Procédure de déploiement d’une application Django sur un VPS Ubuntu ou Debian avec PostgreSQL, Gunicorn, Nginx et Supervisor

Table des matières

Introduction

Cette procédure guidera un apprenant dans le déploiement de son application Django hébergée sur un dépôt GitHub. Elle est conçue pour être compréhensible et claire, étape par étape, pour un premier déploiement. Le projet est hébergé sur un dépôt github et sera déployé sur un serveur Ubuntu en utilisant PostgreSQL, Gunicorn, Nginx et Supervisor.

Prérequis

- **Serveur VPS Ubuntu** avec une connexion initiale via le compte root (avec mot de passe). Par exemple, vous pouvez créer un Droplet DigitalOcean en cliquant sur “Create Droplet” ici ou créer un VPS Ionos en cliquant ici.
- **Adresse IP publique** du serveur (Vous la recevez dès la création de votre VPS.
- **Si vous avez un nom de domaine**, configurez vos zone DNS pour faire pointer ce domaine sur l’adresse IP de votre VPS,
- **Un dépôt GitHub** contenant le projet Django.

Étape 0 : Préparer la structure de configuration Django pour le déploiement

Avant de commencer la procédure de déploiement sur le serveur, nous allons adapter le projet Django sur votre machine locale afin de le préparer pour un environnement de développement et de production.

0.1 Créer une branche git pour votre nouvelle configuration

Afin d’éviter de faire des bêtises avec votre projet, vous allez créer une nouvelle branche pour la mise en place de vos nouveau settings:

```
$ git checkout -b settings-deploiement
```

0.2 Installer cookiecutter

Pour générer des fichiers de configuration pour la production, je vous ai préparé un template pour vous aider dans cette étape. Pour commencer installer cookiecutter :

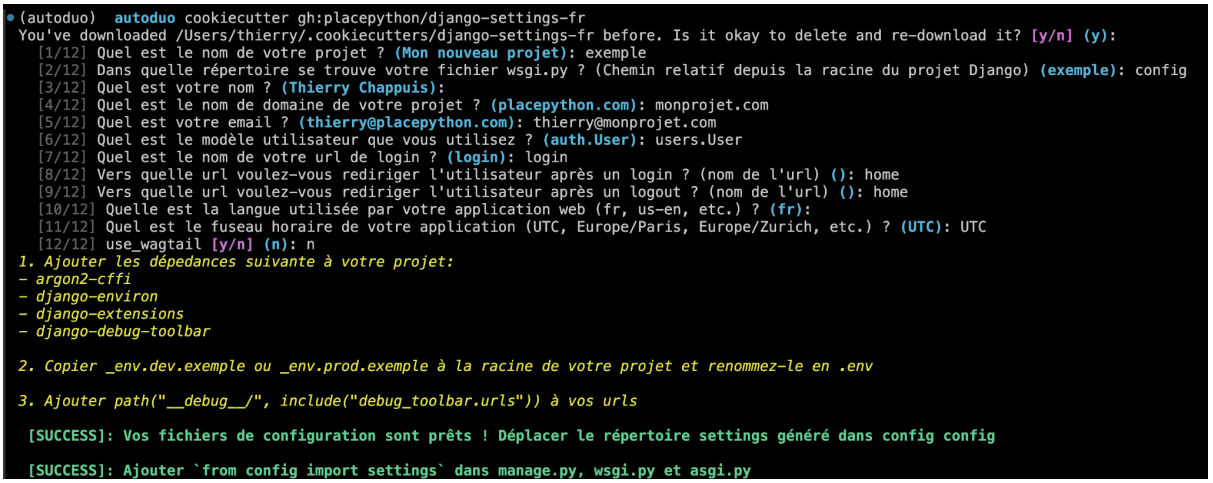
```
$ pip install cookiecutter
```

0.3 Créer des fichiers de settings pour le déploiement

A la racine de votre projet django, exécuter la commande :

```
$ cookiecutter gh:placepython/django-settings-fr
```

Répondez aux questions qui vous seront posées:



Une fois terminé, cet outil vous générera un répertoire nommé settings à la racine du projet. Editez settings/base.py et ajoutez-y vos propres apps django ainsi que vos propres configs (ce que vous avez dans votre settings.py). Ajoutez les configs propres au dev dans settings/dev.py et celles propres à la production dans settings/production.py.

Une fois que vous avez terminé, supprimez (ou sauvegardez) votre fichier de settings.py hors du répertoire de config de django (là où se trouve votre wsgi.py). Ensuite, copiez le fichier settings/_env.prod.exemple à la racine votre projet, et déplacer le répertoire settings/ dans le répertoire contenant votre wsgi.py.

Ajoutez ensuite les bibliothèques suivantes à votre projet :

```
- argon2-cffi
- django-environ
- django-extensions
- django-debug-toolbar
```

Ajoutez également la ligne suivante au sein des urlpatterns de votre module urls.py principal :

```
path("__debug__/", include("debug_toolbar.urls"))
```

Copiez ensuite le code suivant dans le **fichier manage.py** de votre projet, en remplaçant <config> par le nom du répertoire contenant votre module wsgi.py :

```
import os
import sys

from <config> import settings


def main():
    """Run administrative tasks."""
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "config.settings.dev")
```

```
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == "__main__":
    main()
```

Ajoutez finalement l'import suivant au début des modules **wsgi.py** et **asgi.py**, en remplaçant <config> par le nom du répertoire contenant votre module wsgi.py :

```
from <config> import settings
```

0.4 Valider les changements, fusionner avec main et poussez sur GitHub

Vérifiez que vous avez bien ajoutés les fichiers et répertoires suivants à votre fichier **.gitignore**:

```
.venv
.env
node_modules
```

Enregistrez toutes les modifications faites sur vos settings en versionnant votre code sur votre branche settings-deploiement:

```
$ git add .
$ git add status # vérifiez les fichiers qui seront ajoutés au commit
$ git commit -m "Prepared settings for deployment"
```

Après avoir vérifié que votre site fonctionne en local et que tous vos tests passent :), **fusionnez la branche avec main** et poussez ce dernier commit sur GitHub:

```
$ git checkout settings-deploiement
$ git rebase main
$ git checkout main
$ git merge settings-deploiement
$ git push origin main
```

Étape 1 : Connexion initiale et configuration de l'accès SSH par clé

Note: cette étape est d'actualité si vous ne savez pas ce qu'est une clé SSH ou si vous n'avez pas demandé à l'utiliser durant la création du VPS.

1.1 Connexion en tant que root :

Ouvrez votre terminal et connectez-vous à votre serveur en tant que root avec la commande suivante :

```
$ ssh root@votre_adresse_ip
```

1.2 Génération d'une clé SSH ED25519 sur votre machine locale :

Si vous n'avez pas encore de clé SSH, générez-en une de type ed25519 à partir d'un terminal:

```
$ ssh-keygen -t ed25519 -C "votre_email@example.com"
```

Vous pouvez laisser le chemin par défaut pour sauvegarder la clé dans <votre-répertoire-utilisateur>/.ssh/id_ed25519. Appuyez sur Entrée à chaque invite pour accepter les paramètres par défaut. N'oubliez pas de définir une phrase de passe sécurisée si nécessaire.

1.3 Affichage de la clé publique :

Affichez le contenu de votre clé publique générée (id_ed25519.pub) avec la commande suivante:

```
$ cat ~/.ssh/id_ed25519.pub
```

Copiez le texte affiché dans le terminal. Ce texte commencera par ssh-ed25519 suivi de votre clé publique et de votre email.

1.4 Ajout manuel de la clé publique sur le serveur :

Sur le serveur, créez le répertoire .ssh dans le répertoire personnel de root (s'il n'existe pas déjà), et définissez les permissions appropriées :

```
$ mkdir -p ~/.ssh
$ chmod 700 ~/.ssh
```

Ensuite, ouvrez ou créez le fichier authorized_keys :

```
$ nano ~/.ssh/authorized_keys
```

Collez la clé publique que vous avez copiée précédemment dans ce fichier, puis enregistrez et fermez-le. Ensuite, appliquez les bonnes permissions :

```
$ chmod 600 ~/.ssh/authorized_keys
```

1.5. Vérification de la connexion SSH par clé :

Déconnectez-vous du serveur et reconnectez-vous en tant que root sans mot de passe pour vérifier que l'accès SSH par clé fonctionne correctement :

```
$ ssh root@votre_adresse_ip -i ~/.ssh/id_ed25519
```

Étape 2 : Création d'un utilisateur dédié au projet

2.1 Création d'un nouvel utilisateur :

Créez un utilisateur nommé monprojetdjango (ou tout autre nom de votre choix) :

```
$ adduser monprojetdjango
```

Suivez les instructions pour définir un mot de passe sécurisé.

2.2 Ajout de l'utilisateur au groupe sudo :

Accordez les privilèges sudo à cet utilisateur monprojetdjango:

```
$ usermod -aG sudo monprojetdjango
```

2.3 Configuration de l'accès SSH pour le nouvel utilisateur :

Utilisez rsync pour copier le répertoire .ssh et son contenu du compte root vers le nouvel utilisateur monprojetdjango tout en préservant les permissions et en ajustant le propriétaire :

```
$ rsync --archive --chown=monprojetdjango:monprojetdjango ~/.ssh
/home/monprojetdjango
```

Cette commande effectue les opérations suivantes :

- Copie le répertoire ~/.ssh de root (y compris le fichier authorized_keys) dans le répertoire personnel de monprojetdjango.
- Prend en charge le changement de propriétaire (chown) pour le nouvel utilisateur (monprojetdjango), tout en maintenant les permissions d'origine du répertoire .ssh et de ses fichiers.

2.4 Vérification de la connexion SSH avec le nouvel utilisateur :

Déconnectez-vous et reconnectez-vous en tant que monprojetdjango :

```
$ ssh monprojetdjango@votre_adresse_ip -i ~/.ssh/id_ed25519
```

Étape 3 : Désactivation de la connexion SSH root

Si la connexion avec l'utilisateur monprojetdjango est un succès, nous pouvons maintenant désactiver la connexion en root sur le root, de manière à en renforcer la sécurité.

3.1 Modification de la configuration SSH :

Éditez le fichier de configuration SSH /etc/ssh/sshd_config en tant que root :

```
$ sudo nano /etc/ssh/sshd_config
```

3.2 Désactivation de la connexion root :

Recherchez la ligne suivante et modifiez-la pour interdire l'accès SSH au compte root :

```
PermitRootLogin no
```

3.3 Redémarrage du service SSH :

```
$ sudo systemctl restart ssh
```

Étape 4 : Configuration de l'environnement de déploiement

4.1 Installation des paquets nécessaires :

Mettez à jour la liste des paquets et installez les dépendances nécessaires :

```
$ sudo apt update
$ sudo apt install python3-venv python3-dev libpq-dev postgresql
postgresql-contrib nginx supervisor git build-essential curl
```

4.2 Création de la base de données PostgreSQL :

Connectez-vous à PostgreSQL en tant qu'utilisateur postgres pour créer la base de données et l'utilisateur PostgreSQL :

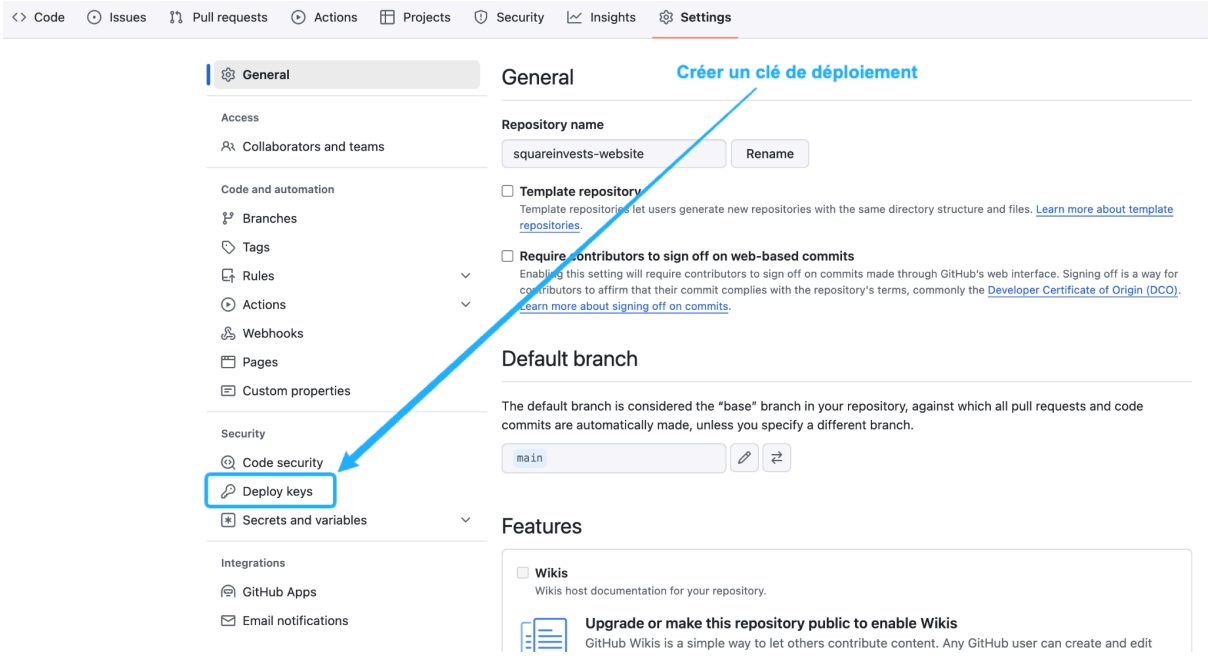
```
$ sudo -u postgres psql
```

Dans l'invite PostgreSQL, exécutez les commandes suivantes :

```
CREATE DATABASE monprojetdjango;

CREATE USER monprojetdjango WITH PASSWORD 'mot_de_passe_securisé';

ALTER ROLE monprojetdjango SET client_encoding TO 'utf8';

ALTER ROLE monprojetdjango SET default_transaction_isolation TO 'read committed';

ALTER ROLE monprojetdjango SET timezone TO 'UTC';

GRANT ALL PRIVILEGES ON DATABASE monprojetdjango TO monprojetdjango;

\q
```

4.3 Clonage du dépôt GitHub :

Note: si votre dépôt GitHub est privé, vous aurez au préalable besoin de créer une clé ssh de déploiement avec la commande `ssh-keygen -t ed25519 -C "votre_email@example.com"` exécutée cette fois sur votre VPS. Puis vous devrez ajouter la clé publique sur GitHub à partir du menu présenté dans la capture d'écran ci-dessous:



Clonez ensuite le dépôt GitHub de l'application Django dans le répertoire personnel de monprojetdjango:

```
$ git clone https://github.com/placepython/monprojetdjango.git
~/monprojetdjango
$ cd ~/monprojetdjango
```

4.4 Création de l'environnement virtuel et installation des dépendances :

Créez un environnement virtuel pour le projet et activez-le :

```
$ cd ~/monprojetdjango
$ python3 -m venv .venv
$ source env/bin/activate
$ pip install -r requirements.txt
```

Si vous utilisez pipenv, installez pipenv avec `python3 -m pip install -user pipenv`, puis déconnectez-vous et reconnectez-vous à votre VPS. Exécutez ensuite les commandes suivantes:

```
$ cd ~/monprojetdjango
$ mkdir .venv
$ pipenv install
```

Si vous utilisez uv ou poetry, faites pareil, installez votre outils de prédilection pour le compte utilisateur courant, puis créez l'environnement virtuel et finalement installez les dépendances du projet.

4.5 Configuration de Django :

Si vous avez suivi les instructions à la section 0 de ce document, il vous suffit à ce stade de **renommer le fichier *env.prod.exemple en .env***. *Remplacez ci-dessous la valeur de la SECRET*_KEY ainsi que les valeurs MONDOMAINE, USER, PASSWORD, EMAIL, EMAILPASSWORD, DBHOST et EMAILHOST par vos propres valeurs:

```
DJANGO_DEBUG=False
DJANGO_SECRET_KEY=<ajouter-une-secret-key-super-secrete>
DJANGO_DATABASE_URL=postgres://USER:PASSWORD@DBHOST:5432/DBNAME
DJANGO_EMAIL_URL=smtp+ssl://EMAIL:EMAILPASSWORD@EMAILHOST:465
DJANGO_ALLOWED_HOSTS=MONDOMAINE,localhost,127.0.0.1
DJANGO_SETTINGS_MODULE=config.settings.production
```

N'hésitez pas d'ajouter à ce fichier .env des variables d'environnement propres à votre projet.

4.6 Migrations de la base de données :

Appliquez les migrations de la base de données, collectez les fichiers statiques et créez un superutilisateur :

```
$ python manage.py migrate
$ python manage.py collectstatic
$ python manage.py createsuperuser
```

Étape 5 : Configuration de Gunicorn avec Supervisor

5.1 Création du fichier de configuration Supervisor :

Créez un fichier de configuration Supervisor pour gérer Gunicorn. Remplacez <monprojetdjango> par le nom de votre projet:

```
$ sudo nano /etc/supervisor/conf.d/<monprojetdjango>.conf
```

Ajoutez le contenu suivant, en remplaçant <monprojetdjango> par le nom de votre projet et <config> par le nom du répertoire qui contient votre wsgi.py :

```
[program:<monprojetdjango>]
command=/home/<monprojetdjango>/<monprojetdjango>/.venv/bin/gunicorn
--workers 3 --bind unix:/home/<monprojetdjango>/<monprojetdjango>.sock
<config>.wsgi:application
directory=/home/<monprojetdjango>/<monprojetdjango>
user=<monprojetdjango>
autostart=true
autorestart=true
stderr_logfile=/var/log/monapplication.err.log
stdout_logfile=/var/log/monapplication.out.log
```

Dans le fichier ci-dessus, les lignes suivantes :

```
command=/home/<monprojetdjango>/<monprojetdjango>/.venv/bin/gunicorn
--workers 3 --bind unix:/home/<monprojetdjango>/<monprojetdjango>.sock
<config>.wsgi:application
```

Ne sont qu’une seule ligne !

5.2 Démarrage de Supervisor et activation de la configuration :

Une fois que supervisor a été configuré pour démarrer notre serveur gunicorn, il nous faut activer la configuration et démarrer notre application. Ici, remplacez <monprojetdjango> par le nom du projet que vous avez utilisé dans la configuration:

```
$ sudo supervisorctl reread
$ sudo supervisorctl update
$ sudo supervisorctl start <monprojetdjango>
```

Étape 6 : Configuration de Nginx pour le proxy avec Gunicorn

6.1 Création du fichier de configuration Nginx :

Créez un nouveau fichier de configuration Nginx en remplaçant <monprojetdjango> par le nom de votre projet :

```
$ sudo nano /etc/nginx/sites-available/<monprojetdjango>
```

Ajoutez le contenu suivant :

```
server {
    listen 80;
    server_name votre_domaine_ou_adresse_ip;

    # Redirection de /static vers le répertoire pointé par STATIC_ROOT
    location /static/ {
        alias /home/<monprojetdjango>/<monprojetdjango>/staticfiles/;
    }

    # Redirection de /media vers le répertoire media correspondant
    location /media/ {
        alias /home/<monprojetdjango>/<monprojetdjango>/media/;
    }

    # Configuration de proxy pour Gunicorn
    location / {
        include proxy_params;
        proxy_pass
http://unix:/home/<monprojetdjango>/<monprojetdjango>.sock;
    }
}
```

6.2 Donner les droits à Nginx d'accéder aux fichiers statiques et de media:

Pour que Nginx puisse accéder à vos fichiers statiques ou fichiers de media, vous devez ajouter www-data au groupe de votre utilisateur. Exécutez la commande suivante en remplaçant <monprojetdjango> par le nom de votre projet :

```
$ sudo usermod -aG <monprojetdjango> www-data
```

6.3 Activation du fichier de configuration Nginx :

Exécutez les commandes suivantes pour activer votre nouvelle configuration Nginx :

```
$ sudo ln -s /etc/nginx/sites-available/<monprojetdjango>
/etc/nginx/sites-enabled
$ sudo nginx -t
$ sudo systemctl restart nginx
```

6.4 Ouverture du pare-feu :

Ouvrez le port 80 pour Nginx :

```
$ sudo ufw allow 'Nginx Full'
```

Étape 7 : Test et accès à l'application

Visitez l'adresse IP de votre serveur ou le nom de domaine associé dans votre navigateur. Vous devriez voir l'application Django s'afficher.

Étape 8 : Création d'un certificat SSL pour servir votre site en https

Dans cette étape, nous allons configurer un certificat SSL gratuit pour sécuriser les connexions HTTPS à votre site en utilisant **Let's Encrypt** et **Certbot**. Cette configuration est destinée aux utilisateurs ayant configuré Nginx avec un **nom de domaine**. Nous allons utiliser Certbot pour générer et configurer automatiquement le certificat SSL sur Nginx, puis nous vérifierons que le certificat se renouvelle automatiquement.

8.1 Installation de Certbot

Certbot est l'outil recommandé pour obtenir des certificats SSL de Let's Encrypt. Installez-le en suivant les instructions ci-dessous :

```
$ sudo snap install core; sudo snap refresh core
```

Ensuite, installez Certbot en utilisant le paquet snap :

```
$ sudo snap install --classic certbot
```

Puis, créez un lien symbolique pour permettre d'exécuter certbot directement :

```
$ sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

8.2 **Autorisation du trafic HTTPS via le pare-feu**
Si vous utilisez ufw comme pare-feu, vous devez autoriser le trafic HTTPS (port 443). Ajoutez la règle suivante pour autoriser le trafic HTTPS et supprimez l'ancienne règle HTTP :

```
$ sudo ufw allow 'Nginx Full'
$ sudo ufw delete allow 'Nginx HTTP'
```

Vérifiez que le pare-feu autorise maintenant le trafic HTTPS :

```
$ sudo ufw status
```

Vous devriez voir Nginx Full autorisé, indiquant que les ports HTTP (80) et HTTPS (443) sont ouverts.

8.2 Obtention et installation du certificat SSL

Exécutez la commande suivante pour générer un certificat SSL et configurer Nginx automatiquement. Remplacez votre_domaine.com par votre nom de domaine :

```
$ sudo certbot --nginx -d votre_domaine.com -d www.votre_domaine.com
```

- --nginx : Spécifie que Certbot doit utiliser le plugin Nginx pour configurer le certificat.
- -d : Spécifie le ou les noms de domaine pour lesquels vous souhaitez obtenir un certificat.

Lors de l'exécution de la commande, vous devrez fournir une adresse e-mail pour recevoir les notifications relatives à l'expiration du certificat, et accepter les conditions d'utilisation de Let's Encrypt. Certbot configurera automatiquement Nginx avec le certificat SSL généré et mettra en place une redirection automatique de HTTP vers HTTPS.

8.3 Vérification de l'installation et configuration HTTPS

Une fois l'installation terminée, Certbot affichera un message confirmant la réussite de l'opération :

```
IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/votre_domaine.com/fullchain.pem
```

```
   Your key file has been saved at:
   /etc/letsencrypt/live/votre_domaine.com/privkey.pem
   ...
```

Vous pouvez maintenant visiter votre site en utilisant https://votre_domaine.com et vérifier que la connexion est sécurisée (indiquée par un cadenas dans la barre d'adresse du navigateur).

8.4 Vérification de la redirection automatique de HTTP vers HTTPS

Certbot devrait avoir configuré automatiquement une redirection HTTP vers HTTPS. Pour vérifier cela, ouvrez à nouveau le fichier de configuration de votre domaine :

```
$ sudo nano /etc/nginx/sites-available/<monprojetdjango>
```

Assurez-vous que le bloc de redirection est présent :

```
server {
    listen 80;
    server_name votre_domaine.com www.votre_domaine.com;

    # Rediriger toutes les requêtes HTTP vers HTTPS
    return 301 https://$host$request_uri;
}
```

Cela garantit que toute tentative d'accès à http://votre_domaine.com est redirigée automatiquement vers https://votre_domaine.com.

8.5 Test de renouvellement automatique

Les certificats Let's Encrypt sont valides pendant 90 jours. Pour s'assurer que le renouvellement automatique est bien configuré, exécutez un test avec la commande suivante :

```
$ sudo certbot renew --dry-run
```

Si tout est configuré correctement, vous verrez un message indiquant que le test de renouvellement a réussi. Certbot utilise un **timer systemd** qui s'exécute deux fois par jour pour vérifier le statut des certificats et les renouveler automatiquement s'ils expirent dans moins de 30 jours.

Vous avez maintenant configuré un certificat SSL gratuit pour votre site, activé le support HTTPS, et mis en place un renouvellement automatique. Votre site est désormais sécurisé avec HTTPS, et vous pouvez surveiller les renouvellements automatiques via les logs de Certbot ou les notifications par e-mail.

Si vous rencontrez des problèmes ou souhaitez des informations complémentaires, consultez la documentation officielle de Certbot ou les logs de Certbot sur votre serveur :

```
$ sudo journalctl -u snap.certbot.renew.service
```

Votre site est désormais accessible de manière sécurisée via HTTPS !