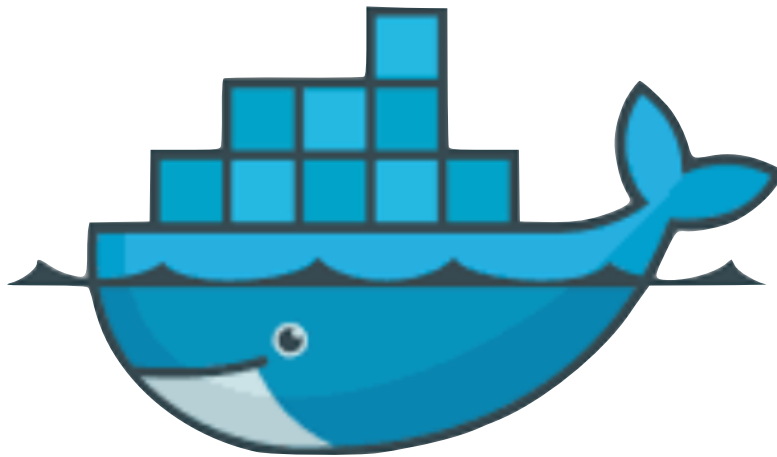


Polytech Tours DII 2022-2023

GÉNÉRATEUR DE DOCKEFILE



Kenza AIT OUZZOU

Ayimane BARA

Professeur référent: Julien GIDEL

Table des matières

Table des matières	2
Introduction	3
Organisation et construction du projet	4
Docker	5
A quoi sert Docker ?	5
Les grands principes des conteneurs Docker	5
Comment fonctionne Docker	6
L'écosystème Docker	7
Docker Architecture	7
Objets Docker	8
Limitation de docker	8
Création d'une image avec Docker	8
Une interface: Django	10
Choix d'un logiciel: pourquoi Django ?	10
Comment ça fonctionne ?	10
Les avantages d'un environnement Django	11
Architecture d'un projet Django	12
Résultats et Solution	13
Conclusion	16
Bibliographie	17

Introduction

Aujourd'hui, docker est une énorme machine qui est utilisée par des millions de personnes à travers le monde. Principalement, Docker est utilisé afin de fournir un environnement commun à plusieurs personnes au travers d'images construites via des DockerFiles.

Les DockerFiles permettent une personnalisation des conteneurs par le lancement de plusieurs commandes cela évite d'avoir à les répéter chaque fois que l'on veut créer un nouveau conteneur. En plus de cette efficacité à ne pas réinventer la roue à chaque build, les fichiers Docker diffèrent très peu d'une organisation à une autre au point que certains proposent des fichiers Docker de base en fonction des projets sur Git Hub mais peuvent contenir un très grand nombre de commandes possibles par forcément évidentes pour une personne qui commencerait.

Créer un générateur de DockerFile permettrait de simplifier ce processus de création en proposant des DockerFile de base, modifiables par une interface. L'idée de ce générateur serait finalement de permettre de faciliter la création de DockerFile en permettant d'y ajouter un certain nombre d'options et de commandes qui peuvent revenir souvent, de manière simple et compréhensible par n'importe qui.

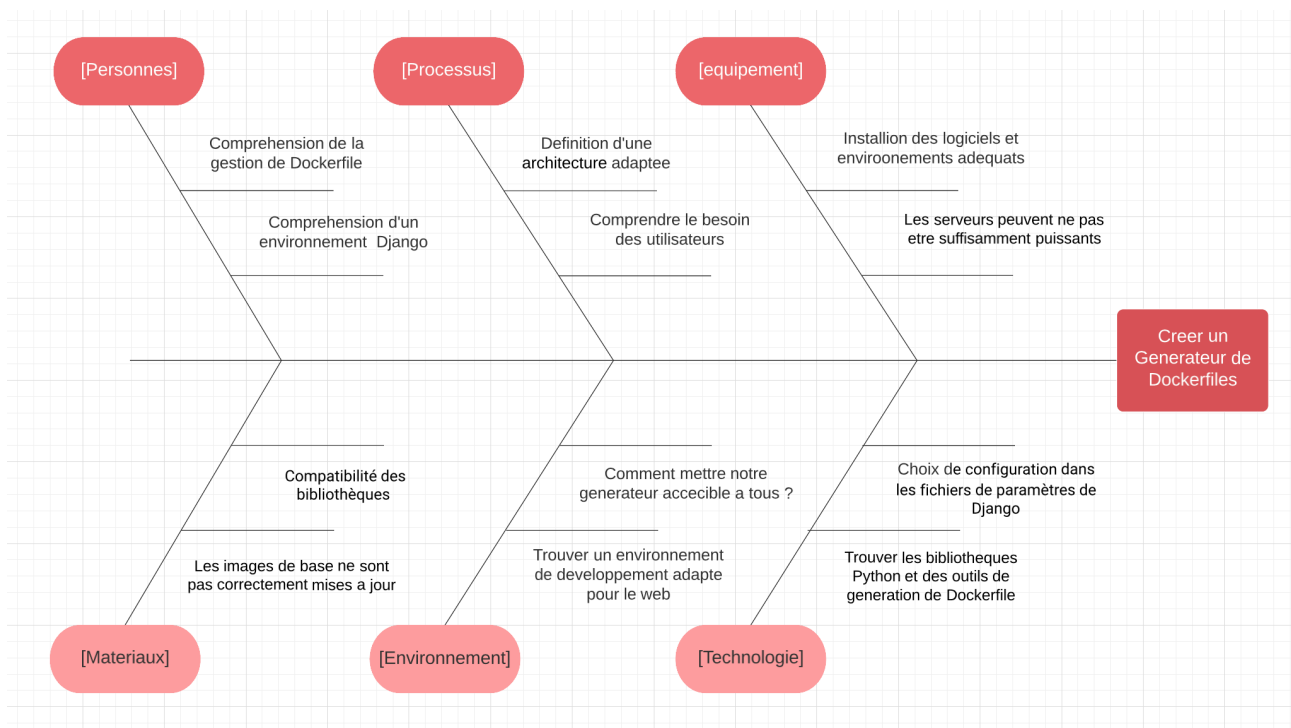
Dans le développement de notre projet, nous expliquerons en plusieurs détails ce que nous avons retenue de l'application Docker et de son utilisation. Nous traiterons également de Django -un framework python- que nous avons utilisés pour développer notre interface web, nous justifierons notre choix de logiciels et les étapes de développement pour que notre modèle fonctionne. Finalement nous conclurons sur le résultat final de notre projet, les difficultés rencontrées et ce que nous en avons retenue.

Organisation et construction du projet

Il nous a été demandé pour la réalisation de notre projet, de créer une interface qui permettrait à l'utilisateur de générer un DockerFile de façon simplifiée. Cette interface permettrait à une personne qui n'est pas familière avec le logiciel de créer facilement un Docker File.

Pour rendre cette interface compréhensible de tous est facile d'utilisation tout en respectant sa fonctionnalité principale: créer un docker File sans problème de configuration, nous avons réfléchi au moyen le plus efficace de créer cette interface.

Afin de prendre en compte les fonctionnalités que nous souhaitions développer, des techniques que nous voulions utiliser pour les implémenter et les contraintes techniques du projet, nous avons réalisé un diagramme d'Ishikawa.



Au vu des problèmes que nous pouvions rencontrer telles que le choix des bibliothèques ou la compatibilité avec Docker, nous avons opté pour le développement de notre interface avec le logiciel Django. En effet, étant en Framework Python, celui-ci nous nous fournit un large panel de librairie et bibliothèque. De plus Django est connu pour être très simple d'utilisation pour des gens qui commencent le développement Web et possède une architecture très simple à maintenir.

Nous avons ensuite réparti les tâches pour avancer au mieux dans le projet. Aymane devait s'approprier Docker et construire les containers tandis que Kenza devait créer l'interface Django.

Docker

A quoi sert Docker ?

Tout d'abord, Docker est un outil qui peut emballer une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel environnement. Entre autre, c'est une plate-forme de conteneurisation qui regroupe l'application et ses dépendances dans un conteneur afin que l'application fonctionne de manière transparente dans n'importe quel environnement, qu'il s'agisse de développement ou de production.

Il présente plusieurs avantages tels que :

- L'utilisation plus efficace des ressources système
Il permet une utilisation plus efficace des ressources d'un système tout en permettant les mêmes avantages qu'une machine virtuelle. Les applications conteneurisées utilisent moins de ressources que les machines virtuelles car elles utilisent le même noyau Linux.
- La reproductibilité
Un conteneur docker est identique quel que soit le système : la bonne version de chaque dépendance est garantie en installation. Chaque membre d'une équipe est certain d'avoir le ou les applications qui fonctionnent identiquement sur des environnements différents.
- L'isolation
Les dépendances ou les configurations d'un conteneur n'affecteront aucune dépendance ou configuration des autres conteneurs ou de la machine hôte.
- La mise à jour et tests
Avec Docker, la mise à jour d'un environnement dans le cloud, même sur de nombreux serveurs est très simple.

Les grands principes des conteneurs Docker

Docker permet de build, de lancer et de partager des applications en utilisant des conteneurs.

Un conteneur permet d'isoler une application pour assurer une parfaite répliquabilité des environnements c'est-à-dire : il permet de s'assurer qu'on exécute l'application sur le même environnement quel que soit la machine physique et l'OS utilisé.

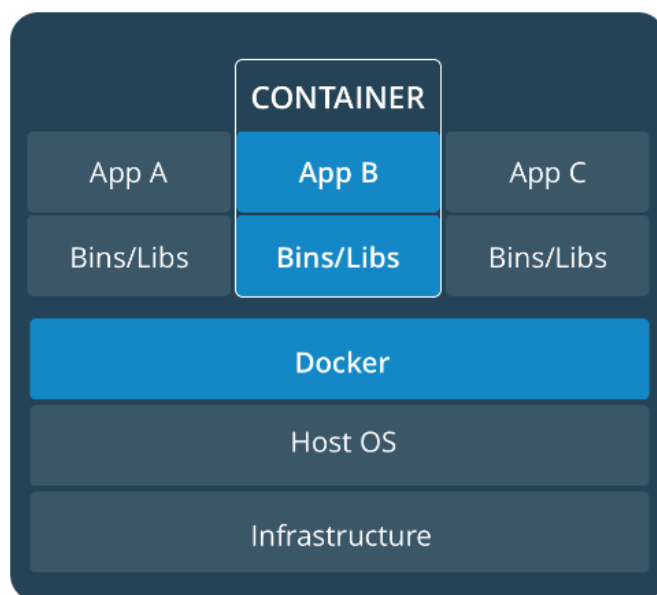
Les principes des conteneurs Docker sont les suivants :

- Flexibilité : n'importe quelle application, même les plus complexes, peuvent être containerisée.
- Légèreté : du fait qu'ils partagent les ressources et le noyau Linux de l'OS de la machine hôte.
- Portabilité : utilisables sur n'importe quel environnement.
- Faible couplage : les conteneurs sont autonomes car ils sont isolés, facile à supprimer et modifiable.
- Scalables : déploiement automatiser de nouveau conteneur (auto-scaling)
- Sécurisés : les contraintes et l'isolation par défaut des conteneurs apporte une grande sécurité des environnements déployés.

Comment fonctionne Docker

Docker utilise le noyau Linux et l'environnement GNU du système. Sur ce même système, il va lancer un conteneur qui est un processus avec des fonctionnalités d'encapsulation lui permettant d'être isolé de l'hôte et des autres conteneurs.

Chaque conteneur a son propre système de fichiers isolé qui est fourni par une image Docker. Ce système de fichiers contient le code, les binaires, les fichiers exécutables et toutes les dépendances requises pour faire fonctionner une application.



Principe de conteneurisation Docker

En clair Docker partage le système d'exploitation entre tous les conteneurs qu'il lance, qui sont des simples processus légers.

L'écosystème Docker

Le Docker Engine ou moteur Docker est une application sur un modèle client / serveur. Il est composé de 3 parties principales :

- Le **serveur docker** qui est exécuté en permanence. C'est lui qui va créer et gérer tous les objets Docker (les images, les conteneurs, les réseaux et les volumes).
- L'**API REST** qui spécifie les interfaces que les programmes peuvent utiliser pour communiquer avec le service docker.
- Le **client** (CLI command line interface) qui permet d'exécuter des commandes Docker. Le CLI traduit les commandes entrées par l'utilisateur en requêtes pour l'API REST.

Docker Architecture

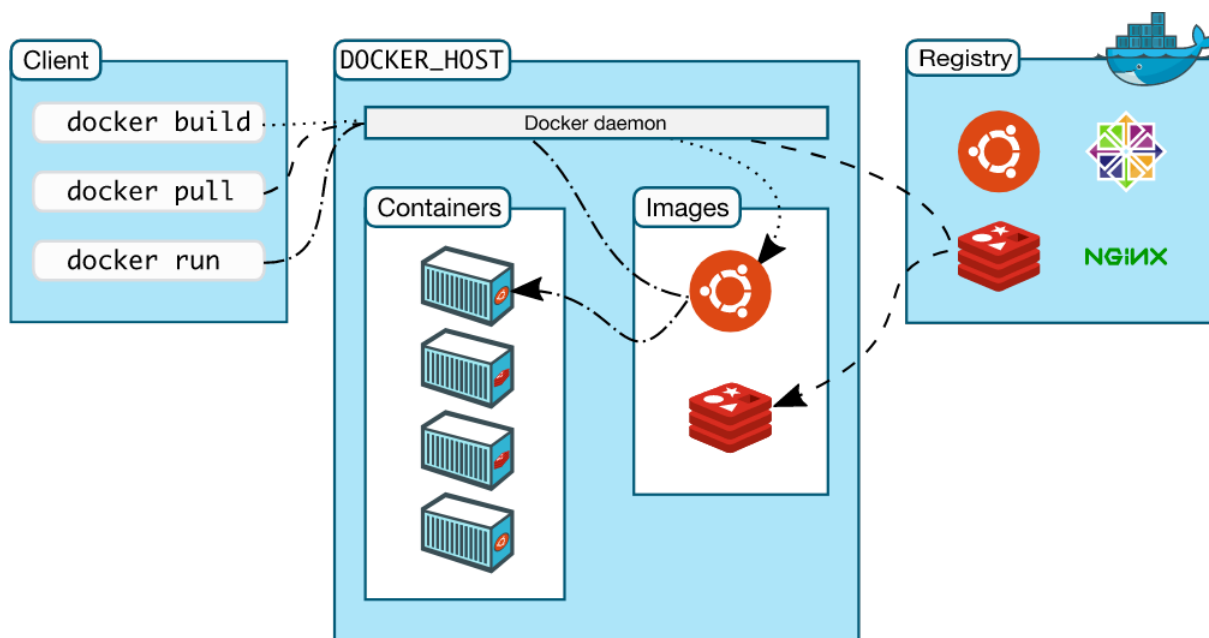


Illustration de la relation entre le CLI, le serveur docker et le Docker Hub (bibliothèque d'images/registry)

Objets Docker

1. Les images

Une image Docker est un schéma en lecture seule qui contient les instructions pour créer un conteneur Docker.

Généralement ou plus souvent, une image est elle-même basée sur une autre image avec des configurations spécifiques. Pour créer une image, il faut utiliser un fichier spécifique appelé DockerFile qui a une syntaxe particulière permettant de définir les étapes nécessaires à la création de l'image. Chaque instruction dans un DockerFile permet de créer une couche dans l'image.

2. Les conteneurs

Un conteneur est une instance d'une image en cours d'exécution qui peut prendre des options de configuration passées lors du lancement. Pour créer, démarrer, arrêter, déplacer ou supprimer un conteneur il faut utiliser le CLI. Un conteneur par défaut est isolé des autres conteneurs et de la machine hôte et il est possible de le connecter à un ou plusieurs réseaux. Il est aussi possible de lui attacher un ou plusieurs volumes de stockage.

3. Les volumes

Les volumes permettant aux conteneurs de stocker des données et sont initialisés lors de la création d'un conteneur. Ils permettent aussi de persister et de partager des données d'un conteneur. Les volumes sont stockés en dehors du système. Enfin ils permettent de conserver des données même si un conteneur est supprimé, mis à jour ou rechargé.

Limitation de docker

Docker est très efficace pour la gestion de conteneurs uniques. Cependant, à mesure qu'augmente le nombre de conteneurs et d'applications conteneurisées, la gestion et l'orchestration se complexifient. Au final, on est amené à regrouper plusieurs conteneurs pour assurer la distribution des services vers tous nos conteneurs.

Création d'une image avec Docker

Un Dockerfile contient les instructions permettant à Docker de construire (build) une image automatiquement. Il contient toutes les commandes nécessaires à la construction de l'image. Il est aussi constitué de 3 parties : Une image de base : c'est l'image à partir de laquelle on effectuera les modifications pour créer notre image. (On ne pourra pas créer un système d'exploitation tout seul). Les instructions : ce sont des commandes Docker

permettant de détailler toutes les modifications à apporter à l'image de base pour aboutir à votre image finale. L'action : C'est la commande qui est exécutée par défaut lorsque l'image sera lancée dans un conteneur.

Pour créer une image on va d'abord créer notre docker file. Ensuite pour créer une image à partir d'un DockerFile il faut la build en utilisant la commande docker image build CHEMIN.

En supposant qu'on est dans le dossier qui contient le DockerFile on fera :

```
docker image build .
```

Le processus de construction d'une image (build) utilise un DockerFile et un contexte. Le contexte est l'ensemble des fichiers contenus dans le chemin passé à la commande docker image build. Le build est effectué par le serveur docker. Il est judicieux de créer le DockerFile dans le dossier de notre application ou projet et non dans le répertoire racine du système.

```
docker build -t node:latest .
```

Une fois l'image créée on va devoir lui donner un nom (Taguer l'image ou donner un tag)

Ensuite après avoir build et tagué l'image, nous pouvons l'afficher en faisant

```
docker image ls
```

[Copier](#)

Enfin on peut maintenant démarrer un conteneur à partir de l'image en faisant

```
docker run node
```

[Copier](#)

Une interface: Django

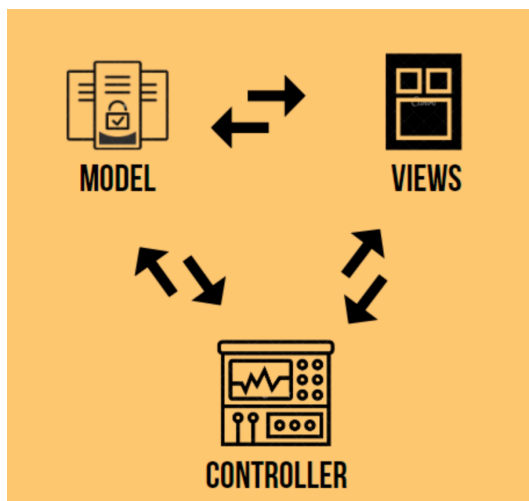
Choix d'un logiciel: pourquoi Django ?

Pour la création de notre générateur, il nous a été demandé de créer une interface qui permettrait de sélectionner les images et leurs paramètres pour générer notre VM Docker.

Nous nous sommes également posé la question de l'accessibilité de notre application. Nous souhaitons quelque chose facile d'accès pour tout le monde, l'idée étant de faciliter l'utilisation de Docker pour une personne novice.

Comment ça fonctionne ?

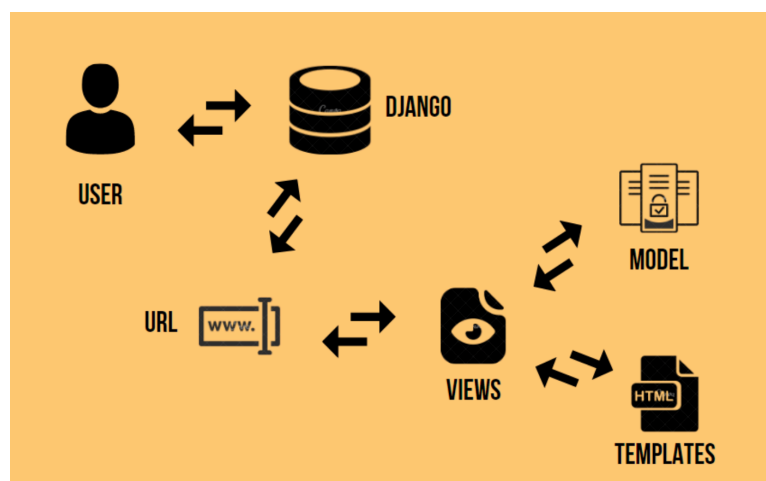
Django est un framework web qui permet de développer sur Python des sites et applications web dynamiques de façon simple et rapide.



Model MVC

Similaire au modèle MVC, Model/View/Controller, le modèle MVT Model/View/Template, choisie par Django, se caractérise par l'utilisation de Template. Le principe reste toutefois identique le modèle (Model) qui contient les données est distinct du contrôleur (Controller) qui traite les données, tandis que la vue (View) contient les données à afficher.

Tout comme les vues dans le model MVC, Django utilise des modèles dans son framework. Les modèles sont entièrement responsables de l'ensemble de l'interface utilisateur. Il prend en charge toutes les parties statiques de la page Web ainsi que le HTML, perçu par les utilisateurs.



Model MVT

Dans Django, les vues font le lien entre les données de modèle et les templates. Tout comme le contrôleur dans MVC, les vues dans Django MVT sont responsables de la gestion de toute la logique métier derrière l'application Web. Ce fonctionnement fait de Django un framework complet, clair et facile à prendre en main. Une fois créés, les applications et sites web codés sous Django sont robustes et leur maintenance est plus simple.

Les avantages d'un environnement Django

Django a été conçu pour le développement de serveur Web. C'est un Framework très haut niveau, ce qui facilite grandement son utilisation. Il est notamment doté de plusieurs outils qui lui sont spécifiques et qui lui permettent d'être très performant.

- **Ses templates**

Un langage de template flexible qui permet de générer du HTML, XML ou tout autre format texte.

- **Possède son propre contrôleur**

Il est fourni sous la forme d'un "remapping" d'URL à base d'expressions rationnelles.

- **Possède une API html**

d'accès aux données est automatiquement générée par le framework compatible CRUD.

- **Les bibliothèques**

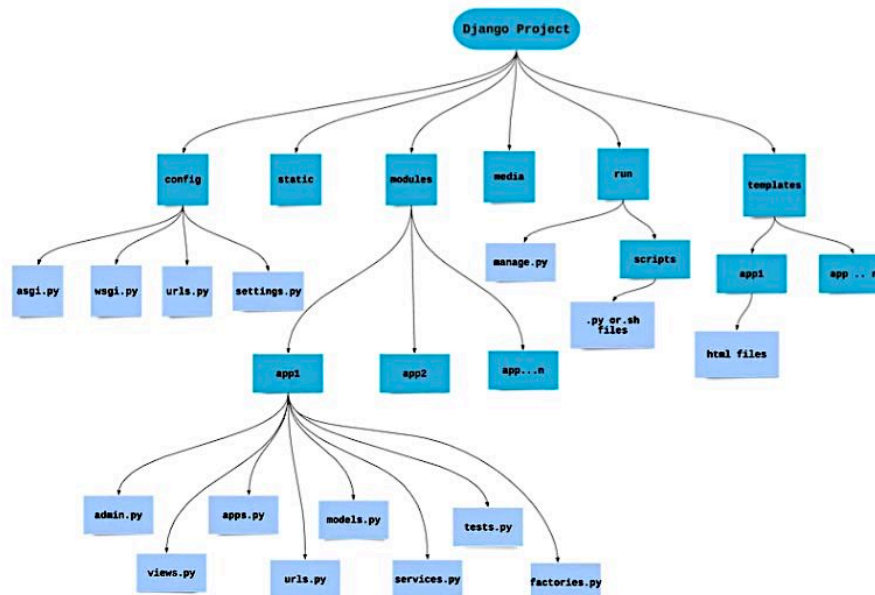
Django à accès aux bibliothèques python qui permettent de gérer des tâches courantes mais aussi plus complexes.

- **L'ORM**

Le mappeur objet-relationnel de Django extrait des données présentes dans des bases de données et les transfère sous formes d'objets. Il permet de gérer la structure des modèles et à construire très rapidement ses prototypes.

- **La sécurisation**

Django à développé une architecture permettant d'éviter les failles de sécurité les plus courantes.



Architecture d'un projet Django

Architecture d'un projet Django

En effet un projet Django se caractérise par sa structure bien spécifique. Elle est composée de plusieurs fichiers qui vont interagir entre eux afin de créer un serveur Web.

```
$ django-admin startproject mysite
```

Voici les fichiers de base créé dans un projet Django:

- **manage.py**
Constitue le script qui gère l'ensemble du projet.
- **__init__.py**
C'est un fichier Python vide qui indique à Python que ce dossier doit être considéré comme un package Python.
- **settings.py**
Il contient les paramètres de configuration générale telle que la base de données.
- **urls.py**
contient les définitions des URL et fait le lien avec les vues pour permettre un affichage du site web.
- **asgi.py et wsgi.py**
Ce sont des fichiers qui définissent l'interface entre les applications web Django et le serveur web.

Le projet est ensuite testé grâce à l'environnement Django qui se lance grâce à la commande suivante.

```
$ python manage.py runserver
```

En effet, il faut faire attention à ce que le fichier manager.py soit bien dans la racine du projet car c'est lui qui va permettre de lancer le serveur et de voir comment l'environnement fonctionne.

```
○ cecilia@Cecilia generateDockerfile % python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 26, 2023 - 10:27:36
Django version 4.1.4, using settings 'generateDockerfiles.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Il est également important de souvent mettre à jour les informations qui sont envoyées dans le serveur c'est pour cela quand utilisé très souvent la commande "migrate" pour recharger le nouveau contenu ajouté.

```
$ python manage.py migrate
```

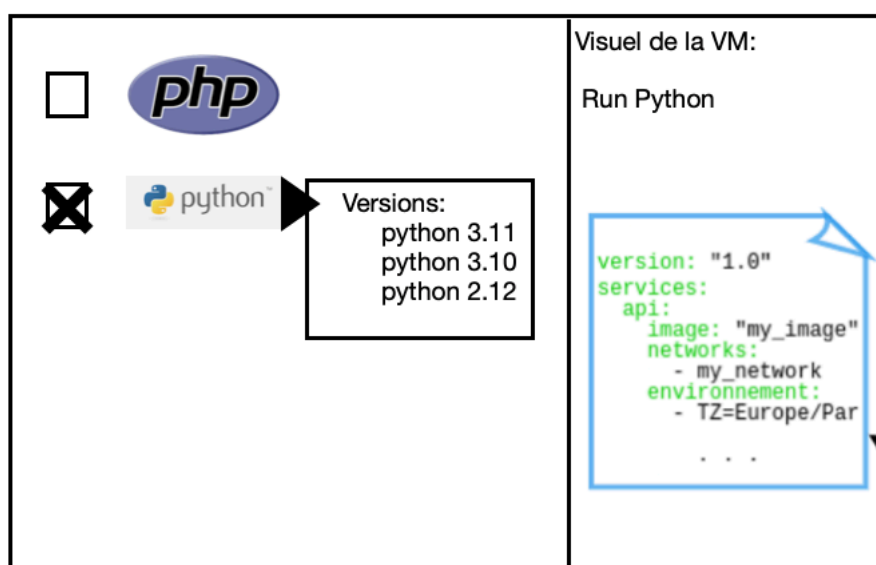
Résultats et Solution

En ce qui concerne l'interface Django, nous voulions quelque chose de simple et très intuitif.

Il nous fallait la possibilité de choisir par l'utilisateur, différentes applications qui figurait dans le docteur fils, après avoir été choisi par l'utilisateur. Nous pensions à ajouter à développer des choix à cocher, avec un défilé roulant, contenant toutes les versions de l'application pour que l'utilisateur fils, définir la version qu'il souhaite de l'application choisie.

Nous voulions une interface assez intelligente pour qu'utilisateur puissent créer une VM docker qui soit fonctionnelle, sans pour autant savoir ce qui s'est passé derrière.

L'interface ressemblait dans nos plans à l'image ci-contre:

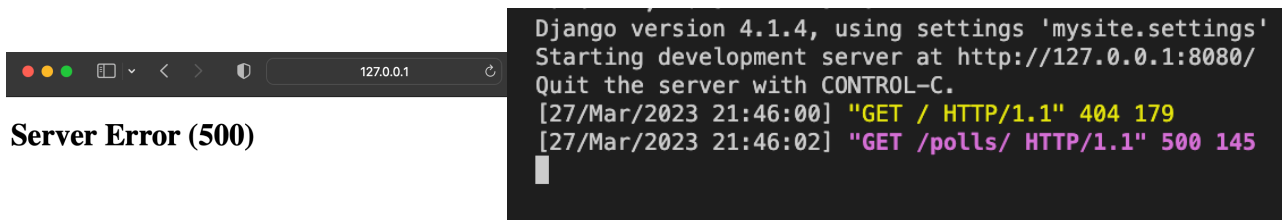


L'idée était de développer l'interface uniquement d'un point de vue visuel, puis de lier les différents onglets coulissants avec les images docker déjà créés et accessible sur leur site pour la rendre fonctionnelle dans un second temps.

Malheureusement, nous avons rencontré plusieurs problèmes lors du développement de l'interface. Nous n'arrivons pas à retrouver nos vues sur le serveur. Il nous était donc impossible de visualiser correctement l'interface. Nous avons généralement une erreur 500 ou 404 qui nous empêchait de visualiser cette interface. Nous avons pu trouver sur Internet que l'erreur 404 est du à un problème du service qui n'arrive pas à fois lancer l'environnement ou son contenu URL. L'erreur 500 est un peu plus complexe elle indique que la requête envoyée par le navigateur n'a pas pu être traitée pour une raison qui n'a pas pu être identifiée. C'est un problème qui affecte le serveur sur lequel le site est hébergé.

Nous avons découvert lors de l'apparition de ses erreurs comment l'environnement interagissait directement avec le développeur. Ce fut très intéressant de voir que l'environnement réagissait directement aux commandes que l'on envoyait au serveur. Lorsque l'on mettait l'adresse et que celle-ci envoyait une trace d'erreur jaune ou rouge selon le code d'erreur.

Nous avons également remarqué qu'elles étaient toujours au nombre de deux. L'erreur 500 était très souvent couplée avec l'erreur 145 tandis que l'erreur 404 était couplée avec l'erreur 179. L'erreur était affichée par le système et c'était à nous d'identifier quelle était la nature de l'erreur pour ensuite la corriger.



En ce qui concerne le design de notre solution, nous avons appris à faire des templates pour avoir des volets coulissants, et des cases à cocher mais nous n'arrivions pas non plus à relier ces cases aux images docker.

Nous avons donc décidé de repartir de zéro dans la création du projet, Il nous fallait refaire le tutoriel de zéro pour mieux comprendre nos erreurs.

Nous avons pu revenir sur une bonne base de code. Le deuxième projet nous permettait d'avoir un compte administrateur, de gérer les différentes views.

De plus, suite au tutoriel accessible sur le site Django, nous pouvions ajouter des questions et les réponses qui corrèlent. Malheureusement, nous n'avons toujours pas pu relier ses views à docker.

Nous avons voulu par manque de temps, changer notre manière de penser l'interface. Nous nous sommes dit qu'il était préférable d'avoir une interface très simple, peut-être pour un utilisateur qui serait plus ce qu'il fait mais qui permettrait avec nos connaissances de créer une VM docker.

La nouvelle interface ressemblait à l'image ci-contre:

The image shows a simple web interface within a rectangular frame. On the left, there is a button labeled 'COPY'. To its right is a text input field with the placeholder text 'ÉCRIVEZ VOTRE COMMAND ICI.....'. In the bottom right corner of the frame, there is a button labeled 'SUBMIT'.

Son principe est simple, grâce à un défilé roulant, n'aurions qu'à proposer les choix de commande telle que RUN, COPY ou encore CMD.

L'utilisateur n'aurait qu'à taper la commande dans une « barre de recherche » et de valider avec un bouton « submit » tu aurais le rôle de régénérer la page. La commande sera alors envoyée et l'utilisateur pourra grâce à celles-ci créer ça VM.

Cette nouvelle interface était plus simple à développer mais, nécessitait de plus solides connaissances de docker par l'utilisateur. Celle-ci peut générer plus facilement des erreurs.

Conclusion

Finalement, créer un générateur de DockerFile consiste à faire interagir des images Docker via une interface Django pour faciliter la création de VM Docker. Notre projet s'est donc articulé en deux phases de la mise en place de Docker à la création de l'interface Django.

Nous avons pu en apprendre plus sur Docker et sa gestion de containers et d'images. Nous avons pu voir son utilité et bénéficier de ses avantages. Toutefois, nous avons beaucoup de mal à prendre en main le logiciel, ce qui nous a fait perdre beaucoup de temps dès le début du projet.

Nous avons donc pris du retard dans le développement de l'interface logiciel développée avec Django. Nous n'avons pas réussi par manque de connaissances sur Django et de temps le développement Web.

Nous avons tout de même pu comprendre comment Django structurait ses projets et comment étaient liés les différents fichiers entre eux. Nous avons eu beaucoup de mal à nous approprier Django pour lier nos views et nous permettre de les afficher les rendre visibles par l'utilisateur.

Notre gestion du temps n'a sans doute pas été la meilleure lors de ce projet. En effet, nous avons mis beaucoup de temps à commencer le développement la phase de planification du projet a été peut-être trop long pour le laps de temps donné. Nous avons donc mal géré notre temps au début sans vraiment prendre en considération le temps que peut prendre une erreur de développement comme ce que nous avons pu rencontrer.

Malgré tout, nous avons eu une bonne communication entre les membres du projet, mais également avec notre tuteur référent. Nous avons rapidement déclaré avoir des problèmes dans le développement de la solution finale ce qui nous a permis de comprendre plus vite d'où le problème survenait.

Créer un générateur de DockerFile reste un projet très intéressant et qui peut s'avérer très utile, notamment dans un parcours industriel comme les nôtres car il est très pratique d'avoir en production, une interface qui permettrait aux nouveaux employés par exemple, de se familiariser plus facilement et plus rapidement avec Docker, une application très prisée des entreprises.

Bibliographie

Mise en place d'un projet docker:

<https://docs.docker.com/>

S'appropriier l'utilisation des images Docker :

<https://www.ionos.fr/digitalguide/serveur/know-how/les-images-docker/>

Mise en place d'un projet Django :

<https://docs.djangoproject.com/fr/4.1/intro>

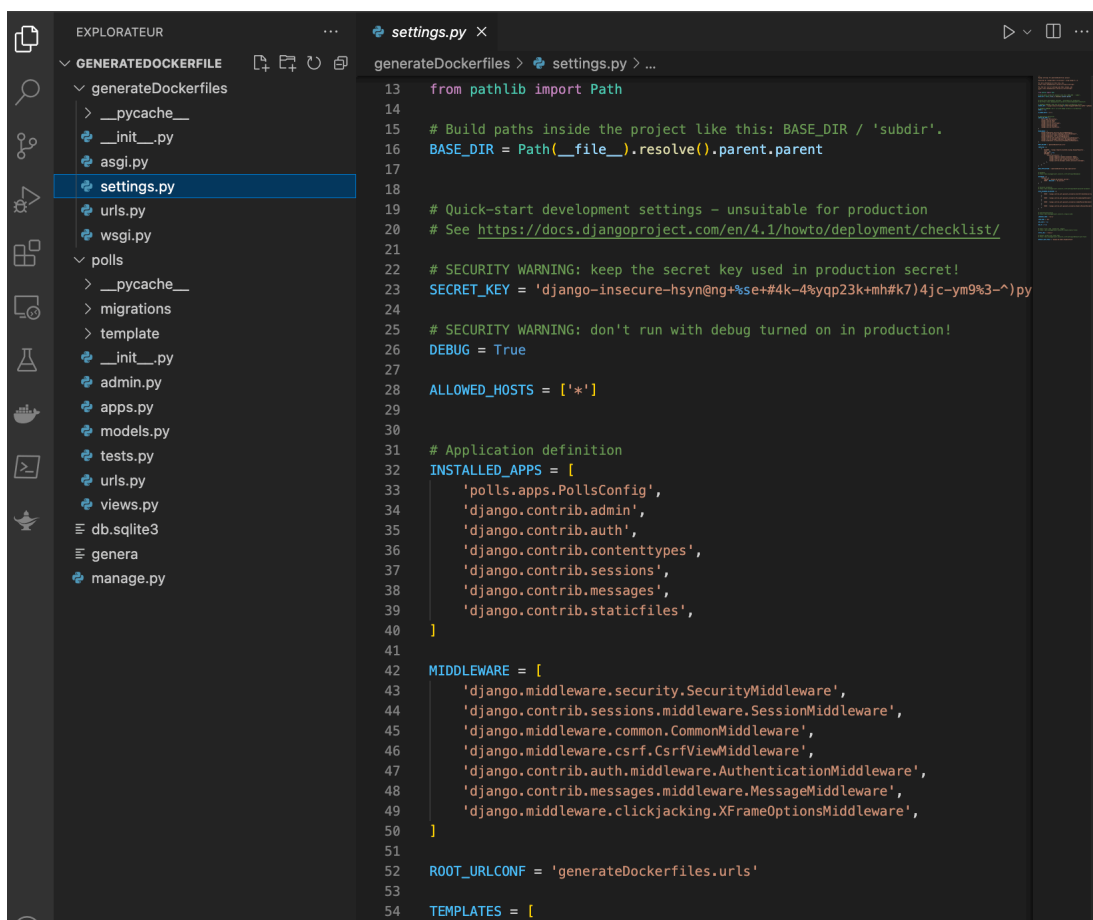
<https://openclassrooms.com/fr/courses/deployez-une-application-django/>

<https://zestedesavoir.com/tutoriels/598/developpez-votre-site-web-avec-le-framework-django/>

Lien du git comprenant le projet Django:

<https://github.com/KenzaAITO/mysite>

Aperçu du code et de l'architecture du projet que vous pouvez trouver sur git



```
13 from pathlib import Path
14
15 # Build paths inside the project like this: BASE_DIR / 'subdir'.
16 BASE_DIR = Path(__file__).resolve().parent.parent
17
18 # Quick-start development settings - unsuitable for production
19 # See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/
20
21 # SECURITY WARNING: keep the secret key used in production secret!
22 SECRET_KEY = 'django-insecure-hsyn@ng+%se+#4k-4%yqp23k+mh#k7)4jc-ym9%3~^)py
23
24 # SECURITY WARNING: don't run with debug turned on in production!
25 DEBUG = True
26
27 ALLOWED_HOSTS = ['*']
28
29 # Application definition
30
31 INSTALLED_APPS = [
32     'polls.apps.PollsConfig',
33     'django.contrib.admin',
34     'django.contrib.auth',
35     'django.contrib.contenttypes',
36     'django.contrib.sessions',
37     'django.contrib.messages',
38     'django.contrib.staticfiles',
39 ]
40
41
42 MIDDLEWARE = [
43     'django.middleware.security.SecurityMiddleware',
44     'django.contrib.sessions.middleware.SessionMiddleware',
45     'django.middleware.common.CommonMiddleware',
46     'django.middleware.csrf.CsrfViewMiddleware',
47     'django.contrib.auth.middleware.AuthenticationMiddleware',
48     'django.contrib.messages.middleware.MessageMiddleware',
49     'django.middleware.clickjacking.XFrameOptionsMiddleware',
50 ]
51
52 ROOT_URLCONF = 'generateDockerfiles.urls'
53
54 TEMPLATES = [
```