# Documentation DOCKER

## C’est quoi docker ?

Docker est un logiciel open source. Il permet de virtualiser dans des conteneurs. On va pouvoir concevoir, tester et déployer rapidement beaucoup d’application.

## Une image ? C’est quoi ?

Une image dans docker est un package qui comprend tout le nécessaire pour l’exécution d’une application. L’image peut être déployé sur autant de container que l’on veut tant que la machine physique à assez de ressource pour tout supporter.

## Mais du coup un conteneur c’est quoi ?

Un conteneur est une machine virtuelle sans noyaux. C’est dans un conteneur que les images installées vont être exécuter. Il y aura le strict nécessaire et les dépendances pour faire tourner l’application.

## Les commandes :

Pour utiliser les commande « docker » il faudra aller dans un terminal PowerShell en mode administrateur.

Au début de chaque commande il faudra mettre « docker » afin que powershell sache avec qu’elle processus exécuter la commande après.

### Pour créer un container :

« docker container run –publish [PORTHOST]:[PORTDOCKER] [IMAGE] –name [NAMECONTAINER] »

Une image contenant texte

Description générée automatiquement

Commande pour créer un container

Port dédier qui sera utilisé par l’application.

Port dédier qui sera utilisé par l’application.

Nom de l’image (généralement le nom de l’application)

Option permettant d’exécuter la commande sans bloquer le CLI

« docker start [IDCONTAINER] » : Démarre un conteneur

« docker stop [IDCONTAINER] » : éteint un conteneur

### Commande information conteneur :

« docker container logs [NAMECONTAINER] » : Permet de voir l’activité du conteneur

« docker container inspect [NAMECONTAINER] » : Nous donne des informations sur d’un conteneur.

« docker stats –all » : Permet de voir

Une fois le conteneur créer, avec la commande « ps » il est possible de voir les différents conteneurs existants avec leur id, nom, port.

« docker ps » : Permet de voir les containers ayant le status « up » donc allumé.

« docker ps -a » : Permet de voir tout les containers existant de connaitre leur statuts

« docker images » : Permet de voir toute les images présente localement

Une image contenant texte, batterie, capture d’écran

Description générée automatiquement

Pour supprimer une image ou une machine :

« rm [IDCONTAINER] » : Supprime le conteneur voulu

« rmi [IDIMAGE] » : Supprime une image installé localement. / ! \ Supprimer avant les conteneurs ayant l’image installé. / ! \

Une image contenant texte

Description générée automatiquement

PING entre deux machines linux ubuntu :

Tout d’abord, il faut créer deux machines virtuelle linux Ubuntu. Etant donné que les machines non aucun package installer, il faut aller chercher le paquage permettant de connaitre la configuration IP de la machine et un package pour avoir la fonctionnalité ping.

Les deux Packages à installer :

Apt-get install net-tools

Apt-get install iputils-ping

### Pour connaitre l’adresse ip de chaque conteneur :

* Sur le conteneur directement avec la commande : « ifconfig »
* Sur docker avec la commande : « docker container inspect [IDCONTAINER] »

Voici le résultat :

Une image contenant texte

Description générée automatiquement

# Dockerfile

Un docker file est en quelque sorte un fichier de configuration d’une image, elle va prédéfinir ce que dois contenir un conteneur utilisant cette image, les éléments d’installation par défaut.

Le DockerFile doit être composé de commande. Les instructions sont lues à la suite.

## Les commandes :

« FROM [IMAGE] » : permet de télécharger une image source

« RUN [INSTRUCTION] » : Permet d’exécuter des commandes dans le conteneur

« ADD [PATHFOLDERHOST] [PATHFOLDERDEST] » : Envoi des fichiers de la machine hôte dans le conteneur à un chemin précis.

« WORKDIR [PATH] » : Défini le répertoire de travail du conteneur

« VOLUME [PATH] » : définir un volume

« ENV [VARIABLE] » : Permet de définir une variable d’environnement

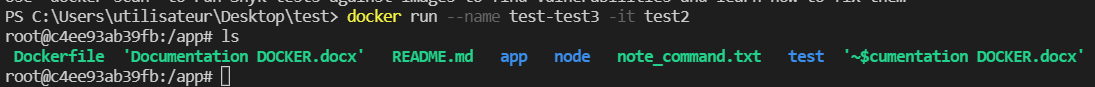
« ARG [VARIABLE] » : Permet de définir une variable avant le FROM

Pour créer une image à partir d’un fichier docker file :

« docker build -t [NAMEIMAGE] [PATHDOCKERFILE] »

Une image contenant texte

Description générée automatiquement



Grace à la méthode ADD nous pouvons bien voir que les documents présents dans mon dossier se retrouve dans le dossier de mon conteneur

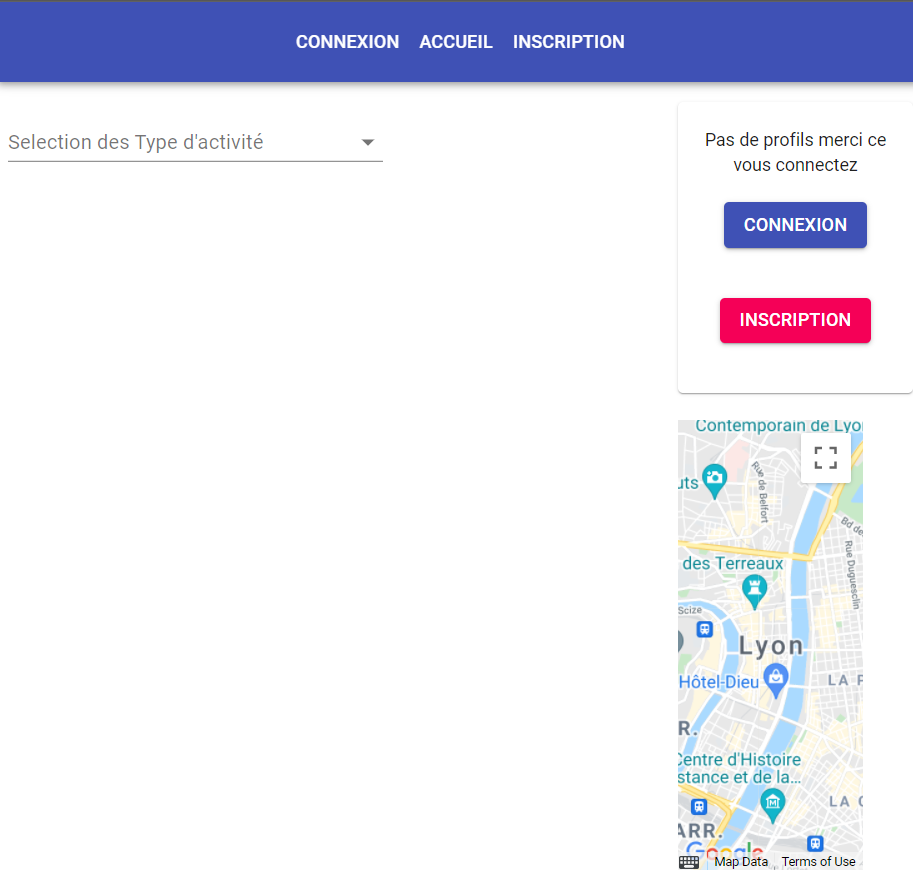
Une image contenant texte, moniteur, écran, capture d’écran

Description générée automatiquement

## Déployer une application React grâce à une image

Une image contenant texte

Description générée automatiquement



Afin de tester cette image veuillez-vous rendre sur ce lien :

<https://hub.docker.com/repository/docker/guiguiso/meetme-app>

Malheureusement, il est possible que l’image ne tourne pas directement. Il faut alors entrer sur le CLI du conteneur et mettre cette commande :

« Service nginx start »

Ne pas oublier également d’attribuer au conteneur le port 80.

Les commandes sont déjà indiquées dans le « Overview » de l’image sur docker hub.

## Envoyer des images sur Docker Hub

Pour envoyer des images créer ou personnalisé par soi-même en ligne, il existe une solution qui se nomme « Docker hub » elle regroupe des millions d’images officiel ou personnalisé par d’autre utilisateur de docker.

Lorsque que l’image est créée, il faut s’assurer que l’application docker sur votre machine est bien authentifié avec votre compte. Si vous n’avez pas de compte, veuillez vous en créer un en suivant ce lien : <https://hub.docker.com/signup>

Une fois le compte créer connectez-vous sur l’application « Docker Desktop »

Une image contenant texte, capture d’écran, noir, écran

Description générée automatiquement

Une fois connectez, ouvrer une fenêtre PowerShell en administrateur et entrer la première commande :

« Docker tag [NAMEIMAGE]:[TAGVERSION] [IDDOCKER]/[NAMEIMAGE]:[TAGVERSION] » :

Cette commande permet d’initialiser une image à l’envoi

Et ensuite, il faut entrer cette deuxième commande :

« Docker push [IDDOCKER]/[NAMEIMAGE]:[TAGVERSION] » : Cette commande permet d’envoyer l’image initialisé sur Docker hub.

Une fois la deuxième commande effectuée, aller sur ce lien : [https://hub.docker.com/u/[IDDOCKER](https://hub.docker.com/u/%5bIDDOCKER)], vous devriez voir votre image désormais disponible au téléchargement.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

# Docker-Compose

## Docker compose c’est quoi ?

Il permet de créer plusieurs conteneurs qui communique entre eux grâce à un fichier de configuration qui se nomme « dockercompose.yml ». Une fois la syntaxe maitrisée, cet outil simplifie le déploiement de multiple environnement.

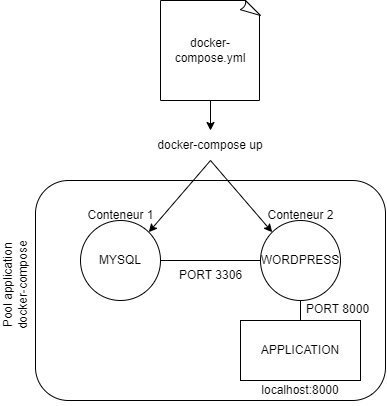
Comme DockerFile, docker compose doit être composé de commande. Les instructions sont lues à la suite.

Il est décomposé en plusieurs partie.

Dans la première partie, il faut mettre la version docker-compose que l’on veut. Plusieurs versions son disponible, il faut faire attention, certaine version ne prenne pas en charge certaine commande dans le fichier de configuration.

La seconde, les services. C’est dans cette partie que nous allons pouvoir mettre nos images avec leur spécification afin de créer les conteneurs déjà configuré par la suite.

Ce Diagram montre comment fonctionne docker-compose avec deux logiciel, Mysql et WordPress :



## Hiérarchie type des commandes d’un fichier docker-compose.yml :

Version : [VERSIONDOCKERCOMPOSE]

Services :

[NAMECONTAINER1] :

Image : [NAMEIMAGE]

Volumes :

* + - [PATHVOLUME1]
    - [PATHVOLUME2]

Restart : always (Cette option permet au conteneur de démarrer en cas de redémarrage du serveur)

Environnement :

* + - [VARIABLEENV1]
    - [VARIABLEENV2]
    - [VARIABLEENV..]

[NAMECONTAINER2] :

Image : [NAMEIMAGE]

Volumes :

* + - [PATHVOLUME1]
    - [PATHVOLUME2]

Restart : always

Environnement :

* + - [VARIABLEENV1]
    - [VARIABLEENV2]
    - [VARIABLEENV..]

Depends\_on : (Ce conteneur est dépendant d’un autre conteneur)

* + - * [NAMECONTAINER1]

Volumes :

* [PATHVOLUME]

Exemple d’un fichier docker-compose.yaml, il permet de faire fonctionner WordPress qui a besoin d’une base de donnée:

Une image contenant texte

Description générée automatiquement

GROUPE :

**Projet Plex**



Tout d’abord nous partons de la distrib Ubuntu. 

On créer donc un fichier dockerfile qui va nous servir à créer notre image.

On copie l’adresse de téléchargement de l’iso.

Une image contenant texte

Description générée automatiquement

On met à jour la version, on installe wget, et on se place dans le dossier téléchargement pour lancer le téléchargement l’exécutable plex.

On test d’abord nos lignes en local via powershell.   
On a récupéré une image ubuntu, mis à jour, puis tenter d’installer les paquets nécessaires tels que :   
- dkpg  
- wget  
- gnung

Plex n’a pas l’air d’être compatible avec la dernière version d’Ubuntu.  
On teste avec une version antérieure d’Ubuntu, la 16.04. Mais nous avons encore une erreur comme on peut le voir ci-dessous

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Autre test :   
Pour celui-ci nous nous sommes inspirés de ces articles :

<https://fr.linuxcapable.com/how-to-install-plex-media-server-on-ubuntu-20-04/>

<https://www.delftstack.com/fr/howto/linux/how-to-install-plex-media-server-on-ubuntu/>

Depart de l’image ubuntu 20.04.   
On commence tout simplement par mettre à jour : *sudo apt update && sudo apt upgrade -y*

On installe les paquets sudo, gnupg, gnupg2, gnupg1, curl …

*sudo apt install apt-transport-https curl -y*

On crée un fichier de référentiel extrait directement du référentiel Plex. Ce qui garantit que l’installation et la mise à jour sont fait à partir de la source officielle :   
*curl https://downloads.plex.tv/plex-keys/PlexSign.key | sudo apt-key add –*

On importe le référentiel :   
*echo deb https://downloads.plex.tv/repo/deb public main | sudo tee /etc/apt/sources.list.d/plexmediaserver.list*

On installe plex : sudo apt install plexmediaserver  
On souhaite connaitre le status : *systemctl status plexmediaserver*

Et là on a une erreur, comme si le service n’existe pas.

Comme on ne voyait pas d’où pouvait provenir l’erreur, nous avons essayé de faire ces mêmes commandes sur une machine virtuelle ubuntu sur vm ware. Cette version avait déjà tous les paquets de bases d’installés. Ce test nous permettait de savoir si le problème venait des paquets à installer non.

Etant donné que sur cette machine nous n’avons pas rencontré d’erreur, nous en avons conclu que nous n’avions pas les bons paquets.   
Nous avons listé tous les services installés sur cette machine pour comparer avec la nôtre, mais il y’avait plus de 1787 lignes… Nous ne nous sommes donc pas attardés dessus.

Autre test :

Depart de l’image ubuntu 20.04.   
On commence tout simplement par mettre à jour : sudo apt update && sudo apt upgrade -y

On installe les paquets sudo, gnupg, gnupg2, gnupg1, curl …

En essayant également de trouver des commandes qui peuvent permettre d’installer tous les paquets de bases pour ressembler au maximum à notre version sous Vm Ware comme :

* Apt install gdebi
* sudo apt install build-essential

sudo apt install apt-transport-https curl -y

curl https://downloads.plex.tv/plex-keys/PlexSign.key | sudo apt-key add –

echo deb https://downloads.plex.tv/repo/deb public main | sudo tee /etc/apt/sources.list.d/plexmediaserver.list

On installe plex : sudo apt install plexmediaserver

Cependant le résultat est le même, nous bloquons sur la commande : systemctl status plexmediaserver qui nous retourne ce message :

*System has not been booted with systemd as init system (PID 1). Can't operate.*

*Failed to connect to bus: Host is down*

Une image contenant texte

Description générée automatiquement

Autre test :

On décide donc de procéder autrement, on ne va plus s’attarder sur ce systemctl qui ne fonctionne pas.   
On va créer notre dockerfile en précisant un port. On build notre image et si en se connectant à localhost : port un résultat s’affiche alors cela voudra dire que notre conteneur est bien en route.   
On décide dans un premier temps de partir d’une version antérieure d’Ubuntu. Une version 16.04 qui est une version stable.

Voici le début de notre dockerfile. On crée une première variable qui contient la version de Plex que nous souhaitons installer. On en crée un deuxième qui contient le lien de téléchargement de l’exécutable.

Ensuite on met à jour les paquets et on installe le paquet wget qui va nous servir pour télécharger plex grâce au lien qu’on a récupéré précédemment, comme inscrit à la ligne 16.

Une image contenant texte

Description générée automatiquement

Pour la suite, à la base nous voulions passer par l’instruction USER Plex, qui veut tout simplement dire d’utiliser l’utilisateur plex.   
Après un test voici l’erreur qui est retournée :



Suite à cette erreur, nous avons trouvé une autre instruction en passant par la création d’un groupe.   
Voici notre source : Une image contenant texte

Description générée automatiquement  
  
Adapté à notre projet elle donne :   
Une image contenant texte

Description générée automatiquement

On s’attaque ensuite à plex directement. Après de nombreuses recherches, nous avons récupéré quelques variables d’environnement qui semblent obligatoire au bon fonctionnement. Voici quelques pages qui nous ont aidé :

**Sources variables d’environnement :**

Une image contenant texte

Description générée automatiquement  
<https://qastack.fr/ubuntu/600026/problems-starting-plexmediaserver-on-kubuntu-15-04>

Une image contenant texte

Description générée automatiquement  
<https://hub.docker.com/layers/tianon/plex/1.18.4.2171-ac2afe5f8-media-server/images/sha256-253b6886997289b77bf90f5cd08a9a1196b8351edb520a0763c4fccef96d701a?context=explore>

La difficulté s’est magnifestée par le grand nombre de variable. Nous ne savions pas lesquelles garder.

Après quelques tests, voici notre sélection.

Par exemple voici une erreur en testant le chemin /usr/lib/plexmediaserver/Plex Media Server. On obtient une erreur car le chemin n’existe pas encore, c’est pour cela qu’il lui faut sa variable d’environnement.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Celles-ci sont présentes pour le bon fonctionnement du service.   
On voit notamment sur la dernière ligne qu’on ajoute l’arborescence au PATH pour que Plex puisse être exécuter sans poser problème.

On lui attribue un port, puis un volume, et enfin se place par défaut dans le volume qu’on a créée.

Une image contenant texte

Description générée automatiquement

Enfin, on lui donne l’instruction de lancer Plex.

Une image contenant texte

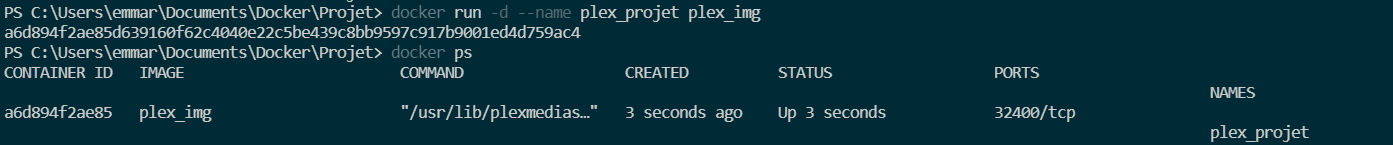
Description générée automatiquement

Ce qui nous donne le résultat suivant :

Une image contenant texte

Description générée automatiquement

On build, notre dockerfile et cela fonctionne.   
On lance notre conteneur :

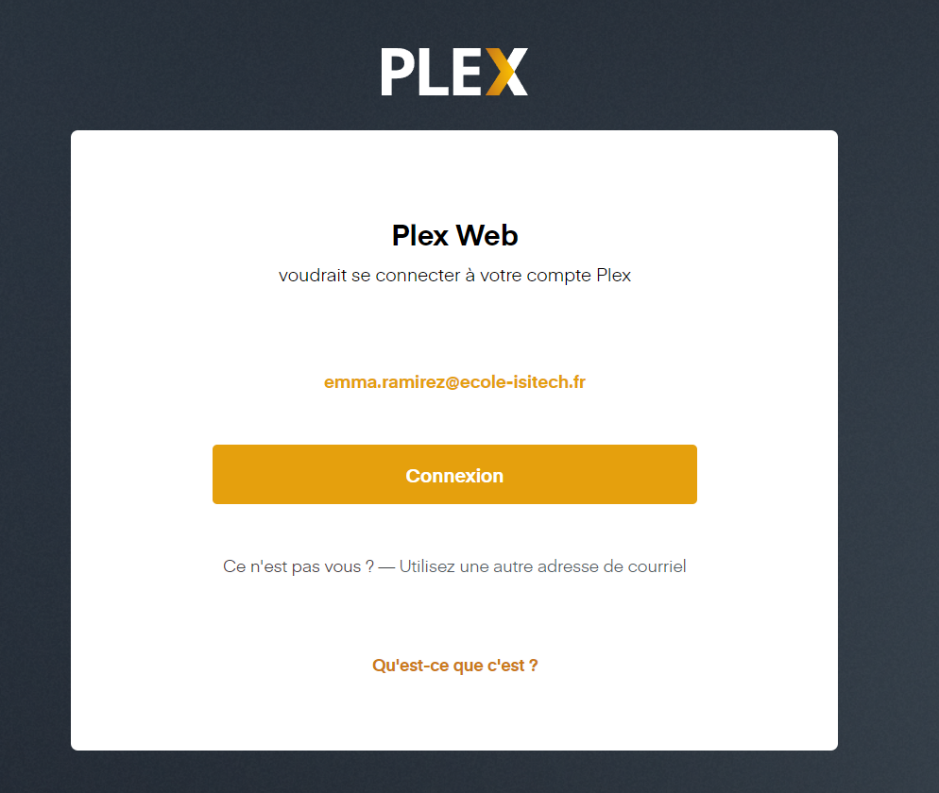
  
  
On tape l’adresse localhost :32400

Une image contenant texte

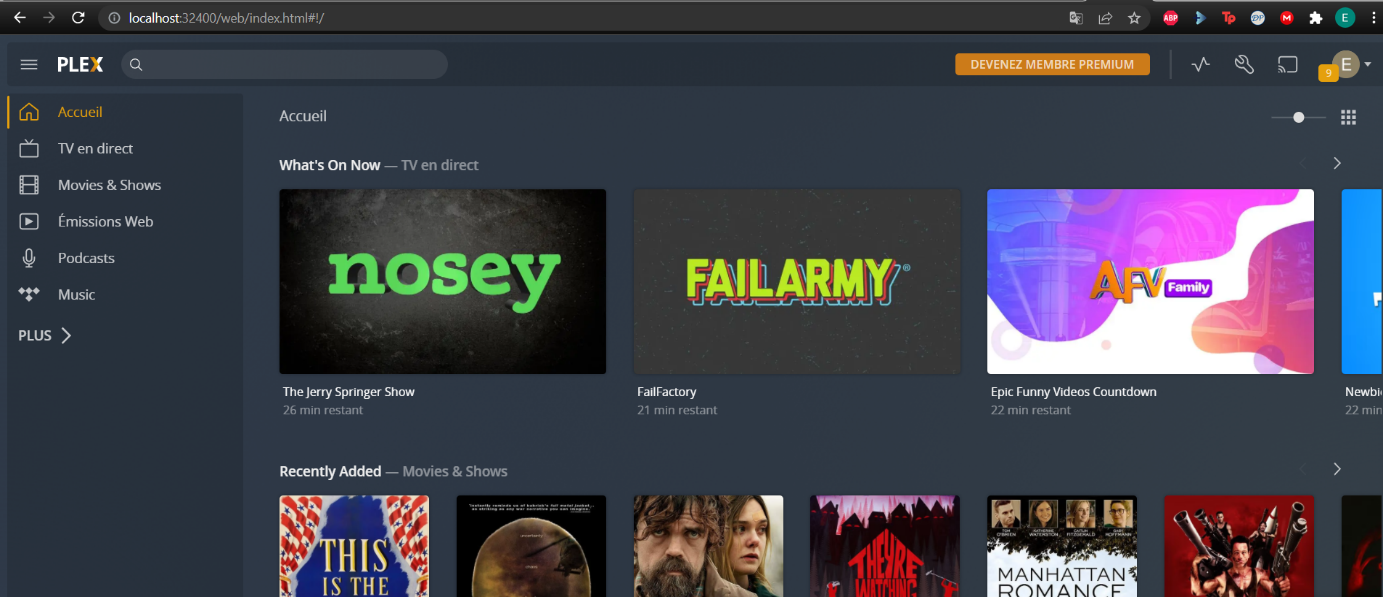
Description générée automatiquement

En recherchant l’erreur sur internet, il fallait en fait rajouter localhost :32400/**manage**

On arrive bien sur la page Plex.







On tente donc de modifier la version d’Ubuntu par une version plus récente, la 20.04. Celle-ci fonctionne également.

Voici une vision de tous les tests :



  
Une image contenant texte

Description générée automatiquement

**Docker-compose**

Une image contenant texte

Description générée automatiquement

Ici nous sommes partis de l’image plex-img créée précédemment pour lui apporter quelques modifications (sans toucher à l’image en elle-même).  
A la place de n’avoir qu’un seul volume, nous avons décider d’en créer 3 pour séparer les différentes données.   
Un volume pour la configuration, un volume pour les films et un volume pour la télévision.

On tape à nouveau localhost :32400/**manage**Et on tombe bien sur la page Plex.   
On peut observer en créant la bibliothèque nos 3 volumes stipulés dans le docker-compose

Une image contenant texte, moniteur, noir, capture d’écran

Description générée automatiquement