Procédure d'installation et de déploiement pour la SAÉ23.

Louis DESVERNOIS

19 juin 2022

Table des matières

Table des figures

Table des codes

1 Machine virtuelle et base de données

Notre solution se base sur une machine virtuelle utilisant la dernière version de Debian 11. La technologie utilisée pour créer cette machine virtuelle n'a peu d'importance, tant que celle-ci est accessible (e.g., carte réseau en mode bridge).

1.1 Installation des dépendances

Après l'installation d'un système minimal Debian 11, nous avons besoin d'installer les différents composants nécessaires au déploiement d'un serveur MariaDB ainsi qu'un serveur HTTP Nginx.

```
apt install git mariadb-server nginx python3 python3-pip python3-venv python3-dev
↪  libmariadb-dev ufw -y
```

Code 1 – Installation des dépendances

1.2 Configuration initiale de la base de données

En installant le paquet `mariadb-server`, le gestionnaire de paquets apt a déjà automatiquement activé le service. Il ne nous reste donc qu'à configurer le serveur.

```
mysql -sfu root <<EOS
UPDATE mysql.user SET Password=PASSWORD('toto') WHERE User='root';
DELETE FROM mysql.user WHERE User='';
DROP DATABASE IF EXISTS test;
DELETE FROM mysql.db WHERE Db='test' OR Db='test\\_%';
FLUSH PRIVILEGES;
CREATE USER 'toto'@'localhost';
EOS
```

Code 2 – Configuration initiale du serveur MariaDB

Pour la configuration, nous utilisons la commande `mysql -sfu root` pour nous connecter à la base de données et ignorer les erreurs (Code 2). Pour commencer, nous changeons le mot de passe de l'utilisateur "root", nous supprimons tous les utilisateurs anonymes, nous supprimons la base de données "test" si elle existe, puis nous créons l'utilisateur "toto", qui permettra à Django d'accéder à la base de données.

1.3 Création de l'utilisateur, clonage du dépôt GitHub et exécution du script SQL

Pour des raisons de sécurité, il est préférable de ne pas exécuter le code python de notre site avec le super-utilisateur, c'est pour cela que nous créons un utilisateur ainsi que son dossier personnel sur notre serveur.

```
useradd -m toto
su - toto -c "git clone https://github.com/ldsvrn/SAE23-TraficAerien /home/toto/django"
```

Code 3 – Création de l'utilisateur et clonage du dépôt GitHub

Après la création de l'utilisateur avec la commande `useradd -m`, nous pouvons cloner le dépôt avec la commande `git clone <url> <dst>` [1].

1. "su - toto -c" est utilisé dans le code 3 pour exécuter la commande avec l'utilisateur toto

```
mysql -u root -p'toto' < /home/toto/django/SAE_23_BDD.sql
mysql -sfu root -p'toto'<<EOS
-- permission d'acces à la base de donnée
GRANT ALL PRIVILEGES ON sae_23.* TO 'toto'@'localhost';
EOS
```

Code 4 – Importation de notre script SQL

Ensuite, nous pouvons utiliser les commandes ci-dessus (4) pour importer notre script SQL préalablement téléchargé lors du `git clone` exécuté précédemment (Code 3). Nous octroyons ensuite à l'utilisateur "toto" tous les droits sur la base de données importée.

2 Environnement virtuel Python

Pour faire fonctionner Django, nous allons avoir besoin d'un environnement virtuel (venv) pour installer Django et ses dépendances sans les installer pour tout le système. Travailler avec des venv permet de garantir que paquets installés soient toujours les mêmes.

2.1 Création de l'environnement et installation des paquets

Le module `venv` de Python nous permet de créer ces environnements facilement avec la commande `python -m venv .venv`. En supposant que l'on utilise le shell bash, nous pouvons ensuite activer cet environnement grâce à la commande `source`. Une fois l'environnement virtuel activé, nous pouvons simplement utiliser `pip3` pour installer les dépendances de notre projet [2]. Nous pouvons donc exécuter les commandes suivantes, cette fois ci avec l'utilisateur créé précédemment (Code 3).

```
python -m venv .venv
source .venv/bin/activate
pip3 install django django-admin mysqlclient gunicorn crispy-bootstrap5 Pillow reportlab
```

Code 5 – Création du venv et installation des paquets

2.2 Configuration du projet Django (settings.py)

Une fois toutes dépendances python installées dans l'environnement virtuel, nous devons modifier les paramètres de notre projet Django afin d'utiliser la base de données externe. Il est intéressant de vérifier si le répertoire des fichiers statiques (e.g., images, css) est correctement configuré [3].

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'sae_23',
        'USER': 'toto',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Code 6 – settings.py : Paramétrage de la base de données

2. Il est également possible d'utiliser le fichier requirements.txt avec la commande "pip3 install -r requirements.txt"
3. Normalement, cela est déjà configuré automatiquement à la création du projet

2.3 Préparation du projet au déploiement

Avant de commencer à mettre en place notre serveur web, nous avons besoin de préparer notre projet au déploiement. Pour cela nous devons utiliser le fichier `manage.py` pour préparer la base de données ainsi que les fichiers statiques. En nous plaçant à la racine de notre projet, nous pouvons exécuter les commandes suivantes [4].

```
python3 manage.py makemigration
python3 manage.py migrate
python3 manage.py collectstatic
```

Code 7 – Préparation de la base de données et copie des fichiers statiques

Les commandes `migrate` et `makemigration` préparent la base de données pour son utilisation par Django, tandis que `collectstatic` copie les fichiers statiques vers le répertoire configuré dans `settings.py`.

3 Mise en place de Gunicorn et du serveur web Nginx

Maintenant que notre projet Django est correctement configuré, il ne reste plus que le serveur web à mettre en place. Pour cela, nous allons utiliser `gunicorn`, installé en Code 5, pour interfacer Django avec Nginx. En effet, il est impossible de directement utiliser Nginx pour distribuer les pages de notre site, dans notre cas, nous allons le configurer comme un *reverse proxy* à Gunicorn.

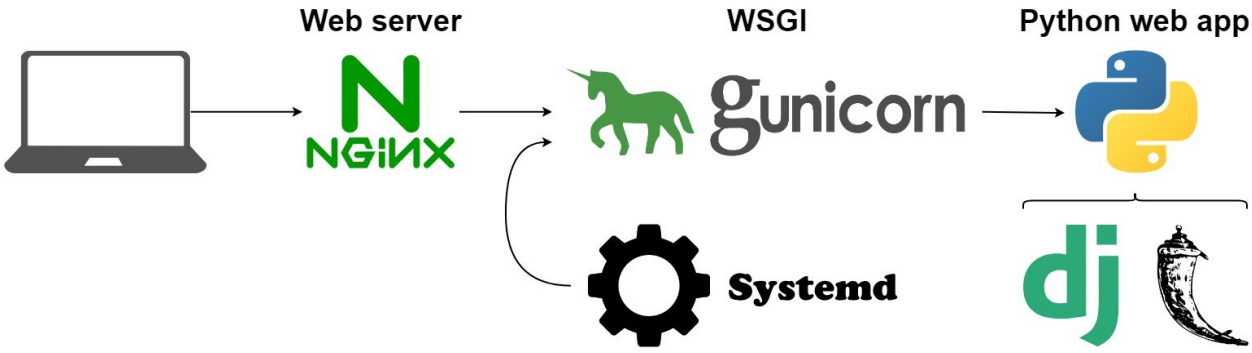

FIGURE 1 – Nginx + Gunicorn + Systemd + Django

Comme montré en Figure 1, nous allons utiliser Gunicorn en tant que *service systemd*, ce qui permettra, entre autre, le démarrage automatique du serveur.

4. Commandes à exécuter avec l'utilisateur crée en Code 3

3.1 Gunicorn

Pour commencer, nous devons créer deux fichiers pour interfacer systemd à Gunicorn, `gunicorn.socket` et `gunicorn.service`.

```
[Unit]
Description=gunicorn socket
[Socket]
ListenStream=/run/gunicorn.sock
[Install]
WantedBy=sockets.target
```

Code 8 – /etc/systemd/system/gunicorn.socket

Ce fichier permet de créer un *socket* (ou une *interface de connexion*), qui est, pour faire simple, un moyen que plusieurs applications peuvent utiliser pour communiquer. Nous allons utiliser cela pour connecter Gunicorn à Nginx.

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target
[Service]
User=toto
Group=www-data
WorkingDirectory=/home/toto/django
ExecStart=/home/toto/django/.venv/bin/gunicorn --access-logfile - --workers 3 --bind
↪  unix:/run/gunicorn.sock SAE23.wsgi:application
[Install]
WantedBy=multi-user.target
```

Code 9 – /etc/systemd/system/gunicorn.service

La création du service nous permet de lancer Gunicorn sous la forme d'un *daemon*. Dans ce fichier, nous précisons l'utilisateur et le groupe avec lesquels le service doit être exécuté ainsi que le service doit être exécuté après l'initialisation du réseau. Nous pouvons maintenant activer Gunicorn au démarrage.

```
systemctl start gunicorn.socket
systemctl enable gunicorn.socket
```

Code 10 – Activation du socket Gunicorn

3.2 Nginx

Une fois Gunicorn correctement paramétré, nous pouvons configurer Nginx pour rediriger les connexions sur le socket. Pour cela il faut créer un fichier *site* en suivant la bonne syntaxe.

```
server {
    listen 80;
    server_name sae23.louis.systems;
    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/toto/django;
    }
    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;
    }
}
```

Code 11 – /etc/nginx/sites-available/sae23

Pour cet exemple, Nginx écoute sur le port 80 et si un client se connecte en utilisant le domaine `sae23.louis.systems` [5], il redirige /static vers le répertoire précédemment configuré, le reste est redirigé vers le socket Gunicorn. Une fois ce fichier crée, nous devons l'activer avec un lien symbolique.

```
ln -s /etc/nginx/sites-available/sae23 /etc/nginx/sites-enabled/
```

Code 12 – Activation du site Nginx

Pour finir, nous pouvons redémarrer le service Nginx avec systemd pour prendre en compte les changements.

```
systemctl restart nginx
```

Code 13 – Redémarrage du service Nginx

4 Script de déploiement automatique

Pour faciliter le déploiement, nous avons écrit un simple script bash permettant d'automatiser toutes les étapes de cette procédure dans une VM de Debian 11 fraîchement installée avec un accès root [6]. Pour utiliser ce script, nous pouvons simplement utiliser `curl` [7] pour le télécharger directement de GitHub.

```
apt install curl
curl -sSL
↪  https://raw.githubusercontent.com/ldsvrn/SAE23-TraficAerien/main/server_install.sh |
↪  bash
```

Code 14 – Exécution du script de déploiement automatique

Une fois le script exécuté, il est normalement possible d'accéder au serveur. Malheureusement, le script n'est pas interactif, il n'est donc pas universel, en effet, certains détails tels que le `ALLOWED_HOSTS` de Django ou le `server_name` de Nginx nécessitent d'être modifiés en fonction de l'adresse du serveur. Une solution serait de créer une image Docker du projet.

5. Il s'agit du domaine d'un VPS utilisé pour tester le déploiement, car nous n'avons pas réussi à utiliser les cartes bridges sans les droits d'administrateur sur les PC fixes
6. Le script crée un utilisateur "toto", il est donc préférable que celui-ci n'existe pas déjà
7. Il est normalement préférable pour la sécurité de lire un script avant de l'exécuter