essayez de configurer. Remplacez le mot de passe par celui de 16 caractères indiqué ci-dessus.  
Tout comme votre mot de passe classique, ce mot de passe spécifique à une application permet d'accorder un accès complet à votre compte Google. Étant donné que vous n'avez pas besoin de le mémoriser, ne le notez nulle part ni ne le partagez avec personne.

[Terminé](#)

Une fois le mdp d'application généré, on peut s'attaquer à notre script, nous avons choisi de présenter deux méthodes différentes.

```

def get_logs():
    try:
        # Connexion à la base de données MySQL
        connection = mysql.connector.connect(
            host=config.DBipaddr,
            database=config.DBname,
            user=config.username,
            password=config.DBpassword
        )

        # Requêtes SQL pour récupérer les logs de la veille
        sql_select_query_access_logs = "SELECT * FROM access_logs WHERE DATE(attempt_time) = %s;"
        sql_select_query_ftp_logs = "SELECT * FROM ftp_logs WHERE DATE(attempt_time) = %s;"

        # Calcul de la date de la veille (format YYYY-MM-DD)
        date = (datetime.now() - timedelta(1)).date()

        # Exécution des requêtes pour récupérer les logs
        cursor = connection.cursor()
        cursor.execute(sql_select_query_access_logs, (date,))
        access_logs = cursor.fetchall()

        cursor.execute(sql_select_query_ftp_logs, (date,))
        ftp_logs = cursor.fetchall()

        return access_logs, ftp_logs

    except Error as e:
        print("Erreur lors de la lecture des données de la base de données MySQL:", e)
        return [], []

    finally:
        # Fermeture de la connexion à la base de données
        if connection.is_connected():
            cursor.close()
            connection.close()

# Fonction pour envoyer un e-mail contenant les logs à l'administrateur
def send_email(access_logs, ftp_logs):
    try:
        # Configuration de l'e-mail
        sender_email = config.smtp_user
        receiver_email = config.mail_dest
        smtp_server = config.smtp_server
        smtp_port = config.smtp_port
        smtp_user = config.smtp_user
        smtp_password = config.smtp_password

        # Formatage des logs dans un texte lisible
        email_content = "Voici les logs des tentatives de connexion de la veille:\n\n"

```

```

60     if access_logs:
61         email_content += "Logs d'accès (access_logs):\n"
62         for log in access_logs:
63             email_content += f"{log}\n"
64     else:
65         email_content += "Aucun log d'accès (access_logs) trouvé.\n"
66
67     email_content += "\n"
68
69     if ftp_logs:
70         email_content += "Logs FTP (ftp_logs):\n"
71         for log in ftp_logs:
72             email_content += f"{log}\n"
73     else:
74         email_content += "Aucun log FTP (ftp_logs) trouvé.\n"
75
76     # Création de l'objet MIMEText pour l'e-mail
77     msg = MIMEText(email_content)
78     msg['Subject'] = "Rapport des Tentatives de Connexion - Veille"
79     msg['From'] = sender_email
80     msg['To'] = receiver_email
81
82     # Connexion au serveur SMTP et envoi de l'e-mail
83     with smtplib.SMTP(smtp_server, smtp_port) as server:
84         server.starttls() # Pour la sécurité
85         server.login(smtp_user, smtp_password)
86         server.sendmail(sender_email, receiver_email, msg.as_string())
87
88     print("E-mail envoyé avec succès à l'administrateur.")
89
90 except Exception as e:
91     print("Erreur lors de l'envoi de l'e-mail:", e)
92
93 # Fonction principale pour récupérer les logs et envoyer un e-mail
94 def main():
95     print("Récupération des logs de la veille...")
96     access_logs, ftp_logs = get_logs()
97
98     print("Envoi des logs par e-mail à l'administrateur...")
99     send_email(access_logs, ftp_logs)
100
101 # Appel de la fonction principale
102 if __name__ == "__main__":
103     main()
104

```

Pour ce script, nous avons décidé d'aller directement chercher les informations sur notre base de données et d'utiliser la bibliothèque config qui permet de gérer la configuration via des fichiers de configuration qui permettent de séparer les paramètres du code principal.

C'est configuration peuvent être stockés dans des fichiers .ini ou .py , voici la différence entre les 2 fichiers

- Un fichier **.ini** est un fichier de configuration externe que tu peux lire à l'aide de la bibliothèque `configparser`. Il est idéal pour les configurations simples et peut être modifié sans toucher au code Python.
- Un fichier **config.py** est un module Python contenant des variables. Il permet une configuration plus flexible et dynamique, mais est moins accessible aux non-développeurs, car il fait partie du code Python.

différence entre les 2 fichiers

Caractéristique	.ini (avec configparser)	.py (fichier Python)
Format	Texte avec sections et paires clé-valeur	Fichier Python avec des variables
Lecture/écriture	Utilise la bibliothèque <code>configparser</code>	Utilise simplement <code>import</code>
Séparation du code	Configuration séparée du code	Configuration fait partie du code
Modifiable par non-dév	Oui	Non (nécessite des connaissances en Python)
Capacités dynamiques	Non, simple lecture des valeurs	Oui, tu peux utiliser de la logique Python

Pour résumer, c'est une bonne pratique d'utiliser un fichier config séparé qui permet de rendre un code plus propre, plus flexible et plus sécurisé.

Voici le fichier config.py que nous avons configurer pour notre script ci-dessus

```
1 #-----#
2
3 #chemin vers la clé privée
4 keyfile = "/home/monitor/.ssh/id_rsa"
5 #Nom d'utilisateur
6 username = "monitor"
7
8 #-----#
9
10 #Configuration du serveur FTP
11 FTPipaddr = "192.168.72.138"
12 FTPlogfile = "/var/log/auth.log"
13
14 #-----#
15
16 #Configuration du serveur mariadb
17 DBipaddr = "192.168.72.142"
18 DBpassword = "monitor"
19 DBname = "DB_access"
20
21 #-----#
22
23 #SMTP
24 smtp_server = "smtp.gmail.com"
25 smtp_port = 587
26 smtp_user = "smtp.user.laplateforme@gmail.com"
27 smtp_password = "kcdt riwf omva xngl"
28 #mail destinataire
29 mail_dest = "rija.rasoanaivo@laplateforme.io"
```

*Voici le résultat notre admin a bien reçu un mail notifiant les erreurs de log de la veille serveur web*



smtp.user.laplateforme@gmail.com

À moi ▾

Voici les logs des tentatives de connexion de la veille:

Aucun log d'accès à la table de données (access\_logs) trouvé.

\*\*\*

Logs FTP (ftp\_logs):

```
(10, 'Rija', datetime.datetime(2024, 10, 1, 10, 53, 15), '192.168.72.1', 'failed for invalid user')
(11, 'Yousef', datetime.datetime(2024, 10, 1, 14, 26, 10), '192.168.72.1', 'failed for invalid user')
(12, 'monitor', datetime.datetime(2024, 10, 1, 14, 34, 2), '192.168.72.1', 'failed for invalid password')
(13, 'Rija', datetime.datetime(2024, 10, 1, 10, 53, 15), '192.168.72.1', 'failed for invalid user')
(14, 'Yousef', datetime.datetime(2024, 10, 1, 14, 26, 10), '192.168.72.1', 'failed for invalid user')
(15, 'monitor', datetime.datetime(2024, 10, 1, 14, 34, 2), '192.168.72.1', 'failed for invalid password')
```

Et sur celui ci il s'agit des erreurs de log du serveur ftp



**smtp.user.laplateforme@gmail.com**

À rijarasoanaivo ▾

Voici les logs des tentatives de connexion de la veille:

Logs d'accès (access\_logs):

```
(1411, 'Manu', datetime.datetime(2024, 9, 30, 11, 8, 35), 'localhost', 'failed')
(1412, 'monitor', datetime.datetime(2024, 9, 30, 11, 13, 4), 'localhost', 'failed')
(1413, 'Rija', datetime.datetime(2024, 9, 30, 11, 14, 3), 'localhost', 'failed')
(1414, 'monitor', datetime.datetime(2024, 9, 30, 11, 15, 46), 'localhost', 'failed')
```

Aucun log FTP (ftp\_logs) trouvé.

## second script envoie mail :

Pour le second script , nous avons opté pour **une configuration en dur**, car plus simple pour les débutants, elle n'a pas besoin de dépendances externes et facilite le débogage en permettant des modifications directes des valeurs dans le code.

et avons choisi de refaire une demande à notre serveur web, de le stocker dans la base de données créée à cet effet et si il se trouve dans la table dans ce cas il a récupéré et l'envoie à notre admin par mail

```

❶ ssh_serveur_mail.py > ...
...
9
10 # Configuration des connexions
11 ssh_host = "192.168.217.132"
12 ssh_user = "monitor"
13 chemin_cle = "/home/monitor/.ssh/id_rsa"
14 log_file_path = '/var/log/apache2/error.log'
15 portSsh = 22
16 portMariadb = 3306
17 db_host = "192.168.217.133"
18 db_user = "monitor"
19 db_password = getpass.getpass("Entrez le mot de passe MySQL: ")
20 db_name = "erreur_log"
21 admin_email = "monitor.admlrm@gmail.com" # Adresse de l'admin système
22
23 # Connexion SSH pour lire le fichier de log
24 ssh_client = paramiko.SSHClient()
25 ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
26 ssh_client.connect(ssh_host, username=ssh_user, key_filename=chemin_cle, port=portSsh)
27
28 # Connexion MySQL
29 db = pymysql.connect(host=db_host, user=db_user, password=db_password, database=db_name, port=portMariadb)
30
31 # Lecture des tentatives échouées depuis le journal Apache
32 stdin, stdout, stderr = ssh_client.exec_command(f"cat {log_file_path}")
33 log_content = stdout.read().decode('utf-8')
34 ssh_client.close()
35
36 # Analyser le contenu du fichier de log
37 connections = []
38 for line in log_content.splitlines():
39     if "AH01618: user" in line and "not found" in line:
40         log_match = re.match(r"\[(.*?)\] \.* \[(client ([\d\.]+):\d+)\] AH01618: user (\w+) not found:.*", line)
41         if log_match:
42             date_str, ip_address, username = log_match.groups()
43             attempt_time = datetime.strptime(date_str, "%a %b %d %H:%M:%S.%f %Y").strftime("%Y-%m-%d %H:%M:%S")
44             connections.append((username, ip_address, attempt_time, 'failed'))
45
46 # Formater les tentatives échouées pour l'email
47 if connections:
48     message_content = "Tentatives de connexion échouées :\n\n"
49     for connection in connections:
50         message_content += f"Utilisateur : {connection[0]}, IP : {connection[1]}, Date/Heure : {connection[2]}\n"
51 else:
52     message_content = "Aucune tentative de connexion échouée aujourd'hui."
53
54 # Envoi de l'email à l'administrateur système via Gmail
55 def envoyer_email(message_content):
56     msg = MIMEText(message_content)
57     msg['Subject'] = 'Rapport quotidien des tentatives de connexion échouées'
58     msg['From'] = 'manon.userlrm@gmail.com' # Votre adresse Gmail
59     msg['To'] = admin_email
60
61     try:
62         with smtplib.SMTP('smtp.gmail.com', 587) as server:
63             server.starttls() # Démarrer la connexion TLS
64             server.login('manon.userlrm@gmail.com', 'lpnu ahmn uaaq houy') # Utilisez votre mot de passe d'application ici
65             server.sendmail('manon.userlrm@gmail.com', admin_email, msg.as_string())
66             print("Email envoyé avec succès à l'administrateur.")
67     except Exception as e:
68         print(f"Erreur lors de l'envoi de l'email : {e}")
69
70 # Envoi de l'email
71 envoyer_email(message_content)
72
73 # Fermeture de la connexion à la base de données
74 db.close()

```

## 8. Sauvegarde des Bases de Données : JOB10

8.1. ssh\_cron\_backup.py : Sauvegarde locale de la base de données avec planification toutes les 3 heures

Nous allons mettre en place un mécanisme automatisé pour **sauvegarder les logs MySQL** de notre vm **serveur SQL distant** vers notre **vm lmrScript**. Le script Python sera exécuté régulièrement grâce à **cron**. Voici les étapes détaillées pour accomplir cela.

Créer le script **ssh\_cron\_backup.py**, on va ensuite se baser sur nos anciens scripts pour l'utilisation de nos paramètres principaux.

```
#!/usr/bin/python3
import re
import getpass
import pymysql
import paramiko
from datetime import datetime
import os

# Demande des informations pour la connexion SSH
print("Connexion SSH au serveur SQL")
ssh_username = "monitor"
key_file = "/home/monitor/.ssh/id_rsa"
ipaddr = "192.168.52.131" # Adresse IP de la VM lmrSQL

# Connexion SSH au serveur SQL
print(f"Connexion SSH à {ipaddr} avec l'utilisateur {ssh_username}...")
try:
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(ipaddr, username=ssh_username, key_filename=key_file)
    print("Connexion SSH réussie.")
except Exception as e:
    print(f"Erreur de connexion SSH : {e}")
    exit(1)

# Demande des informations pour la connexion à la base de données MariaDB
print("\nConnexion à la base de données MariaDB")
DBipaddr = "192.168.52.131" # L'adresse IP de votre serveur SQL
DBusername = "monitor"
DBpassword = "monitorsql"
DBname = "DB_access" # Nom de la base de données

# Connexion à la base de données MariaDB
print(f"Connexion à la base de données MariaDB sur {DBipaddr} avec l'utilisateur {DBusername}...")
try:
    db_connect = pymysql.connect(host=DBipaddr, user=DBusername, password=DBpassword, database=DBname)
    print("Connexion à la base de données réussie.")
except pymysql.MySQLError as e:
    print(f"Erreur lors de la connexion à la base de données : {e}")
    ssh_client.close()
    exit(1)
```

Ces paramètres sont :

- La demande des informations nécessaire lors de la connexion SSH.

- La connexion SSH avec l'utilisateur "monitor" de la vm lrmScript vers le serveur SQL de la vm lmrSQL. Établit la connexion SSH et vérifie que tous les paramètres sont ok.
- La demande des informations nécessaire lors de la connexion à la base de donnée DB\_access MariaDB
- La connexion à la base de données DB\_access avec l'utilisateur créé dans notre serveur sql "monitor" et son mot de passe. Établit la connexion à la base de données et vérifie que tous les paramètres sont ok.

```
# Lecture du fichier de log sur le serveur SQL
log_file_path = '/var/log/mysql/mysql.log' # Chemin vers le fichier de log sur le serveur SQL
print(f'Lecture du fichier de log {log_file_path} sur le serveur SQL...')

# Récupère le contenu du fichier de log
stdin, stdout, stderr = ssh_client.exec_command(f"cat {log_file_path}")
log_content = stdout.read().decode('utf-8')
```

**log\_file\_path** = '/var/log/mysql/mysql.log'

Lis le fichier des erreurs de log sur le serveur SQL, si tout est ok ça nous print que le bon fonctionnement de la lecture du fichier de log.

```
stdin, stdout, stderr = ssh_client.exec_command(f"cat {log_file_path}")
```

Cette ligne exécute la commande **à distance** via une connexion SSH au serveur SQL. La commande **cat {log\_file\_path}** est utilisée pour lire le contenu du fichier de log dont le chemin est contenu dans **log\_file\_path**.

**ssh\_client.exec\_command** exécute une commande sur le serveur distant via SSH. Dans ce cas, il s'agit de la commande cat, qui lit le contenu du fichier log.

**Les paramètres (stdin, stdout, stderr) :**

- **stdin** : C'est le flux d'entrée de la commande, mais dans ce cas, il n'est pas utilisé, car la commande **cat** ne demande aucune entrée supplémentaire.
- **stdout** : C'est le flux de sortie standard, qui contient la **sortie normale** de la commande cat, c'est-à-dire le contenu du fichier de log.
- **stderr** : C'est le flux de sortie d'erreur, qui contiendrait les messages d'erreur si la commande échouait (par exemple, si le fichier n'existe pas).

```
log_content = stdout.read().decode('utf-8')
```

`stdout.read()` lit tout le contenu de la sortie de la commande cat, qui correspond au contenu du fichier de log.

`.decode('utf-8')` est utilisé pour convertir le contenu en chaîne de caractères en format **UTF-8** (un encodage standard).

Création du répertoire de sauvegarde et crée un fichier :

```
51 # Crée un dossier de sauvegarde s'il n'existe pas
52 backup_dir = '/var/log/mysql/backup_db'
53 os.makedirs(backup_dir, exist_ok=True)
54
55 # Crée un nom de fichier de sauvegarde avec l'horodatage
56 timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
57 backup_file = os.path.join(backup_dir, f"mysql_log_backup_{timestamp}.log")
```

`os.makedirs()` est une fonction de la bibliothèque standard os en Python. Elle est utilisée pour créer un ou plusieurs répertoires.

Le paramètre `backup_dir` correspond au chemin du répertoire qu'on veut créer (défini précédemment).

Le paramètre `exist_ok=True` signifie que si le répertoire existe déjà, il ne renverra pas d'erreur.

On peut aussi créer le fichier en amont et vérifier les privilèges du répertoire au cas où il y aurait une erreur lors du lancement du script.

```
monitor@lrmScript:~$ ls -la /var/log/mysql/
total 12
drwxr-xr-x 3 root      root      4096  1 oct.  16:40 .
drwxr-xr-x 8 root      root      4096  2 oct.  15:34 ..
drwxr-xr-x 2 monitor   monitor   4096  2 oct.  15:45 backup_db
```

Ensuite, on crée l'horodatage et l'affichage de nos futurs fichiers sauvegardés.

`timestamp = datetime.now().strftime('%Y%m%d_%H%M%S') :`

la fonction `datetime.now()` du module datetime obtient la date et l'heure actuelles.

`strftime('%Y%m%d_%H%M%S')` formate cette date en une chaîne sous la forme YYYYMMDD\_HHMMSS :

- `%Y` : Année sur 4 chiffres (par exemple, 2024).

- %m : Mois sur 2 chiffres (01 à 12).
- %d : Jour sur 2 chiffres (01 à 31).
- %H : Heure sur 24 heures (00 à 23).
- %M : Minute sur 2 chiffres (00 à 59)
- %S : Seconde sur 2 chiffres (00 à 59).

Le résultat est donc une chaîne comme **20241002\_153015** qui est un horodatage unique au moment de l'exécution du script.

On va ouvrir un fichier de sauvegarde en mode écriture et y stocker le contenu des logs

```

59  # Écrire le contenu des logs dans le fichier de sauvegarde
60  with open(backup_file, 'w') as f:
61      f.write(log_content)
62  print(f"Sauvegarde des logs créée : {backup_file}")
63
64  # Conservation des sept dernières sauvegardes
65  print("Conservation des sept dernières sauvegardes...")
66  backups = sorted(os.listdir(backup_dir))
67  if len(backups) > 7:
68      for backup in backups[:-7]: # Supprime les sauvegardes les plus anciennes
69          os.remove(os.path.join(backup_dir, backup))
70      print(f"Sauvegarde supprimée : {backup}")
71
72  # Affiche la liste des fichiers dans le dossier de sauvegarde
73  print("\nListe des fichiers de sauvegarde dans le dossier 'backup_db' :")
74  for file in sorted(os.listdir(backup_dir)):
75      print(file)

```

`open(backup_file, 'w')` : ouvre (ou crée si inexistant) un fichier dont le chemin est défini par la variable **backup\_file** et le mode '`w`' signifie que le fichier est ouvert en mode écriture. Si, il existe déjà, son contenu sera remplacé.

`f.write(log_content)` : écrit le contenu des logs (`log_content`) dans le fichier ouvert.

`print(f"Sauvegarde des logs créée : {backup_file}")` : cela va afficher un message indiquant que la sauvegarde des logs a été réalisée et précise le chemin du fichier de sauvegarde.

Pour finir, on conserve les sept dernières sauvegardes et supprimer les plus anciennes.

`backups = sorted(os.listdir(backup_dir))` : va récupérer la liste des fichiers présents dans le répertoire de sauvegarde **backup\_dir**, puis les trie dans l'ordre croissant.

**if len(backups) > 7** : Vérifie si le nombre de fichiers dans le répertoire de sauvegarde dépasse 7.

**backups[:-7]** : sélectionne tous les fichiers à supprimer, sauf les sept plus récents.

**[:-7]** va prendre tous les éléments sauf les 7 derniers dans la liste triée.

On lance notre script :

```
monitor@lrmScript:/$ sudo python3 ssh_cron_backup.py
[sudo] Mot de passe de monitor :
Connexion SSH au serveur SQL
Connexion SSH à 192.168.52.131 avec l'utilisateur monitor...
Connexion SSH réussie.

Connexion à la base de données MariaDB
Connexion à la base de données MariaDB sur 192.168.52.131 avec l'utilisateur
monitor...
Connexion à la base de données réussie.
Lecture du fichier de log /var/log/mysql/mysql.log sur le serveur distant...
Sauvegarde des logs créée : /var/log/mysql/backup_db/mysql_log_backup_202410
03_152814.log
Conservation des sept dernières sauvegardes...
Sauvegarde supprimée : mysql_log_backup_20241003_152204.log

Liste des fichiers de sauvegarde dans le dossier 'backup_db' :
mysql_log_backup_20241003_152304.log
mysql_log_backup_20241003_152403.log
mysql_log_backup_20241003_152504.log
mysql_log_backup_20241003_152605.log
mysql_log_backup_20241003_152704.log
mysql_log_backup_20241003_152805.log
mysql_log_backup_20241003_152814.log
Fermeture de la connexion SSH...
```

On paramétrer Cron, pour cela on tape la commande : **crontab -e**

```
GNU nano 7.2
0 */3 * * * /usr/bin/python3 /ssh_cron_backup.py
# Edit this file to introduce tasks to be run by cron.
#
```

**0 \*/3 \* \* \*** : Cela signifie que le script sera exécuté à la minute 0 toutes les trois heures.

**/usr/bin/python3** : L'interpréteur Python utilisé pour exécuter le script.

**/ssh\_cron\_backup.py** : Le chemin vers notre script Python ssh\_cron\_backup.py.

On vérifie l'endroit où l'on a stocké nos logs avec la commande “**ls**” pour vérifier le bon fonctionnement du fichier crontab qui exécute notre script .

Dans cet exemple, le script s'exécute toutes les minutes. Mais à modifier pour l'exécuter toutes les 3 heures.

```
monitor@lrmScript:$ sudo ls /var/log/mysql/backup_db/
mysql_log_backup_20241003_152605.log  mysql_log_backup_20241003_152904.log
mysql_log_backup_20241003_152704.log  mysql_log_backup_20241003_153003.log
mysql_log_backup_20241003_152805.log  mysql_log_backup_20241003_153103.log
mysql_log_backup_20241003_152814.log
monitor@lrmScript:$ sudo ls /var/log/mysql/backup_db/
mysql_log_backup_20241003_152605.log  mysql_log_backup_20241003_152904.log
mysql_log_backup_20241003_152704.log  mysql_log_backup_20241003_153003.log
mysql_log_backup_20241003_152805.log  mysql_log_backup_20241003_153103.log
mysql_log_backup_20241003_152814.log
monitor@lrmScript:$ sudo ls /var/log/mysql/backup_db/
[sudo] Mot de passe de monitor :
mysql_log_backup_20241003_154104.log  mysql_log_backup_20241003_154503.log
mysql_log_backup_20241003_154203.log  mysql_log_backup_20241003_154604.log
mysql_log_backup_20241003_154303.log  mysql_log_backup_20241003_154703.log
mysql_log_backup_20241003_154404.log
```

## 9. Surveillance des Ressources Systèmes : JOB11 & 12 & 13

### 9.1. ssh\_system\_status.py : Récupération et stockage de l'état des ressources systèmes (RAM/CPU/DISK)

Pour ce script, nous devons récupérer l'état des ressources systèmes (RAM,CPU et DISK) sur nos 2 serveurs mariadb et ftp, puis les stocker. Pour se faire voici le script que nous avons fait

```

6  # Fonction qui se connecte en SSH à chaque serveur et récupère l'état des ressources système
7  def get_system_status():
8      try:
9          # Connexion à la base de données MySQL
10         connection = pymysql.connect(
11             host=config.DBipaddr,
12             database=config.DBname,
13             user=config.username,
14             password=config.DBpassword
15         )
16
17         # Requête SQL pour insérer les données dans la table system_status
18         sql_insert_query = "INSERT INTO system_status (server_name, cpu_usage, ram_usage, disk_usage, log_time) VALUES (%s, %s, %s, %s, %s);"
19
20         # Liste des serveurs à monitoren
21         servers = [
22             {"name": "Serveur FTP", "ip": config.FTPipaddr},
23             {"name": "Serveur MariaDB", "ip": config.DBipaddr}
24         ]
25
26         # Connexion en SSH à chaque serveur
27         for server in servers:
28             # Connexion SSH
29             ssh_client = paramiko.SSHClient()
30             ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
31             ssh_client.connect(server["ip"], username=config.username, key_filename=config.keyfile)
32
33             # Commande pour récupérer l'utilisation du CPU
34             stdin, stdout, stderr = ssh_client.exec_command("top -bn1 | grep 'Cpu(s)' | awk '{print 100 - $8}'")
35             cpu_usage_str = stdout.read().decode().strip().replace(',', '.')
36             cpu_usage = float(cpu_usage_str)
37
38             # Commande pour récupérer l'utilisation de la RAM
39             stdin, stdout, stderr = ssh_client.exec_command("free | grep Mem | awk '{print $3/$2 * 100.0}'")
40             ram_usage_str = stdout.read().decode().strip().replace(',', '.')
41             ram_usage = float(ram_usage_str)
42
43             # Commande pour récupérer l'utilisation du disque
44             stdin, stdout, stderr = ssh_client.exec_command("df -h | awk '$NF==\"/\"{print $5}'")
45             disk_usage_str = stdout.read().decode().strip().replace('%', '').replace(',', '.')
46             disk_usage = float(disk_usage_str)
47
48             # Enregistrement du moment de la mesure
49             log_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
50
51             # Insertion des données dans la base de données
52             cursor = connection.cursor()
53             cursor.execute(sql_insert_query, (server["name"], cpu_usage, ram_usage, disk_usage, log_time))
54             connection.commit()
55             cursor.close()
56
57             # Fermeture de la connexion SSH
58             ssh_client.close()
59
60         # Suppression des données de plus de 72 heures
61         delete_query = "DELETE FROM system_status WHERE log_time < %s"
62         threshold_time = (datetime.now() - timedelta(hours=72)).strftime('%Y-%m-%d %H:%M:%S')
63         cursor = connection.cursor()
64         cursor.execute(delete_query, (threshold_time,))
65         connection.commit()
66         cursor.close()
67
68     except Exception as e:
69         print("Erreur lors de la récupération des données système:", e)
70
71     finally:
72         # Fermeture de la connexion à la base de données
73         if connection.open:
74             connection.close()
75
76     # Appel de la fonction pour récupérer l'état des ressources système
77     get_system_status()

```

### Explication commande pour récupérer informations du systèmes dans le script:

On utilise la bibliothèque paramiko pour exécuter nos commandes sur notre serveur distant via SSH pour récupérer les informations de notre système

Puis les commandes suivante :

commande pour récupérer CPU

```
stdin, stdout, stderr = ssh_client.exec_command("top -bn1 | grep 'Cpu(s)' | awk '{print 100 - $8}'")
```

**top -bn1** :

- **top** : Affiche les statistiques en temps réel sur l'utilisation du processeur et des processus système.
- **-b** : Active le **mode batch**, qui permet à **top** de s'exécuter sans interface interactive, en exportant les données dans un format texte brut pour être utilisées dans des scripts.
- **-n1** : Indique à **top** de s'exécuter une seule fois, plutôt que de rafraîchir les données en continu.

**grep 'Cpu(s)'**:

- Filtre la sortie de la commande précédente pour ne garder que les informations sur l'utilisation **CPU**.

**awk '{print 100 - \$8}'** :

- **awk** : est un programme qui permet de traiter et manipuler des données textuelles. Ici, il extrait une partie
- **\$8** : Représente le 8ème champ de la ligne, qui correspond au pourcentage de CPU **inactif** (**idle**).
- **100 - \$8** : Calcule l'utilisation réelle du CPU en soustrayant le pourcentage de CPU inactif à 100. Par exemple, si le CPU est 80% inactif, alors il est utilisé à 20%

Commande pour récupérer la RAM:

```
stdin, stdout, stderr = ssh_client.exec_command("free | grep Mem | awk '{print $3/$2 * 100.0}'")
```

- **free** : Cette commande permet d'obtenir le pourcentage d'utilisation de la RAM
- **grep Mem** : Filtre pour ne garder que la ligne contenant des informations sur la RAM
- **awk '{print \$3/\$2 \* 100.0}'** : Calcule le pourcentage de mémoire utilisée  
\$3 et \$2 correspondent aux colonnes de la ligne.

```
monitor@lrmServeurFtp:~$ free | grep "Mem"
Mem:      1978776      350784    1597960       756      179744    1627992
```

Résultat de la table de données “system\_status”

MariaDB [DB_access]> SELECT * FROM system_status;
+-----+-----+-----+-----+-----+
id   server_name   cpu_usage   ram_usage   disk_usage   log_time
1   Serveur FTP   46.2   19.0781   29   2024-10-01 18:32:19
2   Serveur MariaDB   57   28.3456   33   2024-10-01 18:32:21
3   Serveur FTP   50   17.7099   29   2024-10-02 10:50:13
4   Serveur MariaDB   53   28.2552   33   2024-10-02 10:50:15

4 rows in set (0,003 sec)

## 9.2. ssh\_system\_mail.py : Envoi d'alertes si dépassement des seuils d'utilisation des ressources

Ce script va nous permettre d'envoyer des alertes en cas de dépassement des seuils d'utilisation des ressources ( CPU, RAM, Disque)

Pour ce faire, nous avons repris l'ancien et ajouté les paramètres des seuils d'alerte de nos ressources que nous pouvons modifier à tout moment et le crontab pour le planifier toutes les 5 mn

**cpu\_threshold = 10** : Ce seuil indique que si l'utilisation du CPU dépasse 10%, une alerte sera envoyée. Ce seuil est assez bas pour le CPU, et cela pourrait entraîner des notifications fréquentes car même des tâches courantes peuvent consommer plus de 10% de CPU. Un seuil plus courant pour le CPU pourrait être autour de 70-80%.

**ram\_threshold = 80** : Ici, si l'utilisation de la RAM dépasse 80%, une alerte sera déclenchée. Ce seuil est raisonnable car lorsqu'un système utilise plus de 80% de sa mémoire, il peut commencer à ralentir ou à échanger de la mémoire sur le disque, ce qui impacte les performances.

**disk\_threshold = 90** : Ce seuil indique qu'une alerte sera envoyée si l'utilisation du disque dépasse 90%. Un disque trop rempli peut causer des problèmes de performances et empêcher certaines applications de fonctionner correctement. Il est souvent recommandé d'avoir un peu d'espace libre pour éviter des problèmes inattendus.

```

8  # Seuils de notification
9  CPU_THRESHOLD = 10.0
10 RAM_THRESHOLD = 10.0
11 DISK_THRESHOLD = 10.0
12
13 # Fonction qui envoie un e-mail d'alerte
14 def send_alert_email(server_name, cpu_usage, ram_usage, disk_usage):
15     subject = f"Alerte Ressources Système - {server_name}"
16     body = f"""
17         Attention : Le serveur '{server_name}' a dépassé les seuils définis.
18
19         Détails de l'état des ressources :
20         - Utilisation CPU : {cpu_usage} % (Seuil : {CPU_THRESHOLD} %)
21         - Utilisation RAM : {ram_usage} % (Seuil : {RAM_THRESHOLD} %)
22         - Utilisation Disque : {disk_usage} % (Seuil : {DISK_THRESHOLD} %)
23
24         Veuillez vérifier immédiatement.
25     """
26
27     msg = MIMEText(body)
28     msg['Subject'] = subject
29     msg['From'] = config.smtp_user
30     msg['To'] = config.mail_dest
31
32     try:
33         server = smtplib.SMTP(config.smtp_server, config.smtp_port)
34         server.starttls()
35         server.login(config.smtp_user, config.smtp_password)
36         server.sendmail(config.smtp_user, config.mail_dest, msg.as_string())
37         server.quit()
38         print(f"Email d'alerte envoyé à {config.mail_dest} concernant le serveur {server_name}.")
39     except Exception as e:
40         print(f"Erreur lors de l'envoi de l'e-mail : {e}")
41
42 # Fonction qui se connecte en SSH à chaque serveur et récupère l'état des ressources système
43 def get_system_status():
44     try:
45         # Connexion à la base de données MySQL
46         connection = pymysql.connect(
47             host=config.DBipaddr,
48             database=config.DBname,
49             user=config.username,
50             password=config.DBpassword
51         )
52
53         # Requête SQL pour insérer les données dans la table system_status
54         sql_insert_query = "INSERT INTO system_status (server_name, cpu_usage, ram_usage, disk_usage, log_time) VALUES (%s, %s, %s, %s, %s);"
55
56         # Liste des serveurs à monter
57         servers = [
58             {"name": "Serveur FTP", "ip": config.FTPipaddr},
59             {"name": "Serveur MariaDB", "ip": config.DBipaddr}
60         ]

```

```

61     # Connexion en SSH à chaque serveur
62     for server in servers:
63         # Connexion SSH
64         ssh_client = paramiko.SSHClient()
65         ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
66         ssh_client.connect(server["ip"], username=config.username, key_filename=config.keyfile)
67
68         # Commande pour récupérer l'utilisation du CPU
69         stdin, stdout, stderr = ssh_client.exec_command("top -bn1 | grep 'Cpu(s)' | awk '{print 100 - $8}'")
70         cpu_usage_str = stdout.read().decode().strip().replace(',', '.')
71         cpu_usage = float(cpu_usage_str)
72
73         # Commande pour récupérer l'utilisation de la RAM
74         stdin, stdout, stderr = ssh_client.exec_command("free | grep Mem | awk '{print $3/$2 * 100.0}'")
75         ram_usage_str = stdout.read().decode().strip().replace(',', '.')
76         ram_usage = float(ram_usage_str)
77
78         # Commande pour récupérer l'utilisation du disque
79         stdin, stdout, stderr = ssh_client.exec_command("df -h | awk '$NF==\"/\"{print $5}'")
80         disk_usage_str = stdout.read().decode().strip().replace('%', '').replace(',', '.')
81         disk_usage = float(disk_usage_str)
82
83         # Enregistrement du moment de la mesure
84         log_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
85
86         # Insertion des données dans la base de données
87         cursor = connection.cursor()
88         cursor.execute(sql_insert_query, (server["name"], cpu_usage, ram_usage, disk_usage, log_time))
89         connection.commit()
90         cursor.close()
91
92         # Vérification des seuils pour alerter par e-mail
93         if cpu_usage > CPU_THRESHOLD or ram_usage > RAM_THRESHOLD or disk_usage > DISK_THRESHOLD:
94             send_alert_email(server["name"], cpu_usage, ram_usage, disk_usage)
95
96         # Fermeture de la connexion SSH
97         ssh_client.close()
98
99     # Suppression des données de plus de 72 heures
100    delete_query = "DELETE FROM system_status WHERE log_time < %s"
101    threshold_time = (datetime.now() - timedelta(hours=72)).strftime('%Y-%m-%d %H:%M:%S')
102    cursor = connection.cursor()
103    cursor.execute(delete_query, (threshold_time,))
104    connection.commit()
105    cursor.close()
106
107 except Exception as e:
108     print("Erreur lors de la récupération des données système:", e)
109
110 finally:
111     # Fermeture de la connexion à la base de données
112     if connection.open:
113         connection.close()
114
115 # Appel de la fonction pour récupérer l'état des ressources système
116 get_system_status()

```

voici le résultat à l'exécution du script

```
(myenv) monitor@lrmScript:~$ /home/monitor/myenv/bin/python /home/monitor/ssh_system_mail.py
Email d'alerte envoyé à rija.rasoanaivo@laplateforme.io concernant le serveur Serveur FTP.
Email d'alerte envoyé à rija.rasoanaivo@laplateforme.io concernant le serveur Serveur MariaDB.
```

voici le mail reçu à l'admin

 [smtp.user.laplateforme@gmail.com](mailto:smtp.user.laplateforme@gmail.com)  
À moi ▾

Attention : Le serveur 'Serveur MariaDB' a dépassé les seuils définis.

Détails de l'état des ressources :

- Utilisation CPU : 100.0 % (Seuil : 10.0 %)
- Utilisation RAM : 29.0076 % (Seuil : 10.0 %)

Pour l'automatiser, on ajoute une ligne dans crontab. Ci dessous pour accéder à crontab

**monitor@lrmScript:~\$ crontab -e**

Ci-dessous la commande enregistré dans crontab, cette ligne lance le script toutes les 5 minutes et enregistre les logs dans un fichier "cron\_log.txt"

```
# Active le script "ssh_system_mail.py"  
*/5 * * * * /bin/bash -c "/home/monitor/myenv/bin/python /home/monitor/ssh_system_mail.py >> /home/monitor/cron_log.txt 2>&1"
```

Cron\_log.txt :

```
monitor@lrmScript:~$ cat cron_log.txt  
Email d'alerte envoyé à rija.rasoanaivo@laplateforme.io concernant le serveur Serveur FTP.  
Email d'alerte envoyé à rija.rasoanaivo@laplateforme.io concernant le serveur Serveur MariaDB.
```

### 9.3. Ne pas envoyer plus d'un mail par heure à l'administrateur

Pour cette partie de l'exercice, nous avons repris le script précédent, et avons mis en place un mécanisme de journalisation qui enregistre l'heure du dernier envoi dans un fichier qui s'appelle "**last\_mail\_time.txt**". Si le dernier envoi a été effectué il y a moins d'une heure, le script n'enverra pas de nouveau mail. Cela permet d'éviter de spammer l'administrateur tout en gardant une vérification toutes les 5 minutes.

Ci-dessous les variables contenant le chemin du dossier et le nom du fichier "**last\_mail\_time.txt**".

```
# Dossier et fichier pour enregistrer l'heure du dernier envoi d'e-mail  
LAST_EMAIL_TIME_DIR = "/home/monitor"  
LAST_EMAIL_TIME_FILE = os.path.join(LAST_EMAIL_TIME_DIR, "last_mail_time.txt")
```

Ci-dessous les fonctions qui permettent de vérifier si un e-mail peut être envoyé, pour cela il va checker l'heure dans le fichier "**last\_mail\_time.txt**" et le comparer à l'heure du système. S'il le fichier à la même heure que le système alors le mail ne s'envoie pas, si il n'a pas la même heure que le système alors il va envoyer un mail puis mettre à jour le fichier à l'heure du système.

Ci-dessous le fichier **last\_mail\_time.txt** et les fonctions.

2024-10-02 15:49:15

```
# Fonction pour vérifier si un e-mail peut être envoyé (pas plus d'un par heure)
def can_send_email():
    try:
        if os.path.exists(LAST_EMAIL_TIME_FILE):
            with open(LAST_EMAIL_TIME_FILE, "r") as f:
                last_mail_time_str = f.read().strip()
                last_mail_time = datetime.strptime(last_mail_time_str, "%Y-%m-%d %H:%M:%S")
                if datetime.now() - last_mail_time < timedelta(hours=1):
                    return False
    # Si le fichier n'existe pas, on peut envoyer le mail
    return True
except Exception as e:
    print(f"Erreur lors de la vérification de l'heure du dernier e-mail : {e}")
    return False

# Fonction pour mettre à jour l'heure du dernier envoi de mail
def update_last_mail_time():
    try:
        with open(LAST_EMAIL_TIME_FILE, "w") as f:
            f.write(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    except Exception as e:
        print(f"Erreur lors de la mise à jour de l'heure du dernier e-mail : {e}")
```

## 10. Gestion des Mises à Jour des Serveurs : JOB14

### 10.1. ssh\_update.py : vérifier et appliquer les mises à jour des serveurs

Ce script doit nous permettre de vérifier et appliquer les mises à jour de nos serveurs,

Voici le script que nous avons fait, il permet d'envoyer un mail à l'admin après chaque application des mise a jour et vérifier si un redémarrage du système est nécessaire ou non. i

```
#!/usr/bin/env python3

import paramiko
import smtplib
from email.mime.text import MIMEText

# Configuration des connexions SSH et MySQL
servers = [
    {"name": "Web", "ip": "192.168.217.132"},
    {"name": "SQL", "ip": "192.168.217.133"},
    {"name": "FTP", "ip": "192.168.217.131"}
]
ssh_user = "monitor"
chemin_cle = "/home/monitor/.ssh/id_rsa"
```

```

portSsh = 22

admin_email = "monitor.admlrm@gmail.com"

# Envoi de l'email à l'administrateur système via Gmail

def envoyer_email(subject, message_content):

    print(f"Préparation de l'envoi de l'email : {subject}")

    msg = MIMEText(message_content)

    msg['Subject'] = subject

    msg['From'] = 'manon.userlrm@gmail.com' # Votre adresse Gmail

    msg['To'] = admin_email


try:

    with smtplib.SMTP('smtp.gmail.com', 587) as server:

        server.starttls() # Démarre la connexion TLS

        server.login('manon.userlrm@gmail.com', 'lpnu ahmn uaaq houy') # mot de passe d'application

        server.sendmail('manon.userlrm@gmail.com', admin_email,
msg.as_string())

        print("Email envoyé avec succès à l'administrateur.")

except Exception as e:

    print(f"Erreur lors de l'envoi de l'email : {e}")


# Vérification des mises à jour et redémarrage nécessaire

def verifie_et_applique_MAJ(server):

    print(f"Connexion au serveur {server['name']} ({server['ip']})...")

    ssh_client = paramiko.SSHClient()

    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:

        # Connexion au serveur

```

```
    ssh_client.connect(server['ip'], username=ssh_user,
key_filename=chemin_cle, port=portSsh)

    print(f"Connexion établie avec succès à {server['name']} ({server['ip']})")

# Vérifier et appliquer les mises à jour

print(f"Vérification et application des mises à jour sur {server['name']}...")

stdin, stdout, stderr = ssh_client.exec_command('sudo apt-get update &&
sudo apt-get upgrade -y')

stdout.channel.recv_exit_status() # Attendre la fin de l'exécution de la commande

print(f"Mises à jour effectuées sur {server['name']}")

# Vérifier si un redémarrage est nécessaire

print(f"Vérification si un redémarrage est requis sur {server['name']}...")

stdin, stdout, stderr = ssh_client.exec_command('sudo test -f /var/run/reboot-required && echo "yes" || echo "no"')

reboot_required = stdout.read().strip().decode('utf-8')

if reboot_required == "yes":

    print(f"Redémarrage requis sur {server['name']}, envoi de l'email...")

    subject = f"Redémarrage requis sur le serveur {server['name']} ({server['ip']})"

    message_content = f"Un redémarrage est requis après la mise à jour sur le serveur {server['name']} ({server['ip']})."

    envoyer_email(subject, message_content)

else:

    envoyer_email(f"Aucun redémarrage requis sur {server['name']}",
f"Aucun redémarrage requis sur {server['name']}.")

    print(f"Aucun redémarrage requis sur {server['name']}.")

except Exception as e:

    print(f"Erreur lors de la connexion ou de la mise à jour de {server['name']} : {e}")
```

```
finally:

    ssh_client.close()

    print(f"Connexion fermée pour {server['name']}.")

# Exécuter les mises à jour pour chaque serveur

if __name__ == "__main__":
    print("Démarrage du processus de mise à jour des serveurs...")

    for server in servers:
        verifie_et_applique_MAJ(server)

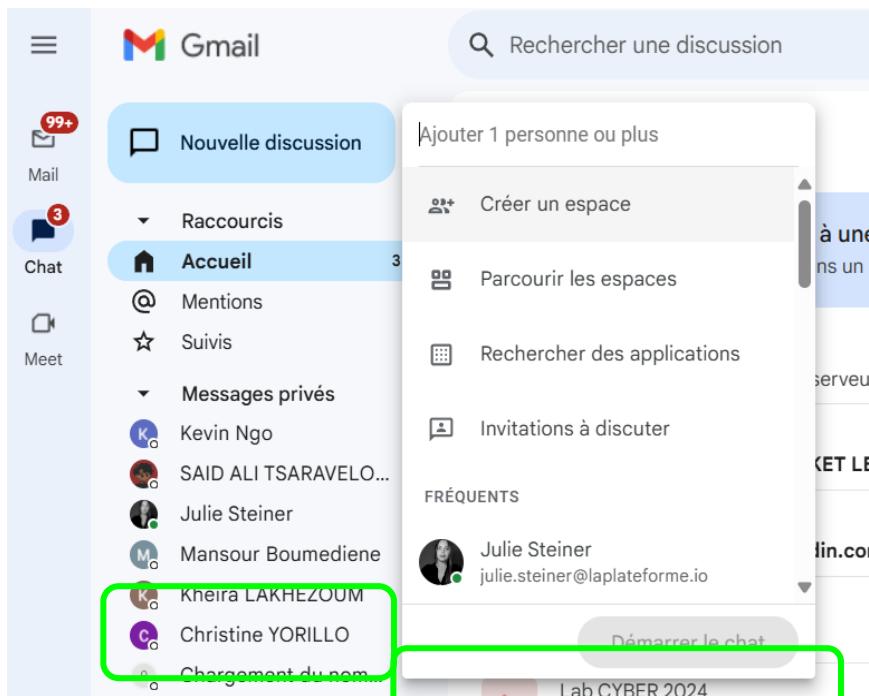
    print("Mise à jour des serveurs terminée.")
```

```
(mon_environnement) root@lrmScript:~# /root/mon_environnement/bin/python /root/ssh_update.py
Démarrage du processus de mise à jour des serveurs...
Connexion au serveur Web (192.168.217.132)...
Connexion établie avec succès à Web (192.168.217.132)
Vérification et application des mises à jour sur Web...
Mises à jour effectuées sur Web
Vérification si un redémarrage est requis sur Web...
Préparation de l'envoi de l'email : Aucun redémarrage requis sur Web
Email envoyé avec succès à l'administrateur.
Aucun redémarrage requis sur Web.
Connexion fermée pour Web.
Connexion au serveur SQL (192.168.217.133)...
Connexion établie avec succès à SQL (192.168.217.133)
Vérification et application des mises à jour sur SQL...
Mises à jour effectuées sur SQL
Vérification si un redémarrage est requis sur SQL...
Préparation de l'envoi de l'email : Aucun redémarrage requis sur SQL
Email envoyé avec succès à l'administrateur.
Aucun redémarrage requis sur SQL.
Connexion fermée pour SQL.
Connexion au serveur FTP (192.168.217.131)...
Connexion établie avec succès à FTP (192.168.217.131)
Vérification et application des mises à jour sur FTP...
Mises à jour effectuées sur FTP
Vérification si un redémarrage est requis sur FTP...
Préparation de l'envoi de l'email : Aucun redémarrage requis sur FTP
Email envoyé avec succès à l'administrateur.
Aucun redémarrage requis sur FTP.
Connexion fermée pour FTP.
Mise à jour des serveurs terminée.
```

## 11. Automatisation via Google Chat : JOB15

11.1. Création d'un espace Google Chat pour l'équipe et envoi de rapports périodiques via un script

Pour les besoins du job, nous avons créé un espace dans google chat que l'on a nommé "**JOB15**".



Puis on définit le nom de notre espace, on le laisse en espace de “Collaboration” et on clique sur “Créer”

### Créer un espace

Nom de l'espace  
Lab CYBER 2024

Quel est le but de cet espace ?  
Optimisez votre espace avec des paramètres et des suggestions d'applications utiles. [En savoir plus](#)

Collaboration  
Collaborez sur des projets, des plans ou des thèmes.  
Partagez des fichiers, attribuez des tâches et organisez vos conversations par fil de discussion, en toute simplicité.

Annonces  
Diffusez et partagez des informations avec votre groupe.

#### Paramètres d'accès

**Privé**  
Seuls les utilisateurs et les groupes ajoutés peuvent participer

Autoriser les membres externes à rejoindre l'espace  
Ce paramètre ne peut plus être modifié une fois l'espace créé

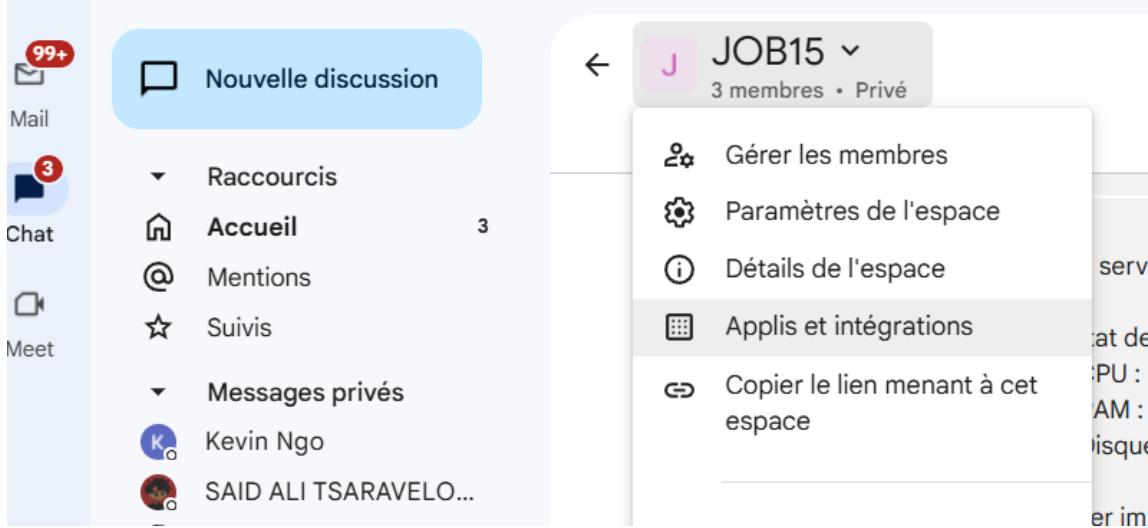
**Annuler** **Créer**

Pour que nous puissions envoyer des notifications dans l'espace google "JOB15", nous allons utiliser un "**WEBHOOK**". Un webhook dans Google Chat est un moyen simple d'envoyer des messages automatiques à une conversation ou un espace dans Google Chat depuis un autre service ou application.

Créer un webhook:

source: <https://developers.google.com/workspace/chat/quickstart/webhooks?hl=fr>

Cliquez sur la flèche située à droite du nom de votre espace, puis sur "**Applis et intégrations**".



Cette interface va s'afficher, cliquez sur "**Ajouter des webhooks**" situé en bas à droite.

A screenshot of the 'Webhooks' section in Google Workspace. On the left, a sidebar shows 'Applications' (0 appli) and a list with 'Gérer les membres', 'Paramètres de l'e...', 'Détails de l'espace', 'Applis et intégrati...', 'Applications' (selected), and 'Webhooks'. The main area is titled 'Webhooks' with '2 webhooks'. It features a central graphic of a speech bubble with arrows pointing to it from four colored circles (green, yellow, blue, red). Below the graphic is the text 'Trouvez des outils pour améliorer vos workflows'. In the bottom right corner, there's a button labeled '+ Ajouter des webhooks' which is highlighted with a green rectangular border.

Cette fenêtre apparaît, vous allez définir le nom de votre webhook.

La ligne située juste dessous est facultative, elle permet de définir un avatar via un url.

**Toutefois**, si vous tenez à ce que votre webhook ai un jolie petit avatar, vous pouvez en récupérer sur ce lien que je recommande:

[Gravatar - Avatar URL Generator \(vinicius73.github.io\)](https://gravatar.com/avatar/).

Puis cliquez sur "Enregistrer".

Webhooks entrants X

Consulter la documentation

Webhook JOB15

`https://gravatar.com/avatar/2b09f1d3d135b94aae67a7892e03e8fd?s=200&d=robohash&r=`

Optional; min size 128 x 128px

**Enregistrer**

Le webhook est maintenant créé. Copiez la clé car nous allons la coller dans un script - **au niveau du variable "url"** - que nous allons voir juste après.

différencier vos workflows

## Webhooks

[+ Ajouter des webhooks](#)

3 webhooks

Nom	URL	
Quickstart Webhook	<code>https://chat.googleapis.com/v1/spaces/AAAAPihoQiU/messages?key=AlzaSy...</code>	⋮
Web_book_test	<code>https://chat.googleapis.com/v1/spaces/AAAAPihoQiU/messages?key=AlzaSy...</code>	⋮
Webhook JOB15	<code>https://chat.googleapis.com/v1/spaces/AAAAPihoQiU/messages?key=AlzaSy...</code>	⋮

Mis en place du script:

Le [tuto de google](#) propose un script pé-fait pour tester le webhook:

```
from json import dumps
from httplib2 import Http

# Copy the webhook URL from the Chat space where the webhook is registered.
# The values for SPACE_ID, KEY, and TOKEN are set by Chat, and are included
# when you copy the webhook URL.

def main():
    """Google Chat incoming webhook quickstart."""
    url = "https://chat.googleapis.com/v1/spaces/SPACE_ID/messages?key=KEY&token=TOKEN"
    app_message = {"text": "Hello from a Python script!"}
    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__":
    main()
```

Nous allons donc récupérer le script de google + le script “**ssh\_system\_mail.py**” créé précédemment et les fusionner dans un nouveau script nommé “**ssh\_google\_chat.py**”. Il faudra bien évidemment adapter le nouveau script car nous souhaitons ici envoyer des notifications et non plus des mails.

Pour ce faire, nous avons modifié la fonction “**send\_alert\_email**” en “**send\_alert\_app**” puis ajouté et adapté le script de Google.

```
def send_alert_app(server_name, cpu_usage, ram_usage, disk_usage):
    """Google Chat incoming webhook quickstart."""
    url = "https://chat.googleapis.com/v1/spaces/AAAAAPihQiu/messages?key=AIza"
    app_message = {"text": f"""
        Attention : Le serveur '{server_name}' a dépassé les seuils définis.

        Détails de l'état des ressources :
        - Utilisation CPU : {cpu_usage} % (Seuil : {CPU_THRESHOLD} %)
        - Utilisation RAM : {ram_usage} % (Seuil : {RAM_THRESHOLD} %)
        - Utilisation Disque : {disk_usage} % (Seuil : {DISK_THRESHOLD} %)

        Veuillez vérifier immédiatement.
    """}
    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)
```

Une fois le script terminé, on le lance et nous devons avoir ce résultat dans notre espace JOB15.

Webhook JOB15 Application 25 min



Attention : Le serveur 'Serveur FTP' a dépassé les seuils définis.

Détails de l'état des ressources :

- Utilisation CPU : 100.0 % (Seuil : 50.0 %)
- Utilisation RAM : 18.7265 % (Seuil : 80.0 %)
- Utilisation Disque : 29.0 % (Seuil : 80.0 %)

Veuillez vérifier immédiatement.

Attention : Le serveur 'Serveur FTP' a dépassé les seuils définis.

Détails de l'état des ressources :

- Utilisation CPU : 50.0 % (Seuil : 10.0 %)
- Utilisation RAM : 17.5581 % (Seuil : 80.0 %)
- Utilisation Disque : 29.0 % (Seuil : 80.0 %)

Veuillez vérifier immédiatement.