Virtualisation SAE23
Solution Informatique

Pour commencer dans ce projet dans la partie virtualisation , j'ai personnellement commencé par créer ma base de données externe sur mysql soit MariaDB qui est un système de gestion de données sur Linux.

MariaDB (Mysql) config (Base de donnée externe) :

Il faut commencer par installer tous les paquets dont nous avons besoin. Cela inclura le serveur Web Apache, le module mod_wsgi utilisé pour s'interfacer avec notre application Django et pip, le gestionnaire de packages Python qui peut être utilisé pour télécharger nos outils liés à Python , les paquets pour Mysql et git pour import notre projet Django sur notre Debian machine virtuelle :

- apt-get update
- apt-get install git
- apt-get install python3-pip apache2 libapache2-mod-wsgi-py3
- apt-get install default-mysql-server
- apt-get install default-mysql-client
- apt-get install default-libmysqlclient-dev
- apt-get install default-libmysqlclient-mariadb

Une fois les paquets installés nous pouvons lancer MariaDB en utilisant la commande { Mysql }.

Une fois MariaDB lancer il faut créer un utilisateur avec un mot de passe grâce à la commande :

{ CREATE USER “SAE23”@”localhost” IDENTIFIED BY “toto”; }
Après avoir créé l'utilisateur nous devons lui donner tous les privilèges grâce à la commande :

{ GRANT ALL PRIVILEGES ON *.* TO “SAE23”@”localhost”; }
Pour terminer nous enregistrons :

{ FLUSH PRIVILEGES; }

Puis nous pouvons passer à la configuration de notre fichier settings.py dans nottre project django :

```
ALLOWED_HOSTS = ['*']  # Permet de laisser accéder tous les hôtes

DATABASES = {
    'default': {
        'ENGINE' : 'django.db.backends.mysql'
        'NAME' : 'SAE23'  # nom de notre database
        'USER' : 'SAE23'  # nom d'utilisateur
        'PASSWORD' : 'toto' # mot de passe de l'utilisateur
        'HOST' : 'localhost' # local
        'PORT' : '3306'
    }
}
```

(Config settings pour le serveur web)
STATIC_ROOT = os.path.join(BASE_DIR, 'static/') #Django peut collecter et sortir tous les actifs statiques dans un répertoire connu afin que le serveur Web puisse les servir directement.

Configurer un environnement virtuel Python :

La première étape est de créer un environnement virtuel Python afin que notre projet Django soit séparé des outils du système et de tout autre projet Python sur lequel nous pourrions travailler. Nous devons installer la commande virtualenv pour créer ces environnements. Nous pouvons obtenir ce paquet en utilisant pip.

{ pip3 install virtualenv }

Ensuite je crée un dossier à l'aide de la commande { mkdir }
puis j'utilise la commande pour cloner mon projet dans le dossier que j'ai créé :

{ git clone https://github.com/JLKayser/SAE23.git }

Une fois dans mon dossier nommée django-app j'utilise la commande qui permet de créer l'environnement :

{ virtualenv django-appenv }

Puis j'utilise la commande qui installera une version locale de Python et une version locale de pip. Nous pouvons l'utiliser pour installer et configurer un environnement Python isolé pour notre projet :

{ source /home/toto/django-app/django-appenv/bin/activate }

Config Apache2 + Config Django :

J'installe tous les paquets qui vont me permettre d'utiliser mon projet :

- pip3 install django
- pip3 install fpdf
- pip3 install mysqlclient

Pour commencer on créer un utilisateur administratif avec la commande :

{ python3 manage.py createsuperuser }

Nous pouvons collecter tout le contenu statique dans l'emplacement du répertoire que nous avons défini avec STATIC_ROOT en tapant :

{ python3 manage.py collectstatic }

Nous devrons confirmer l'opération. Comme prévu, les fichiers statiques seront placés dans un répertoire appelé static dans le répertoire de votre projet.

Nous devrons ajuster les paramètres de notre pare-feu pour autoriser le trafic vers notre serveur de développement Django, que nous exécuterons sur le port 8000.

Si vous utilisez un pare-feu ufw, vous pouvez autoriser le trafic vers le port 8000 en tapant :

{ ufw allow 8000 }
{ iptables -I INPUT -p tcp --dport 8000 -j ACCEPT }

Puis nous pouvons démarrer notre site sur le port 8000 grâce à la commande :

{ python3 manage.py runserver 0.0.0.0:8000 }

Puis on quitte l'environnement virtuelle pour configurer apache2 à l'aide de la commande :

{ deactivate }

Maintenant que notre projet Django fonctionne, nous pouvons configurer Apache en tant que frontal. Les connexions client qu'il reçoit seront traduites au format WSGI que l'application Django attend en utilisant le module mod_wsgi. Cela aurait dû être automatiquement activé lors de l'installation précédente.

Pour configurer le pass WSGI, nous devrons modifier le fichier d'hôte virtuel par défaut :

{ nano /etc/apache2/sites-available/000-default.conf }

```
<VirtualHost *:80>

    Alias /static /home/toto/django-app/static
    <Directory /home/toto/django-app/static>
        Require all granted
    </Directory>

    <Directory /home/toto/django-app/Drive_SAE23>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    WSGIDaemonProcess django-app
python-home=/home/toto/django-app/django-appenv
python-path=/home/toto/django-app
    WSGIProcessGroup django-app
```

WSGIScriptAlias / /home/toto/django-app/Drive_SAE23/wsgi.py

</VirtualHost>

Récapitulation de certains problèmes d'autorisations :

In order to write to the file, we also need to give the Apache group ownership over the database's parent directory :

{ chown :www-data /home/toto/django-app }

Nous devons à nouveau nous ajuster à travers notre pare-feu. Nous n'avons plus besoin d'ouvrir le port 8000 puisque nous utilisons un proxy via Apache, nous pouvons donc supprimer cette règle. Nous pouvons ensuite ajouter une exception pour autoriser le trafic vers le processus Apache :

{ ufw delete allow 8000 }

Nous utilisons iptables donc nous devons utiliser c'est règles :

{ iptables -D INPUT -p tcp --dport 8000 -j ACCEPT }
{ iptables -I INPUT -p tcp --dport 80 -j ACCEPT }

Puis nous vérifions nos fichiers Apache pour vous assurer que nous n'avons pas fait d'erreurs de syntaxe :

{ apache2ctl configtest }

Une fois ces étapes terminées, nous sommes prêts à redémarrer le service Apache pour implémenter les modifications que vous avez apportées. Redémarrez Apache en tapant :

{ systemctl restart apache2 }

Puis notre site est fonctionnel !!!