Mise en place du serveur web

Installation déjà faite d'apache2,

SSL autosigné

apt install openssl
openssl genrsa -out mydomain.key 2048
Créer une demande de signature de certificat (CSR)
openssl req -new -key mydomain.key -out mydomain.csr
Générer un certificat auto-signé (mettre les chemins de .crt et .key)
openssl x509 -req -days 365 -in mydomain.csr -signkey mydomain.key -out mydomain.crt
Configurer le serveur web
nano /etc/apache2/sites-available/default-ssl.conf
Modifier les chemins vers les fichiers de certificat et de clé privée
SSLEngine on

```
   SSLCertificateFile /etc/ssl/private/mon_serveur.crt
   SSLCertificateKeyFile /etc/ssl/private/mon_serveur.key

   SSLProtocol -ALL +TLSv1.1 +TLSv1.2 TLSv1.3
   SSLHonorCipherOrder On
   SSLCipherSuite ECDHE-RSA-AES128-SHA256:AES128-GCM-SHA256:HIGH:!MD5:!aNULL:!EDH:!
RC4
   SSLCompression off
```

Activer le site
a2enmod ssl
a2ensite default-ssl.conf
systemctl reload apache2

1. Installer PHP

Apache2 seul ne permet pas d'exécuter des scripts PHP, donc tu dois installer PHP et ses modules pour qu'il puisse interpréter les fichiers PHP.

- Installe PHP et les modules nécessaires :

apt update
apt install php libapache2-mod-php

Tu peux aussi installer des modules supplémentaires de PHP selon tes besoins (par exemple pour le support de bases de données, images, etc.) :

apt install php-mysql php-cli php-curl php-xml php-mbstring
apt install php-gd php-opcache php-zip php-imagick php-intl php-soap php-json php-sqlite3 php-bz2

2. Installer MySQL (ou MariaDB)

Si tu souhaites utiliser une base de données, tu peux installer **MySQL** ou **MariaDB** (MariaDB est une alternative à MySQL, souvent recommandée).

- Pour installer **MySQL** :

apt install mysql-server
Ensuite, sécurise l'installation de MySQL :
mysql_secure_installation
Et pour tester si MySQL fonctionne bien :
sudo systemctl status mysql

ou

- Pour installer **Mariadb** :

apt install mariadb-server
systemctl status mariadb
systemctl enable mariadb

Sécuriser l'installation de MariaDB

Après l'installation, il est fortement recommandé de sécuriser votre installation de MariaDB en exécutant un script de sécurisation. Ce script configure certaines options de sécurité importantes, comme la suppression des utilisateurs anonymes et la configuration du mot de passe pour l'utilisateur root de MariaDB.

mysql_secure_installation

Voici ce qui vous sera demandé pendant le processus de sécurisation :

- **Configurer le mot de passe pour l'utilisateur root ?** : Vous serez invité à définir un mot de passe pour l'utilisateur root de MariaDB (le mot de passe d'administration de la base de données).
- **Supprimer les utilisateurs anonymes ?** : Il est recommandé de répondre **Oui** pour supprimer les utilisateurs anonymes.
- **Désactiver l'accès à distance pour l'utilisateur root ?** : Il est recommandé de répondre **Oui** pour éviter que l'utilisateur root se connecte à distance.
- **Supprimer la base de données test ?** : Répondez **Oui** pour supprimer la base de données de test.
- **Recharger les privilèges des tables ?** : Répondez **Oui** pour appliquer les modifications.

Se connecter à MariaDB

Une fois la sécurisation terminée, vous pouvez vous connecter à MariaDB en utilisant l'utilisateur root et le mot de passe que vous avez défini précédemment.

mysql -u root -p

vous pouvez exécuter des commandes SQL.

Ajouter un utilisateur MariaDB

CREATE USER 'nouvel_utilisateur'@'localhost' IDENTIFIED BY 'mot_de_passe';
Ensuite, vous pouvez accorder des privilèges à cet utilisateur pour qu'il puisse gérer des bases de données : GRANT ALL PRIVILEGES ON *.* TO 'nouvel_utilisateur'@'localhost' WITH GRANT OPTION;
Enfin, rechargez les privilèges pour appliquer les modifications :
FLUSH PRIVILEGES;

Créer une base de donnée

Se connecter à MariaDB

Tout d'abord, vous devez vous connecter au serveur MariaDB avec un utilisateur ayant les privilèges suffisants (généralement l'utilisateur root).

sudo mysql -u root -p

CREATE DATABASE nom_de_la_base;

Vérifier la création de la base de données
SHOW DATABASES;

Utiliser la nouvelle base de données

Si vous souhaitez commencer à travailler avec la nouvelle base de données, vous devez la sélectionner en utilisant la commande USE :

USE ma_nouvelle_base;

Créer des tables dans la base de données

Une fois la base de données sélectionnée, vous pouvez commencer à créer des tables dans cette base de données. Par exemple, pour créer une table utilisateurs avec des colonnes id, nom, et email, vous pouvez utiliser la commande suivante :

```
CREATE TABLE utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    nom VARCHAR(100),
    email VARCHAR(100)
);
```

Cela crée une table utilisateurs avec trois colonnes : id, nom, et email. La colonne id est une clé primaire auto-incrémentée, ce qui signifie qu'elle s'incrémente automatiquement à chaque nouvel enregistrement.

Quitter MariaDB

Lorsque vous avez terminé, vous pouvez quitter l'interface MariaDB en tapant :

```
EXIT;
```

3. Configurer Apache pour PHP

Apache doit être configuré pour interpréter les fichiers PHP. Normalement, après l'installation du module libapache2-mod-php, il devrait le faire automatiquement, mais tu peux vérifier que le fichier php7.x.conf (ou php8.x.conf selon la version de PHP) est bien activé dans Apache :

- Si nécessaire, active le module PHP :

```
sudo a2enmod php8.2 # Remplace 7.x par la version installée.
sudo systemctl restart apache2
```

4. (Optionnel) Installer phpMyAdmin

Si tu veux une interface web pour gérer ta base de données, tu peux installer **phpMyAdmin**.

- Installation :

```
apt install phpmyadmin
```

5. Configurer les permissions

Assure-toi que les fichiers et répertoires dans /var/www/html ont les bonnes permissions pour Apache et PHP. Par exemple :

- Donne les bonnes permissions à ton répertoire web :

```
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 755 /var/www/html
```

6. Installer un certificat SSL

Si tu veux sécuriser ton site avec HTTPS, tu peux installer Certbot pour obtenir un certificat SSL gratuit de Let's Encrypt.

- Installe Certbot :

sudo apt install certbot python3-certbot-apache

Obtiens un certificat SSL pour ton domaine :
sudo certbot --apache

7. Redémarrer Apache

Après avoir installé tout ça, pense à redémarrer Apache pour que toutes les configurations prennent effet :

sudo systemctl restart apache2

Pour un second site :

Si tu veux héberger plusieurs sites web (aussi appelés "virtual hosts") sur ton serveur Apache, tu peux le faire en configurant des hôtes virtuels (Virtual Hosts). Voici comment procéder pour ajouter un deuxième site web à ton serveur Apache.

1. Créer un répertoire pour ton nouveau site web

Commence par créer un répertoire pour ton deuxième site web, par exemple /var/www/mon_deuxieme_site :

mkdir -p /var/www/mon_deuxieme_site

Ensuite, donne les bonnes permissions à ce répertoire :
sudo chown -R www-data:www-data /var/www/mon_deuxieme_site
sudo chmod -R 755 /var/www/mon_deuxieme_site

Place les fichiers de ton deuxième site dans ce répertoire. Par exemple, tu pourrais avoir une page d'index comme index.html ou index.php.

2. Créer un fichier de configuration pour le site

Ensuite, tu dois créer un fichier de configuration pour ton deuxième site dans Apache. Par convention, les fichiers de configuration des sites sont stockés dans le répertoire /etc/apache2/sites-available/. Crée un fichier de configuration pour ton deuxième site, par exemple mon_deuxieme_site.conf :

sudo nano /etc/apache2/sites-available/mon_deuxieme_site.conf

Ajoute-y la configuration suivante :

```
<VirtualHost *:80>
    ServerAdmin webmaster@mon_deuxieme_site.com
    ServerName mon_deuxieme_site.com
    DocumentRoot /var/www/mon_deuxieme_site

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Remplace mon_deuxieme_site.com par le nom de domaine de ton site ou l'adresse IP de ton serveur si tu n'as pas de domaine. Assure-toi que la directive DocumentRoot pointe bien vers le répertoire de ton deuxième site.

3. Activer le nouveau site

Une fois que tu as créé le fichier de configuration, tu dois activer ton nouveau site en utilisant la commande a2ensite :

a2ensite mon_deuxieme_site.conf

4. Désactiver le site par défaut (optionnel)

Si tu n'as pas besoin du site par défaut d'Apache (000-default.conf), tu peux le désactiver pour éviter les conflits :

sudo a2dissite 000-default.conf

5. Vérifier la configuration d'Apache

Avant de redémarrer Apache, vérifie si ta configuration est correcte avec la commande suivante :

sudo apache2ctl configtest

6. Redémarrer Apache

Pour que les changements prennent effet, redémarre Apache :

7. Configurer les DNS (si nécessaire)

Si tu utilises un nom de domaine comme mon_deuxieme_site.com, assure-toi que le DNS pour ce domaine pointe vers l'adresse IP de ton serveur. Si tu fais des tests localement, tu peux ajouter une entrée dans le fichier /etc/hosts de ton ordinateur pour simuler la résolution DNS.

Par exemple, ajoute cette ligne à ton fichier /etc/hosts :

127.0.0.1 mon_deuxieme_site.com

Cela redirigera mon_deuxieme_site.com vers localhost (ton propre serveur). Si ton serveur est accessible sur une IP publique, assure-toi que le domaine pointe bien vers cette adresse.

8. Accéder à ton deuxième site

Maintenant, si tout est bien configuré, tu devrais pouvoir accéder à ton deuxième site via le navigateur en utilisant l'URL http://mon_deuxieme_site.com (ou l'IP de ton serveur si tu n'as pas configuré de DNS).

9. (Optionnel) Configurer SSL pour le nouveau site

Si tu veux ajouter SSL (HTTPS) à ton deuxième site, tu peux utiliser **Certbot** pour générer un certificat SSL, comme pour le premier site.

Exécute cette commande pour obtenir un certificat SSL pour ton nouveau site :

sudo certbot --apache -d mon_deuxieme_site.com

Cela configurera automatiquement Apache pour utiliser HTTPS avec ton certificat SSL.

ok mais s'il y a plusieurs site web sur le même serveur, comment est ce possible car ils auront tous la même IP?

C'est une très bonne question ! Même si plusieurs sites web partagent la même adresse IP sur le même serveur, Apache peut toujours les distinguer grâce à un mécanisme appelé **"Virtual Hosts"** (hôtes virtuels).

Voici comment cela fonctionne et comment Apache gère plusieurs sites web sur une même adresse IP.

1. Le principe des Virtual Hosts (hôtes virtuels)

Apache peut gérer plusieurs sites web sur un même serveur grâce à des **Virtual Hosts**. Lorsqu'un client (navigateur) fait une demande HTTP à ton serveur, il envoie également un en-tête Host qui indique le nom du domaine demandé (par exemple, www.site1.com ou www.site2.com). Apache utilise ce nom de domaine (ou sous-domaine) pour savoir quel site web il doit renvoyer, même si tous les sites sont hébergés sur la même adresse IP.

Cela permet à Apache de répondre à la bonne demande en fonction de l'**hôte virtuel** configuré pour chaque domaine ou sous-domaine.

2. Comment Apache gère les Virtual Hosts

Chaque site web que tu ajoutes à ton serveur Apache est configuré avec un fichier de **VirtualHost**. Chaque fichier de configuration contient une directive ServerName qui correspond au nom de domaine ou sous-domaine auquel Apache doit répondre. Par exemple, pour deux sites différents, cela pourrait ressembler à ceci :

Exemple 1 : site1.com

```
<VirtualHost *:80>
   ServerAdmin webmaster@site1.com
   ServerName site1.com
   DocumentRoot /var/www/site1
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Exemple 2 : site2.com

```
<VirtualHost *:80>
   ServerAdmin webmaster@site2.com
   ServerName site2.com
   DocumentRoot /var/www/site2
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

3. **Comment Apache sait quel site servir ?**

Même si tous ces sites sont sur la même adresse IP, Apache utilise l'en-tête Host de la requête HTTP pour déterminer quel site doit être servi. Par exemple :

- Si un utilisateur accède à http://site1.com, Apache voit que l'en-tête Host est site1.com et il envoie la réponse du VirtualHost qui correspond à ce nom de domaine (celui avec ServerName site1.com).
- Si un utilisateur accède à http://site2.com, Apache voit que l'en-tête Host est site2.com et il envoie la réponse du VirtualHost qui correspond à ce nom de domaine (celui avec ServerName site2.com).

4. Port 80 et HTTPS (port 443)

Quand tu configures un site en HTTP (port 80), Apache sait quel site servir grâce à l'en-tête Host. Si tu utilises HTTPS, c'est pareil, mais sur le port **443**. Le processus reste identique, mais il est

important de noter que chaque VirtualHost HTTPS aura aussi une configuration SSL spécifique, où tu indiqueras les certificats SSL pour chaque site.

Exemple pour HTTPS avec un VirtualHost :

```
<VirtualHost *:443>
   ServerAdmin webmaster@site1.com
   ServerName site1.com
   DocumentRoot /var/www/site1
   SSLEngine on
   SSLCertificateFile /etc/ssl/certs/site1.crt
   SSLCertificateKeyFile /etc/ssl/private/site1.key
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Le port 443 est utilisé pour les connexions sécurisées (HTTPS), et Apache choisira le bon certificat SSL en fonction du ServerName de l'en-tête Host.

5. Limite de l'adresse IP

Il n'y a aucune limite quant au nombre de sites web que tu peux héberger sur une même adresse IP, à condition que chaque site ait son propre **nom de domaine** (ou sous-domaine) et que Apache soit configuré correctement avec des Virtual Hosts. En fait, c'est l'un des principaux avantages de l'utilisation des Virtual Hosts.

6. DNS et Configuration

Si tu héberges plusieurs sites sur une même IP, tu dois t'assurer que les **DNS** de chaque domaine (ou sous-domaine) pointent vers l'adresse IP de ton serveur. Par exemple :

- site1.com doit pointer vers ton adresse IP (publique ou locale).
- site2.com doit aussi pointer vers la même adresse IP.

Si tu fais des tests locaux (par exemple sur ton ordinateur), tu peux ajouter ces noms de domaine dans ton fichier /etc/hosts comme je l'ai expliqué précédemment.

7. Exemple de configuration complète pour plusieurs sites

Si tu veux héberger plusieurs sites sur le même serveur, tu pourrais avoir plusieurs fichiers de configuration comme suit :

- /etc/apache2/sites-available/site1.conf (pour site1.com)
- /etc/apache2/sites-available/site2.conf (pour site2.com)

Tu actives chaque site avec a2ensite, puis tu redémarres Apache pour appliquer les modifications :

```
sudo a2ensite site1.conf
sudo a2ensite site2.conf
```

sudo systemctl restart apache2

Conclusion

Apache gère plusieurs sites sur une seule IP en utilisant les **Virtual Hosts**, où chaque site est configuré pour répondre à un nom de domaine particulier. Bien que tous les sites partagent la même adresse IP, Apache peut les différencier en fonction du nom de domaine dans l'en-tête Host de la requête HTTP. Cela te permet d'héberger autant de sites que tu veux sur un même serveur, tant qu'ils ont des noms de domaine distincts.

Postfix

- **Description** : Postfix est un serveur de messagerie SMTP (Simple Mail Transfer Protocol), utilisé pour envoyer des e-mails.
- **Installation** : C'est probablement l'application principale dont tu auras besoin pour envoyer des e-mails depuis tes sites web.

sudo apt install postfix

Lorsque tu seras invité à choisir une configuration, choisis l'option "Site Internet" (si tu souhaites que ton serveur soit un relais d'e-mail classique). Ensuite, configure les paramètres comme le nom de domaine du serveur de messagerie.

2. Dovecot (optionnel, mais recommandé pour recevoir des e-mails)

- **Description** : Si tu souhaites également recevoir des e-mails, tu devras installer un serveur IMAP/POP3 comme Dovecot. Mais si ton objectif est uniquement l'envoi d'e-mails, ce n'est pas nécessaire.
- **Installation** :

apt install dovecot-core dovecot-imapd

SpamAssassin (optionnel mais utile)

- **Description** : Un logiciel de filtrage anti-spam pour éviter que tes e-mails ne soient considérés comme indésirables.
- **Installation** :

apt install spamassassin

Opération avec PHP : Mailer

- Pour envoyer des e-mails depuis des scripts PHP, tu peux configurer un **script PHP** pour utiliser ton serveur SMTP.
- Utilise une bibliothèque comme **PHPMailer** pour envoyer des e-mails via Postfix.

Installation de PHPMailer :

A mettre dans le dossier du projet ou dans un dossier dont le lien sera mis dans chaque dossier des sites.

composer require phpmailer/phpmailer

Installation dans un dossier commun

Créez un dossier partagé, par exemple :

```
mkdir /var/www/php-libs
cd /var/www/php-libs
composer require phpmailer/phpmailer
```

Ensuite, dans vos sites, ajoutez cette librairie :
Dans **site1** (/var/www/html/site1/sendmail.php) :

```
require '/var/www/php-libs/vendor/autoload.php';
use PHPMailer\PHPMailer\PHPMailer;
```

Dans **site2** (/var/www/html/site2/sendmail.php) :

```
require '/var/www/php-libs/vendor/autoload.php';
use PHPMailer\PHPMailer\PHPMailer;
```

Webmin

Télécharger la clé publique et la stocker dans `/usr/share/keyrings` :
wget -qO - http://www.webmin.com/jcameron-key.asc | tee /usr/share/keyrings/webmin-archive-keyring.gpg

Mettre à jour le fichier de dépôt Webmin pour utiliser le nouveau keyring
echo 'deb [signed-by=/usr/share/keyrings/webmin-archive-keyring.gpg] http://download.webmin.com/download/repository sarge contrib' | tee /etc/apt/sources.list.d/webmin.list

Mettre à jour à nouveau les paquets (cela vérifie si tout est ok):
apt update

Installer Webmin

sudo apt install webmin -y

Vérifier si Webmin fonctionne
systemctl status webmin
systemctl start webmin

Accéder à Webmin
https://TON_IP_SERVEUR:10000

Besoin de créer un utilisateur pour se connecter à webmin

Mot de passe et identifiant dans
nano /etc/webmin/miniserv.conf

Vérifiez les paramètres suivants :

- passwd_file: Assurez-vous que ce paramètre pointe vers un fichier existant qui contient les informations d'authentification.
- port: Assurez-vous que le port spécifié est correct (par défaut, c'est le port 10000).
- ssl: Assurez-vous que SSL est activé si vous utilisez une connexion HTTPS.
- **Créer manuellement un utilisateur Webmin** : Si le fichier miniserv.users n'existe pas, vous pouvez le créer manuellement et ajouter un utilisateur. Utilisez cette commande pour éditer (ou créer si nécessaire) le fichier /etc/webmin/miniserv.users :

sudo nano /etc/webmin/miniserv.users

root:x:<hashed_password>:0

Remplacez <hashed_password> par un mot de passe haché. Pour générer un mot de passe haché, vous pouvez utiliser une commande comme celle-ci :
echo -n 'votre_mot_de_passe' | openssl sha1

Redémarrer le service Webmin : Après avoir vérifié et modifié les configurations, redémarrez le service Webmin pour appliquer les modifications :
sudo service webmin restart