HTTPS POUR FLASK

Utilisation de Gunicorn et NGINX. Tuto disponible pour Raspberry (Debian) ou Ubuntu Server
Vérification de l'installation des outils python3 :

```
$sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-setuptools
```

Étape 1 - Ajouter les redirections DNS au nom de domaine

NOTE POUR DDNS :

Un DDNS gratuit tel qu'avec No-IP n'as pas besoin de redirection car par défaut il en a une ! Cependant on ne pourra pas accéder au site avec www.

Étape 2 - NGINX

```
$ sudo apt update
$ sudo apt install nginx
```

Réglage du pare-feu (UFW présent par défaut sur Ubuntu Serveur) :

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

Voir si Nginx est bien en fonctionnement :

```
$ systemctl status nginx
```

Étape 3 - Préparation pour Gunicorn

Nouveau dossier pour le projet :

```
$ mkdir ~/APPLI
$ cd  ~/APPLI
```

Installation d'un environnement virtuel :

```
$ sudo apt install python3-venv
$ python3 -m venv VENVAPP
$ source VENVAPP/bin/activate
```

(Pour quitter celui-ci ultérieurement : $deactivate)

Quelle que soit la version de Python que vous utilisez, lorsque l'environnement virtuel est activé, vous devez utiliser la commande pip (et non pip3).

Installation de wheel qui permet d'assurer que nos paquets s'installeront même s'il leur manque des paquets (regroupe un grand nombre de librairies)

```
$ pip install wheel
```

Installation de Gunicorn et Flask si pas installé

```
$ pip install gunicorn flask
```

(Pour mon projet pip install requests, pip install mysql.connector)

Étape 4 - Configuration d'une appli Flask simple

```
$ nano ~/APPLI/main.py
```

>>>Exemple de base (à copier/coller)

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "<h1 style='color:blue'>Hello There!</h1>"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Si pare feu activé :

```
$ sudo ufw allow 5000
```

Lancement de l'appli Flask :

```
$ python main.py
```

Tester son accessibilité ! : http://@IP:5000

Ici toujours en HTTP

Appuyez sur CTRL-C dans la fenêtre de votre terminal pour arrêter le serveur de développement Flask.

Création du point d'entrée WSGI

```
$ nano ~/APPLI/wsgi.py
```

>>>Exemple de base (à copier/coller)

```
from main import app    #nom fichier python !

if __name__ == "__main__":
    app.run()
```

Étape 4 - Configuration de Gunicorn

```
$ cd  ~/APPLI
```

*Lancement de l'appli Flask **AVEC GUNICORN**:*

```
$ gunicorn --bind 0.0.0.0:5000 wsgi:app
```

Tester son accessibilité ! : http://@IP:5000

Ici toujours en HTTP

Appuyez sur CTRL-C dans la fenêtre de votre terminal pour arrêter le serveur de développement Flask.

Nous en avons maintenant fini avec notre environnement virtuel, nous pouvons donc le désactiver (Toutes les commandes Python utiliseront à nouveau l'environnement Python du système):

```
$ deactivate
```

Étape 5 - Configuration du démarrage du serveur auto

Création d'un service :

```
$ sudo nano /etc/systemd/system/appli.service
```

#nom à donner

Ensuite, définissons le répertoire de travail et la variable d'environnement PATH afin que le système d'initialisation sache que les exécutables du processus sont situés dans notre environnement virtuel. Précisons également la commande de démarrage du service. Cette commande fera ce qui suit :

Démarrez 3 processus de travail (mais vous devez ajuster cela si nécessaire)

- Créez et reliez à un fichier socket Unix, APPLI.sock, dans notre répertoire de projet. Nous fixerons une valeur d'umask de 007 pour que le fichier socket soit créé en donnant l'accès au propriétaire et au groupe, tout en limitant tout autre accès
- Précisez le nom du fichier du point d'entrée du WSGI, ainsi que le nom de l'appel Python dans ce fichier (wsgi:app)
- Systemd exige que nous donnions le chemin complet à l'exécutable Gunicorn, qui est installé dans notre environnement virtuel.

>>>A copier/coller (ou à modifier) **#mettre son propre user #nom dossier #nom env virtuel**

```
[Unit]
Description=Execution de GUNICORN pour le projet
After=network.target

[Service]
User= pi
Group=www-data
WorkingDirectory=/home/pi/APPLI
Environment="PATH=/home/pi/APPLI/VENVAPP/bin"
ExecStart=/home/pi/APPLI/VENVAPP/bin/gunicorn --workers 3 --bind
unix:APPLI.sock -m 007 wsgi:app

[Install]
WantedBy=multi-user.target
```

Nous pouvons maintenant démarrer le service Gunicorn que nous avons créé et l'activer afin qu'il démarre au boot :

```
$   sudo systemctl start appli
$   sudo systemctl enable appli
```

Voir le statut d'exécution :

```
$   sudo systemctl status appli
```

Si on voit >>active (running) c'est que Gunicorn lance bien le serveur.
A ce point il faut configurer NGINX pour que le site « s'exécute »

Étape 6 - Configuration de Nginx pour les demandes de proxy

Créer un nouveau fichier de configuration du bloc serveur dans le répertoire sites-available de Nginx :

```
$   sudo nano /etc/nginx/sites-available/appli
```

ATTENTION POUR LA SUITE !

>>>>>>A copier/coller en modifiant **AVEC UN NOM DE DOMAINE !**

```
server {
        listen 80;
        server_name projet1.com www.projet1.com ;

        location / {
                include proxy_params;
                proxy_pass http://unix:/home/pi/APPLI/APPLI.sock;
        }
}
```

>>>>>>A copier/coller en modifiant **AVEC UN DDNS !**

```
server {
        listen 80;
        server_name projet1.ddns.net ;

        location / {
                include proxy_params;
                proxy_pass http://unix:/home/pi/APPLI/APPLI.sock;
        }
}
```

Activer la configuration du bloc serveur Nginx en reliant le fichier au répertoire sites-enabled de NGINX :

```
$      sudo ln -s /etc/nginx/sites-available/appli /etc/nginx/sites-enabled
```

Tester les erreurs de syntaxe :

```
$  sudo nginx -t
```

Redémarrer le service Nginx :

```
$  sudo systemctl restart nginx
```

Si pare feu activé :

```
$  sudo ufw delete allow 5000
$  sudo ufw allow 'Nginx Full'
```

Tester son accessibilité ! : http://projet1.com
Ici toujours en HTTP

Étape 6 — Sécurisation de l'application (HTTPS)

Installez le paquet Nginx de Certbot :

```
$   sudo apt install python3-certbot-nginx
```

Certbot propose différents moyens d'obtenir des certificats SSL par le biais de plugins. Le plugin Nginx se charge de reconfigurer Nginx et de charger la configuration chaque fois que nécessaire. Pour utiliser ce plugin, tapez ce qui suit : (Cela exécute certbot avec le plugin --nginx, en utilisant -d pour spécifier les noms pour lesquels nous aimerions que le certificat soit valide.)

ATTENTION POUR LA SUITE !

AVEC UN NOM DE DOMAINE :

```
$     sudo certbot --nginx -d projet1.com -d www.projet1.com
```

AVEC UN DDNS :

```
$    sudo certbot --nginx -d projet1.ddns.net
```

Certbot demandera comment vous souhaitez configurer vos paramètres HTTPS :

```
>>> 1    #http et https
>>> 2    #https uniquement !
```

--> Choix n°2 pour ma part, je me sépare complétement d'HTTP

Tester son accessibilité ! : ***https://projet1.com***

Pour renouveler le certificat :

```
$ sudo certot renew
```