



---

## TP VCL

---

2ème Année Cycle Supérieur (2CS)  
2022-2023

Option : Systèmes Informatiques et Logiciels ( SIL )

### TP 4

Mise en conteneurs avec Dockers
---------------------------------

Réalisé par

- GOUASMIA Malak
- TOUHAR Afnane

Encadré par

D.E. Menacer (ESI)

Groupe:

SIL2

---

# Table de matiere

---

<b>Table de matiere</b>	<b>1</b>
<b>Chapitre 1</b>	<b>2</b>
Introduction	2
1. Préparation d'environnement de travail	2
<b>Chapitre 2</b>	<b>6</b>
1. Création du simple site web en python	6
1.1. Partie Python	6
1.2. Partie HTML	7
2. Création D'image	8
2.1. Création Du fichier Dockerfile	8
2.2. Exécution Du fichier Dockerfile	9
3. Accès au site	10
<b>Chapitre 3</b>	<b>12</b>
Conclusion	12

# Chapitre 1

---

## Introduction

Le Docker est un outil de virtualisation de conteneurs qui permet de déployer et d'exécuter des applications de manière isolée et reproductible. Il est devenu très populaire ces dernières années grâce à sa simplicité et sa flexibilité, ce qui en fait un choix idéal pour la création et le déploiement d'applications.

Dans ce TP, nous allons nous familiariser avec le Docker en créant une image Docker pour un projet web simple et en embarquant une application Web ou Mobile à l'intérieur de cette image. Nous apprendrons également comment accéder à cette application à partir de notre machine locale et comment la mettre à jour.

## 1. Préparation d'environnement de travail

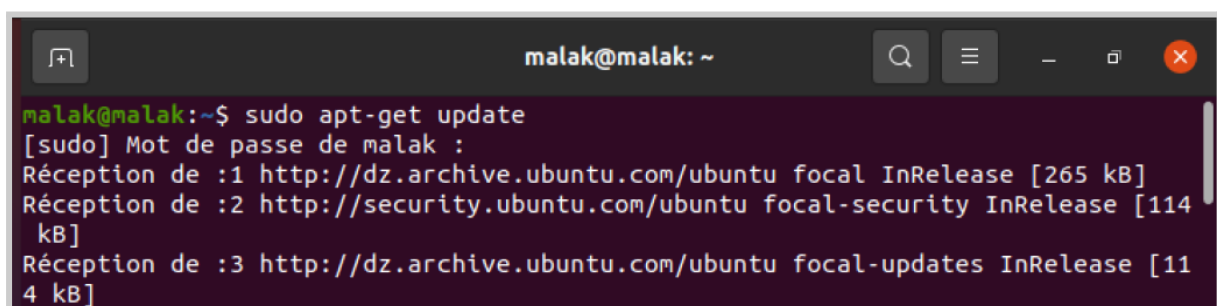
Dans ce TP, nous allons travailler avec un serveur Ubuntu 18.04 installé et un compte sur Docker Hub. Nous utiliserons Docker pour créer une image de conteneur qui lancera un projet accessible sur le port 8081 de la machine locale. Pour s'assurer d'obtenir la dernière version de Docker, nous devons ajouter une nouvelle source de package, ajouter la clé GPG de Docker et installer le package. Nous devons également suivre les instructions de configuration de Docker pour pouvoir utiliser notre compte sur Docker Hub.

### 1.1. Installation de Docker

Voici les étapes à suivre pour installer Docker sur Ubuntu :

- on a utilisé la commande suivante pour ajouter la source de package Docker :

```
sudo apt-get update
```



```
malak@malak: ~  
malak@malak:~$ sudo apt-get update  
[sudo] Mot de passe de malak :  
Réception de :1 http://dz.archive.ubuntu.com/ubuntu focal InRelease [265 kB]  
Réception de :2 http://security.ubuntu.com/ubuntu focal-security InRelease [114  
kB]  
Réception de :3 http://dz.archive.ubuntu.com/ubuntu focal-updates InRelease [11  
4 kB]
```

- Par la suite nous avons ajouté la clé GPG de Docker en utilisant la commande suivante :

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
malak@malak:~$ sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
[sudo] Mot de passe de malak :
Désolé, essayez de nouveau.
[sudo] Mot de passe de malak :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  libcurl4 python3-software-properties software-properties-gtk
Les NOUVEAUX paquets suivants seront installés :
  apt-transport-https curl
Les paquets suivants seront mis à jour :
  ca-certificates libcurl4 python3-software-properties
  software-properties-common software-properties-gtk
```

```
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
```

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
7EA0A9C3F273FCD8
```

```
malak@malak:~$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 7EA0A9C3F273FCD8
Executing: /tmp/apt-key-gpghome.HamspI5L91/gpg.1.sh --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 7EA0A9C3F273FCD8
gpg: key 8D81803C0EBFCD88: 1 duplicate signature removed
gpg: clef 8D81803C0EBFCD88 : clef publique « Docker Release (CE deb) <docker@docker.com> » importée
gpg: Quantité totale traitée : 1
gpg:          importées : 1
malak@malak:~$
```

- Puis on a ajouté le référentiel Docker en utilisant la commande suivante :

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu bionic stable"
```

```
malak@malak:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docke  
r.com/linux/ubuntu bionic stable"  
  
Atteint :1 http://dz.archive.ubuntu.com/ubuntu focal InRelease  
Atteint :2 http://dz.archive.ubuntu.com/ubuntu focal-updates InRelease  
Atteint :3 http://security.ubuntu.com/ubuntu focal-security InRelease  
Atteint :4 http://dz.archive.ubuntu.com/ubuntu focal-backports InRelease  
Atteint :5 https://download.docker.com/linux/ubuntu bionic InRelease  
Lecture des listes de paquets... Fait
```

- Il fallait mettre à jour la liste des packages disponibles en utilisant la commande suivante :

```
sudo apt-get update
```

```
malak@malak:~$ sudo apt-get update  
Atteint :1 http://security.ubuntu.com/ubuntu focal-security InRelease  
Atteint :2 https://download.docker.com/linux/ubuntu bionic InRelease  
Atteint :3 http://dz.archive.ubuntu.com/ubuntu focal InRelease  
Atteint :4 http://dz.archive.ubuntu.com/ubuntu focal-updates InRelease  
Atteint :5 http://dz.archive.ubuntu.com/ubuntu focal-backports InRelease  
Lecture des listes de paquets... Fait  
malak@malak:~$
```

- Finalement on a installé le Docker en utilisant la commande suivante :

```
sudo apt-get install docker-ce
```

```
malak@malak:~$ sudo apt-get install docker-ce  
Lecture des listes de paquets... Fait  
Construction de l'arbre des dépendances  
Lecture des informations d'état... Fait  
Les paquets supplémentaires suivants seront installés :  
  containerd.io docker-ce-cli docker-ce-rootless-extras docker-scan-plugin  
  git git-man liberror-perl pigz slirp4netns
```

- Une fois l'installation terminée, on a pu vérifier que Docker est installé et fonctionne correctement en utilisant la commande suivante :

```
sudo docker run hello-world
```

```
root@malak:/home/malak# docker --version
Docker version 20.10.22, build 3a2c30b
root@malak:/home/malak# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

# Chapitre 2

---

Ce chapitre consistera à décrire les étapes qu'il faut suivre pour créer une image Docker à partir d'une simple application web en Python. Nous allons inclure tous les détails nécessaires, tels que les commandes que nous avons utilisées, les fichiers créés (par exemple, le fichier Dockerfile) et nous allons également décrire les étapes que nous avons suivies pour lancer le conteneur à partir de cette image et vérifier que notre application web est accessible sur le port 8081 de notre machine locale.

## 1. Création du simple site web en python

La calculatrice scientifique que nous avons créée avec Python, HTML et Flask est une application web qui permet à l'utilisateur de réaliser des calculs mathématiques et de les afficher sur la page web. L'interface de l'application est intuitive et permet à l'utilisateur de sélectionner les opérations à effectuer et d'entrer les valeurs à calculer. L'application utilise Flask, un framework Python pour la création d'applications web, pour gérer les requêtes HTTP et interagir avec l'interface HTML. Grâce à Flask, nous avons pu facilement créer un site web qui offre une expérience utilisateur fluide et intuitive pour la réalisation de calculs scientifiques en ligne.

### 1.1. Partie Python

La partie Python du site web de calculatrice consiste à définir les fonctionnalités de calcul et à gérer la logique de l'application. Cela peut inclure des opérations de base comme l'addition, la soustraction, la multiplication et la division, ainsi que des opérations plus avancées comme les fonctions trigonométriques et les opérations de puissance.

```

1 from flask import Flask, render_template, request
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def index():
7     return render_template('index.html')
8
9 @app.route('/calculate', methods=['POST'])
10 def calculate():
11     num1 = request.form['num1']
12     num2 = request.form['num2']
13     operation = request.form['operation']
14     result = 0
15
16     if operation == 'add':
17         result = float(num1) + float(num2)
18     elif operation == 'subtract':
19         result = float(num1) - float(num2)
20     elif operation == 'multiply':
21         result = float(num1) * float(num2)
22     elif operation == 'divide':
23         result = float(num1) / float(num2)
24     else:
25         result = 'Invalid Operation'
26
27     return render_template('index.html', result=result)
28
29
30 if __name__ == '__main__':
31     app.run(debug=True)

```

## 1.2. Partie HTML

La partie HTML du site web de calculatrice s'occupe de la présentation visuelle de l'application. Cela peut inclure la mise en forme du formulaire de calcul, l'ajout de boutons pour les différentes opérations et la création d'un affichage pour afficher le résultat du calcul. En utilisant des technologies comme CSS et JavaScript, il est possible de donner une apparence attrayante et interactive à l'application.

```

<form action="/calculate" method="post">
  <input type="number" name="num1" placeholder="Enter a number">
  <input type="number" name="num2" placeholder="Enter another number">
  <select name="operation">
    <option value="add">+</option>
    <option value="subtract">-</option>
    <option value="multiply">*</option>
    <option value="divide">/</option>
  </select>
  <button type="submit">Calculate</button>
</form>

{% if result %}
  <p>Result: {{ result }}</p>
{% endif %}

```



En utilisant Flask, il est possible de combiner ces deux parties pour créer une application web complète de calculatrice scientifique. Flask permet de gérer les requêtes HTTP et de rendre le contenu dynamique en fonction des données entrées par l'utilisateur. Ainsi, lorsque l'utilisateur entre des données dans le formulaire de calcul et clique sur un bouton d'opération, Flask peut utiliser le code Python pour effectuer le calcul et renvoyer le résultat à l'utilisateur via l'interface HTML.

## 2. Création D'image

La création d'une image Docker à partir d'un fichier Dockerfile est un processus simple et pratique qui vous permet de définir l'environnement de conteneur de manière déclarative. Pour créer une image, nous devons d'abord écrire un fichier Dockerfile qui décrit les étapes nécessaires pour construire votre image.

### 2.1. Création Du fichier Dockerfile

Voici les étapes de base pour créer une image à partir d'un fichier Dockerfile :

Création d'un répertoire Dockerfile dont lequel on va créer le fichier Dockerfile appelé "Dockerfile"

```
root@malak:/home/malak# mkdir Dockerfile
root@malak:/home/malak# 
root@malak:/home/malak/Dockerfile# touch Dockerfile
root@malak:/home/malak/Dockerfile#
```

Dans le fichier Dockerfile on va écrire ces instructions suivantes

```
1 FROM python:3.7
2 RUN pip install flask
3 # Copier les fichiers du projet dans le répertoire de travail
4 COPY a.py /app/
5
6
7
8 # Exposer le port 8081
9 EXPOSE 8081
10
11 CMD ["python3", "./app/a.py"]
12
13
14
15
```

- La base d' image en utilisant l'instruction "FROM".

Cette instruction indique à Docker quelle image de base vous souhaitez utiliser pour votre conteneur. Par exemple, vous pouvez utiliser l'instruction "FROM ubuntu" pour utiliser l'image de base d'Ubuntu comme base de votre image.

- L'ajout des étapes de construction de votre image en utilisant les instructions appropriées.

- L utilisation de l'instruction "RUN" pour exécuter des commandes dans votre conteneur,
- "COPY" pour copier des fichiers d' application qui sera déployée dans le conteneur.
- Déclaration des ports que votre conteneur devra exposer en utilisant l'instruction "EXPOSE".

Cette instruction indique à Docker les ports sur lesquels votre conteneur sera accessible depuis l'extérieur. Dans l'énoncé du tp le port demandé était le 8081

- Déclaration de la commande à exécuter avec CMD

Cette instruction indique à Docker les commandes à exécuter lorsque le conteneur est lancé à partir de l'image et leurs paramètres dans l'ordre adéquat

- Enregistrement du fichier Dockerfile une fois qu'on a fini de le créer.

## 2.2. Exécution Du fichier Dockerfile

Dans le terminal, nous devons se positionner sur le répertoire du projet, après nous allons exécuter la commande suivante pour exécuter le fichier Dockerfile qu'on vient de créer

```
docker build -t image1 ./Dockerfile
```

```
root@malak:/home/malak# docker build -t image1 ./Dockerfile
Sending build context to Docker daemon 5.12kB
Step 1/4 : FROM python:3.8
--> 51a078947558
Step 2/4 : COPY a.py /image
--> Using cache
--> 2242934344a5
Step 3/4 : EXPOSE 8081
--> Running in 58e49813981a
Removing intermediate container 58e49813981a
--> f56c11911930
Step 4/4 : CMD ["python3", "a.py"]
--> Running in 19bb1f550e26
Removing intermediate container 19bb1f550e26
--> 311571656503
Successfully built 311571656503
Successfully tagged image1:latest
root@malak:/home/malak#
```

On remarque que l'exécution s'est effectuée avec succès, pour s'assurer encore plus que l'image était créée réellement nous allons afficher la liste des images

```
root@malak: /home/malak
root@malak:/home/malak# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
image1         latest    311571656503   About a minute ago  913MB
python         3.8       51a078947558   10 days ago    913MB
redis          latest    0256c63af7db   11 days ago    117MB
ubuntu         latest    6b7dfa7e8fdb   3 weeks ago    77.8MB
hello-world    latest    feb5d9fea6a5   15 months ago   13.3kB
root@malak:/home/malak#
```

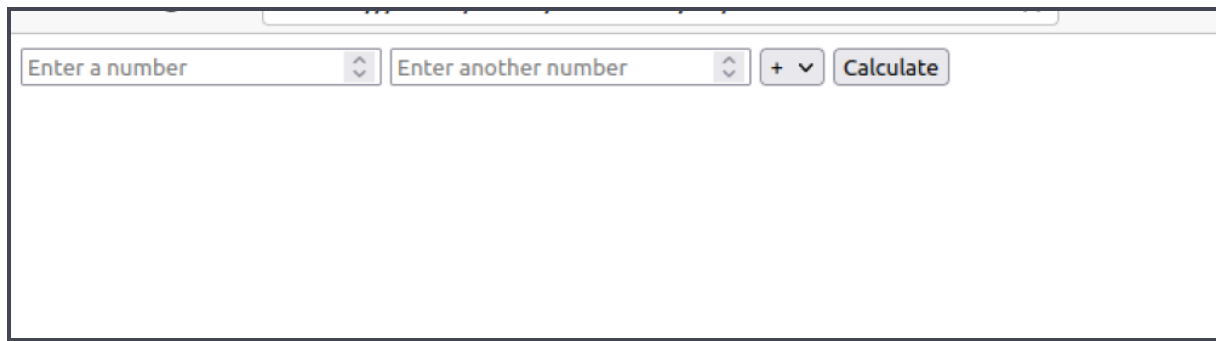
Docker exécutait alors chaque instruction de notre fichier Dockerfile et créait une image qui contient l'environnement défini par notre fichier. Nous pouvons alors utiliser cette image pour déployer un conteneur et lancer notre application.

### 3. Accès au site

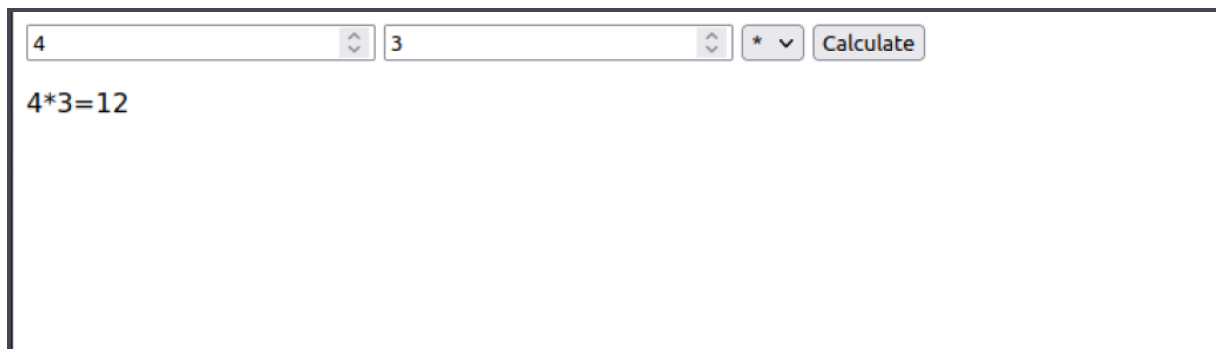
Une fois l'image Docker de notre application créée, nous avons utilisé la commande "docker run" pour la lancer dans un conteneur. Nous avons également spécifié le port sur lequel le conteneur doit être accessible, afin que nous puissions accéder à l'application en utilisant un navigateur web en entrant l'adresse "http://localhost:8081". Cela nous a permis d'accéder à notre application de calculatrice scientifique depuis n'importe quel appareil connecté à notre réseau local.

```
root@malak: /home/malak
root@malak:/home/malak# docker run -p 8081:8081 image1
* Serving Flask app 'a'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment
. Use a production WSGI server instead.
* Running on http://127.0.0.1:8081
Press CTRL+C to quit
```

- Voici l’affichage du site



A screenshot of a web browser window showing a simple form. At the top, there is a browser address bar. Below it, the form contains two text input fields. The first field is labeled "Enter a number" and the second is labeled "Enter another number". Both fields have a small upward and downward arrow icon on their right side. To the right of the second input field is a small button with a "+" sign and a downward arrow. Further right is a button labeled "Calculate". The rest of the page area is empty.



A screenshot of the same web browser window, but now with data entered. The first input field contains the number "4" and the second contains the number "3". The "+" button now displays a "\*" symbol. Below the input fields, the text "4\*3=12" is displayed in a monospaced font. The "Calculate" button remains visible to the right.

# Chapitre 3

---

## Conclusion

En conclusion, nous avons vu comment créer une image Docker à partir d'un fichier Dockerfile. Nous avons également vu comment déployer cette image dans un conteneur et comment accéder à l'application web créée en utilisant Python, HTML et Flask.

Pour créer l'image Docker, nous avons utilisé la commande "docker build" en spécifiant le chemin du fichier Dockerfile et en donnant un nom à l'image. Nous avons ensuite démarré le conteneur en utilisant la commande "docker run" en spécifiant le port que nous voulions utiliser pour accéder à l'application.

L'application elle-même était une calculatrice scientifique que nous avons créée en utilisant Python et Flask. Nous avons également utilisé du HTML pour créer l'interface utilisateur de l'application.

Enfin, nous avons vu comment accéder à l'application en entrant l'adresse "localhost" suivie du port spécifié dans notre commande "docker run" dans un navigateur web.

En résumé, ce TP nous a permis de comprendre comment créer une image Docker, déployer cette image dans un conteneur et accéder à une application web créée avec Python, HTML et Flask. Cela nous permet de facilement déployer et mettre en production des applications web de manière rapide et efficace.