

TD 6

(J'ai refait le TD sur mon PC entreprise car pb de permission d'accéder à mon file.txt sur mon mac.)

Je commence par pull les images mongo et python.

```
C:\Users\ymerine\td6>docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
b65bcf19d144: Pull complete
83449d568304: Pull complete
20d982b8a1de: Pull complete
40b500d3ec7f: Pull complete
e040b873b6d4: Pull complete
96629844a56c: Pull complete
cf5ae0a54c6d: Pull complete
1fd08fb20faf: Pull complete
39e93bf7666c: Pull complete
Digest: sha256:a4f2db6f54aeabba562cd07e5cb758b55d6192dcc6f36322a334ba0b0120aaf1
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest

C:\Users\ymerine\td6>docker pull python
Using default tag: latest
latest: Pulling from library/python
32fb02163b6b: Downloading [=====>] 21.96MB/55.05MB
167c7feebec8: Download complete
32fb02163b6b: Pull complete
167c7feebec8: Pull complete
d6dfff1f6f3d: Pull complete
e9cdcd4942eb: Pull complete
ca3bce705f6c: Pull complete
5e1c6c4f8bbf: Pull complete
efba3dc31239: Pull complete
b45fafb4411c: Pull complete
70eb3e954fe5: Pull complete
Digest: sha256:d3c16df33787f3d03b2e096037f6deb3c1c5fc92c57994a7d6f2de018de01a6b
Status: Downloaded newer image for python:latest
docker.io/library/python:latest

C:\Users\ymerine\td6>_
```

Puis je crée mon application Flask. Je commence par créer mon **app.py**.

```
app.py Dockerfile
C: > Users > ymerine > td6 > app.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def hello():
7      return 'Hello, World!'
8
9  # Start the Flask application if this file is being executed as the main
10 if __name__ == "__main__":
11     # Start the Flask application, listening on all available interfaces
12     app.run(host="0.0.0.0")
13
```

Ce code va créer une nouvelle application Flask et définit une route qui va renvoyer le message « Hello, World ! ».

Je crée ensuite un **Dockerfile**, dans le même dossier que **app.py**, qui va définir l'environnement de notre application.

```
app.py Dockerfile X
C: > Users > ymerine > td6 > Dockerfile > ...
1  FROM python:latest
2  WORKDIR /app
3  COPY . ./
4  RUN pip install -r requirements.txt
5  EXPOSE 5000
6  CMD ["python", "-m", "flask", "run", "--host=0.0.0.0"]
```

Ce Dockerfile démarre à partir de l'image officielle de Python 3.8, définit le répertoire de travail à /app, copie le fichier requirements.txt dans le conteneur, installe les dépendances, copie le reste des fichiers dans le conteneur et, enfin, exécute le script app.py. Voici mon fichier texte requirements qui va indiquer les dépendances de mon application Flask.

```
requirements.txt - Bloc-notes
Fichier Edition Format Affichage Aide
Flask
```

A présent, je crée mon image Docker et à partir de mon image je run un nouveau container.

```
C:\Users\ym erine\td6>docker build -t my-flask-app .
[+] Building 11.1s (9/9) FINISHED
=> [internal] load build definition from Dockerfile 0.3s
=> => transferring dockerfile: 188B 0.0s
=> [internal] load .dockerignore 0.2s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/python:latest 0.0s
=> [1/4] FROM docker.io/library/python:latest 1.1s
=> [internal] load build context 0.4s
=> => transferring context: 598B 0.0s
=> [2/4] WORKDIR /app 0.1s
=> [3/4] COPY . ./ 0.1s
=> [4/4] RUN pip install -r requirements.txt 8.5s
=> exporting to image 0.7s
=> => exporting layers 0.4s
=> => writing image sha256:aa6044476fe1008c391d3a7ed9bbb330e63c4e86febed59fe 0.0s
=> => naming to docker.io/library/my-flask-app 0.0s

C:\Users\ym erine\td6>
C:\Users\ym erine\td6>docker run -p 5000:5000 my-flask-app
* 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 all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

Quand j'ouvre mon navigateur, cela affiche bien « Hello, World ! ».



Part 1: Without using docker-compose

Step 1

L'objectif est d'ajouter à mon application la fonctionnalité de lire un fichier texte sur mon hôte et d'afficher son contenu sur une page Web. Toute modification du contenu de ce fichier doit être affichée en rafraîchissant la page web grâce à l'utilisation du bind mount.

Ainsi, je dois modifier mon **app.py** avec la fonction **hello()** qui lit le contenu du fichier texte **file.txt** et le restitue dans mon modèle HTML.

```
app.py x Dockerfile index.html
C: > Users > ymerine > td6 > app.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def hello():
7      with open('file.txt', 'r') as f:
8          content = f.read()
9      return content
10
11
12 # Start the Flask application if this file is being executed as the main script
13 if __name__ == "__main__":
14     # Start the Flask application, listening on all available interfaces
15     app.run(host="0.0.0.0")
16
```

Ensuite je modifie mon Dockerfile en incluant **VOLUME /app** pour inclure le fichier file.txt comme un bind mount. Cette ligne va créer un volume appelé **/app** dans mon conteneur.

```
app.py Dockerfile index.html
C: > Users > ymerine > td6 > Dockerfile > ...
1  FROM python:latest
2  WORKDIR /app
3  VOLUME /app
4  COPY . ./
5  RUN pip install -r requirements.txt
6  EXPOSE 5000
7  CMD ["python", "-m", "flask", "run", "--host=0.0.0.0"]
8
```

Voici mon modèle html et mon fichier **file.txt** :

```
app.py Dockerfile index.html
C: > Users > ymerine > td6 > index.html > ...
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Contenu du fichier</title>
5      </head>
6      <body>
7          <h1>Contenu du fichier</h1>
8          <p>{{ content }}</p>
9      </body>
10 </html>
11
```

file.txt - Bloc-notes

Fichier Edition Format Affichage Aide

Les volumes Docker sont des mécanismes de stockage de données

Persistance des données : Les volumes Docker sont persistants

Partage de données entre conteneurs : Les volumes Docker peuvent être montés dans n'importe quel conteneur

Séparation des données et du code : En utilisant les volumes

Flexibilité : Les volumes Docker peuvent être montés dans n'importe quel conteneur

Sauvegarde et récupération des données : Les volumes Docker peuvent être sauvegardés et restaurés

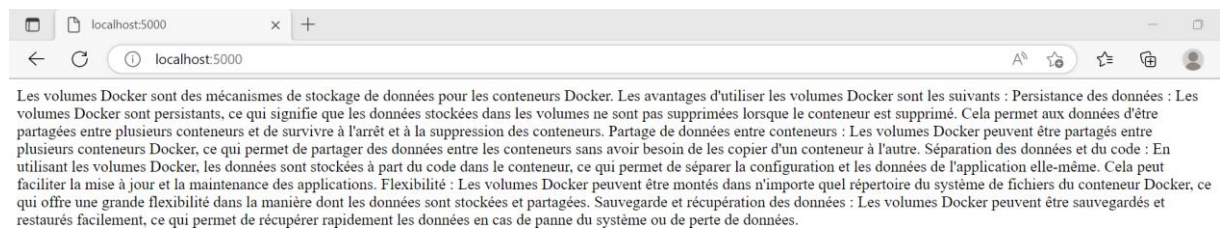
MERINE Yasmine
CCC1

A présent je peux run mon conteneur Docker en utilisant cette commande ci-dessous, cette commande indique à Docker de monter le fichier **file.txt** sur la machine hôte vers le fichier **/app/file.txt** à l'intérieur du conteneur.

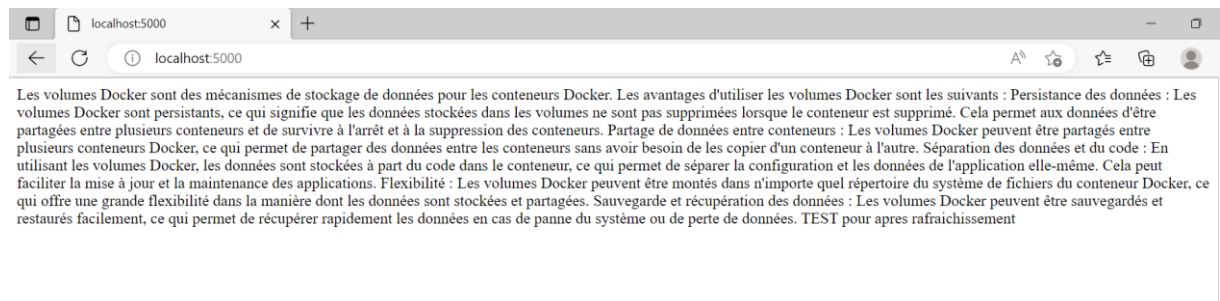
```
C:\Users\ymerine\td6> docker run -d -p 5000:5000 -v C:\Users\ymerine\td6\file.txt:/app/file.txt flaskimg
f3fb86d4cbb9f3ba101a03b650e263a70ad98b53d054d7352f7b358bed6818ca

C:\Users\ymerine\td6>
```

Ainsi, quand je vais sur mon navigateur je peux voir le contenu de mon file.txt :



Lorsque je modifie localement mon **file.txt** et que je rafraichis ma page je peux voir la modification apportée :



Step 2:

A présent, je vais ajouter MongoDB à mon application Docker. Pour cela je dois modifier mon app.py, Dockerfile et requirements.txt :

```
app.py Dockerfile index.html
C: > Users > ymerine > td6 > Dockerfile > ...
1 FROM python:latest
2 WORKDIR /app
3 COPY app.py .
4 COPY requirements.txt .
5 RUN pip3 install -r requirements.txt
6 EXPOSE 5000
7 VOLUME /app
8 CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
9
```

```
app.py Dockerfile index.html
C: > Users > ymerine > td6 > app.py > ...
1 from flask import Flask
2 from pymongo import MongoClient
3
4 app = Flask(__name__)
5 # Connect the database
6 client = MongoClient("mongodb://mongodb:27017/")
7
8 db = client.td6db
9
10 collection = db.td6_collection
11
12 app = Flask(__name__)
13
14 # Insert data in mongodb
15 documents = [{"name": "Jane Doe", "age": 25}, {"name": "Jim Smith", "age": 35}]
16 collection.insert_many(documents)
17
18 @app.route('/')
19 def hello():
20     # Fetch all document from the test_collection
21     data = list(collection.find({}))
22     with open('file.txt', 'r') as f:
23         content = f.read()
24     # return content
25     return str(data) + "\n" + content
26 # Start the Flask application if this file is being executed as the main script
27 if __name__ == "__main__":
28     # Start the Flask application, listening on all available interfaces
29     app.run(debug=True, host="0.0.0.0")
```

Ce code importe la classe MongoClient du paquet pymongo, crée un nouveau client connecté à un conteneur MongoDB appelé mongo sur le port 27017

requirements.txt - Bloc-notes

Fichier Edition Format Affichage Aide

Flask
pymongo

MERINE Yasmine
CCC1

Puis, je crée un network pour connecter mon application Flask à mon container MongoDB grâce à la commande : **docker network create my-network**

Ensuite, je crée mon conteneur mongo et volumes : **docker run -d --name mongodb --network my-network -p -v C:\Users\ymerine\td6\db:/data/db mongo**

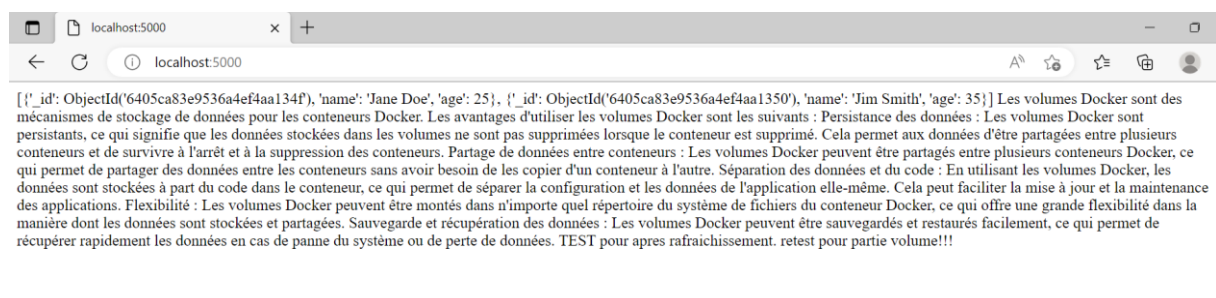
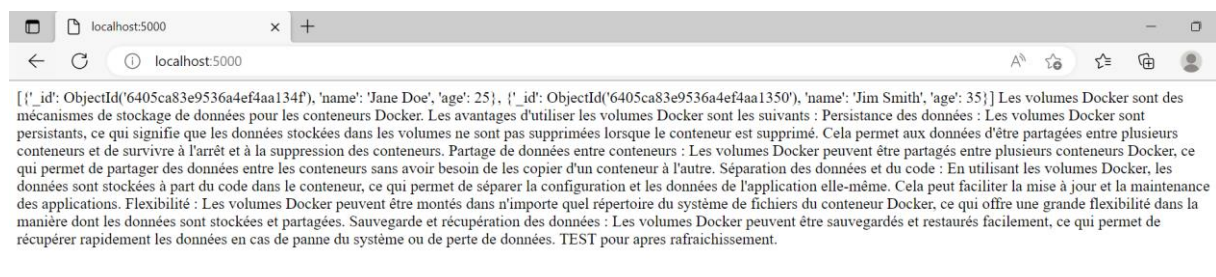
Cette dernière va créer un conteneur MongoDB et monter un volume sur le chemin local "C:\Users\ymerine\td6\db" dans le conteneur MongoDB sur le chemin "/data/db". Cela permet de stocker les données de la base de données MongoDB de manière persistante sur le système hôte.

Enfin, je run mon application : **docker run -d --network my-network -p 5000:5000 -v C:\Users\ymerine\td6\file.txt:/app/file.txt tp6**

-v C:\Users\ymerine\td6\file.txt:/app/file.txt : monte un volume sur le chemin local "C:\Users\ymerine\td6\file.txt" dans le conteneur sur le chemin "/app/file.txt". Cela permet de stocker le fichier "file.txt" sur le système hôte et de le rendre accessible dans le conteneur.

Dans les deux cas, Docker va automatiquement créer un volume sur le système hôte si celui-ci n'existe pas déjà. Si on veut utiliser un volume existant, on peut spécifier le nom du volume à la place du chemin local.

On ouvre dans le navigateur et on peut voir que si je modifie mon file.txt et que je rafraichis la page, les modifications sont visibles.



Je vérifie que mon conteneur mongo utilise bien des volumes : **docker inspect mongodb**

```
"Cmd": [
    "mongod"
],
"Image": "mongo",
"Volumes": {
    "/data/configdb": {},
    "/data/db": {}
},
"WorkingDir": "",
"Entrypoint": [
    "docker-entrypoint.sh"
],
"OnBuild": null,
"Labels": {
    "org.opencontainers.image.ref.name": "ubuntu",
    "org.opencontainers.image.title": "MongoDB"
}
```

Juste pour une vérification je vais voir si ma base de données a bien été créée et son contenu : **docker exec -it mongodb sh**

```
test> show dbs
admin      40.00 KiB
config    108.00 KiB
local      72.00 KiB
td6db      72.00 KiB
td6db> db.td6_collection.find()
[switched to db td6db
  {
    _id: ObjectId("6405ca83e9536a4ef4aa134f"),
    name: 'Jane Doe',
    age: 25
  },
  {
    _id: ObjectId("6405ca83e9536a4ef4aa1350"),
    name: 'Jim Smith',
    age: 35
  },
  {
    _id: ObjectId("6405e6ea72a83877bf6bb624"),
    name: 'Jane Doe',
    age: 25
  },
  {
    _id: ObjectId("6405e6ea72a83877bf6bb625"),
    name: 'Jim Smith',
    age: 35
  }
]
td6db>
```


Migration

Pour migrer un conteneur MongoDB avec un volume existant, vous pouvez utiliser la commande `docker export` pour exporter les données du conteneur sous forme d'archive tar, puis utiliser la commande `docker import` pour importer les données dans un nouveau conteneur.

Si vous voulez migrer en local, vous pouvez simplement lier l'ancien volume db à votre nouveau conteneur mongo.

Voici les étapes pour migrer correctement un conteneur MongoDB : tout d'abord éteindre mon conteneur MongoDB. Puis exporter les données de mon conteneur mongodb en tant que fichier archive, puis je crée un nouveau container mongodb-new avec le même volume et j'importe les données du fichier archive dans le nouveau container. Ensuite je vérifie à l'aide de la dernière commande si le nouveau container a bien reçu la data.

```
C:\Users\ymerine\td6>docker export mongodb > mongodb.tar

C:\Users\ymerine\td6>docker run -d --name mongodb-new -v db:/data/db mongo:latest
e156a416da627b379a75822c0bf0d564963a0ac1d8d447d0ffe7202e4d19537c

C:\Users\ymerine\td6>docker import - mongodb-new < mongodb.tar
sha256:cde183072adc43743ab02fa5a4eba8c7bc67e5c5e6c1e17fb6e24dcc6a9dcd0

C:\Users\ymerine\td6>docker run -it --rm --volumes-from mongodb-new busybox ls /data
/db
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
205dae5015e7: Pull complete
Digest: sha256:7b3ccabffc97de872a30dfd234fd972a66d247c8cfc69b0550f276481852627c
Status: Downloaded newer image for busybox:latest
WiredTiger                                diagnostic.data
WiredTiger.lock                           index-1-9124530705300850882.wt
WiredTiger.turtle                         index-3-9124530705300850882.wt
WiredTiger.wt                             index-5-9124530705300850882.wt
WiredTigerHS.wt                           index-6-9124530705300850882.wt
_mdb_catalog.wt                           journal
collection-0-9124530705300850882.wt      mongod.lock
collection-2-9124530705300850882.wt      sizeStorer.wt
collection-4-9124530705300850882.wt      storage.bson

C:\Users\ymerine\td6>
```

Busybox est une image Docker légère qui fournit un ensemble minimal d'outils UNIX et qui peut être utilisée pour exécuter des commandes dans un conteneur Docker.

Cela affiche le contenu du répertoire **/data/db** dans le conteneur 2.

MERINE Yasmine
CCC1

```
C:\Users\ymeline\td6>docker run -it --rm --volumes-from mongodb busybox ls /data/db
WiredTiger
WiredTiger.lock
WiredTiger.turtle
WiredTiger.wt
WiredTigerHS.wt
_mdb_catalog.wt
collection-0--4850762794918895801.wt
collection-0-7304617297637046569.wt
collection-2-7304617297637046569.wt
collection-4-7304617297637046569.wt
diagnostic.data
index-1--4850762794918895801.wt
index-1-7304617297637046569.wt
index-3-7304617297637046569.wt
index-5-7304617297637046569.wt
index-6-7304617297637046569.wt
journal
mongod.lock
sizeStorer.wt
storage.bson

C:\Users\ymeline\td6>
```

Pour vérifier que les données ont été correctement importées du conteneur 1 vers le conteneur 2, on exécute la même commande sur le conteneur 2 et comparer les résultats. Les deux sorties de la commande sont identiques, cela signifie que les données ont été correctement transférées.

A présent, pour partager le conteneur MongoDB migré avec une autre instance du même moteur de base de données, vous pouvez pousser le conteneur vers un registre Docker, puis le tirer sur l'autre instance.

Voici les étapes à suivre pour partager le conteneur MongoDB migré : je commence par push le conteneur dans mon registre Docker.

```
C:\Users\ymeline\td6>docker tag mongodb-new myregistry.com/mongodb-new

C:\Users\ymeline\td6>docker push myregistry.com/mongodb-new
Using default tag: latest
The push refers to repository [myregistry.com/mongodb-new]
a45fcc670fcd: Preparing
error parsing HTTP 404 response body: invalid character '<' looking for beginning of value
: "\r\n\r\n!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w
3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\r\n<html xmlns="http://www.w3.org/1999/xht
ml" class="mr-culture-en-US" lang="en">\r\n<head><title>\r\n\tMyRegistry.com - Baby R
egistry, Bridal Registry, Wedding Registry, Universal Gift Registry\r\n</title><link rel="
icon" type="image/png" sizes="16x16" href="/images/icons/favicons/favicon-16x16.png
"><link rel="icon" type="image/png" sizes="32x32" href="/images/icons/favicons/fav
icon-32x32.png"><link rel="icon" type="image/png" sizes="192x192" href="/images/ic
ons/favicons/favicon-192x192.png"><link rel="shortcut icon" href="/images/icons/favico
ns/favicon.ico"><meta name="theme-color" content="#ffffff"><meta http-equiv="X-UA-Co
mpatible" content="IE=edge" />\r\n <script type="text/javascript" src="/JavaScrip
tResourceHandler.ashx?culture=en-US&version=202303031927428991"></script>\r\n <script
type="text/javascript" src="/ScriptSet/MrJqueryScripts.js?version=202303031840109629">
</script>\r\n <script type="text/javascript" src="/ScriptSet/MrScripts.js?version=20
2303031840109629"></script>\r\n <script type="text/javascript" src="/Scripts/CheckA
uthCookie.js?version=202303031840109629"></script>\r\n <link href="/StyleSet/MrMainSt
yles.css?version=202303031927428991" rel="stylesheet" type="text/css" />\r\n\r\n\r\n<
!-- Quantcast Tag -->\r\n<script type="text/javascript">\r\nvar _qevents = _qevents || [
];\r\n\r\n(function() {\r\nvar elem = document.createElement('script');\r\nelem.src = (doc
ument.location.protocol == "https:" ? "https://secure" : "http://edge") + ".quantse
rve.com/quant.js";\r\nelem.async = true;\r\nelem.type = "text/javascript";\r\nvar scpt
= document.getElementsByTagName('script')[0];\r\nscpt.parentNode.insertBefore(elem, scpt);
\r\n})();\r\n\r\n_qevents.push({\r\nqacct:"p-6bNL-aPgbdPBU"\r\n});\r\n</script>\r\n\r\n<
noscript>\r\n<div style="display:none;">\r\n\r\n</div>\r\n<
/noscript>\r\n<!-- End Quantcast tag -->\r\n\r\n\r\n<script async src="https://www.google
tagmanager.com/gtag/js?id=G-M8W8DX2XZQ"></script>\r\n<script>\r\n // GA4\r\n window
```

Sur l'autre instance du moteur de base de données, je pull le conteneur du registre Docker :

```
C:\Users\ymerine\td6>docker pull myregistry.com/mongodb-new
Using default tag: latest
Error response from daemon: error parsing HTTP 404 response body: invalid character '<' looking for beginning of value: "<!\DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml" class="mr-culture-en-US" lang="en"><head><title><!-- MyRegistry.com - Baby Registry, Bridal Registry, Wedding Registry, Universal Gift Registry --></title><link rel="icon" type="image/png" sizes="16x16" href="/images/icons/favicons/favicon-16x16.png"><link rel="icon" type="image/png" sizes="32x32" href="/images/icons/favicons/favicon-32x32.png"><link rel="icon" type="image/png" sizes="192x192" href="/images/icons/favicons/favicon-192x192.png"><link rel="shortcut icon" href="/images/icons/favicons/favicon.ico"><meta name="theme-color" content="#ffffff"><meta http-equiv="X-UA-Compatible" content="IE=edge" /><script type="text/javascript">
<script src="/JavaScriptResourceHandler.ashx?culture=en-US&version=202303031927424578"></script><script type="text/javascript">
<script src="/ScriptSet/MrJqueryScripts.js?version=202303031840109629"></script><script type="text/javascript">
<script src="/ScriptSet/MrScripts.js?version=202303031840109629"></script><script type="text/javascript">
<script src="/Scripts/CheckAuthCookie.js?version=202303031840109629"></script></li>
```

Puis je run un nouveau conteneur **mongodb-new2**

```
C:\Users\ymerine>docker run -d --name mongodb-new2 -v db:/data/db myregistry.com/mongodb-new mongod --bind_ip_all
b2fb33ce6b36f036135a71b8f340bd02d38d304620e4dd0d1fc4fcd6c5b17
```

Containers Give feedback							
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more							
<div><div></div> Only show running containers</div> <div>Search</div>							
	Name	Image	Status	Port(s)	Last started	Actions	
<input type="checkbox"/>	<div><div>optimistic_stonebraker</div><div>3ade0270be3d</div></div>	tp6	Running	5000:5000	1 hour ago	<div></div>	<div></div>
<input type="checkbox"/>	<div><div>mongodb</div><div>02174cef815c</div></div>	mongo	Exited		45 minutes ago	<div></div>	<div></div>
<input type="checkbox"/>	<div><div>naughty_herschel</div><div>f3fb86d4cbb9</div></div>	flaskimg	Exited (137)	5000:5000	1 hour ago	<div></div>	<div></div>
<input type="checkbox"/>	<div><div>funny_yonath</div><div>b3551f11fd44</div></div>	my-flask-app	Exited (137)	5000:5000	4 hours ago	<div></div>	<div></div>
<input type="checkbox"/>	<div><div>mongodb-new</div><div>e156a416da62</div></div>	mongo:latest	Exited		41 minutes ago	<div></div>	<div></div>
<input type="checkbox"/>	<div><div>mongodb-new2</div><div>b2fb33ce6b36</div></div>	myregistry.com/mongodb-ne	Running		0 seconds ago	<div></div>	<div></div>

```
C:\Users\ymerine>docker run -it --rm --volumes-from mongodb-new busybox ls /data/db
WiredTiger
WiredTiger.lock
WiredTiger.turtle
WiredTiger.wt
WiredTigerHS.wt
_mdb_catalog.wt
collection-0-9124530705300850882.wt
collection-2-9124530705300850882.wt
collection-4-9124530705300850882.wt
diagnostic.data
index-1-9124530705300850882.wt
index-3-9124530705300850882.wt
index-5-9124530705300850882.wt
index-6-9124530705300850882.wt
journal
mongod.lock
sizeStorer.wt
storage.bson

C:\Users\ymerine>docker run -it --rm --volumes-from mongodb-new2 busybox ls /data/db
WiredTiger
WiredTiger.lock
WiredTiger.turtle
WiredTiger.wt
WiredTigerHS.wt
_mdb_catalog.wt
collection-0-9124530705300850882.wt
collection-2-9124530705300850882.wt
collection-4-9124530705300850882.wt
diagnostic.data
index-1-9124530705300850882.wt
index-3-9124530705300850882.wt
index-5-9124530705300850882.wt
index-6-9124530705300850882.wt
journal
mongod.lock
sizeStorer.wt
storage.bson

C:\Users\ymerine>
```

On peut voir que les données ont bien été copiées.

Part 2: By using docker-compose

Je crée un **docker-compose.yaml** dans le même dossier où il y a mon app.py et mon Dockerfile.

```
app.py  Dockerfile  docker-compose.yaml  index.html

C: > Users > ymerine > td6 > docker-compose.yaml
1  version: '3'
2  services:
3    web:
4      build: .
5      ports:
6        - "5000:5000"
7      volumes:
8        - ./file.txt:/app/file.txt
9
10   mongodb:
11     image: mongo:latest
12     volumes:
13       - ./db:/data/db
14
```











Ce code définit deux services : web et mongo. Le service web construit l'image Docker définie dans le répertoire courant (.) et mappe le port 5000 du conteneur au port 5000 de la machine hôte. Il


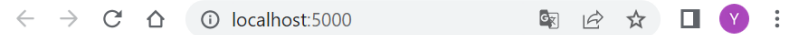
MERINE Yasmine
CCC1

monte également le fichier file.txt sur la machine hôte vers le fichier /app/file.txt à l'intérieur du conteneur. Le service mongo utilise l'image officielle mongo et mappe le port 27017 du conteneur au port 27017 de la machine hôte. Il crée également un nouveau volume appelé td6db-data pour stocker les données MongoDB.

```
C:\Users\ymerine\td6>docker-compose up -d
[+] Running 3/3
 - Network td6_default      Created           0.0s
 - Container td6-mongodb-1  Started        1.4s
 - Container td6-web-1      Started        1.3s
C:\Users\ymerine\td6>
```

Cette commande indique à Docker Compose de démarrer les conteneurs définis dans le fichier docker-compose.yaml.

<input type="checkbox"/>		td6	-	Running (2/2)	27 seconds ago		
<input type="checkbox"/>		mongodb-1 a2594f959789		mongo:latest	Running	27 seconds ago	
<input type="checkbox"/>		web-1 707b6b04c7d2		td6-web	Running	5000:5000 	27 seconds ago 



localhost:5000

```
[{"_id": ObjectId("6405ca83e9536a4ef4aa134f"), "name": "Jane Doe", "age": 25}, {"_id": ObjectId("6405ca83e9536a4ef4aa1350"), "name": "Jim Smith", "age": 35}, {"_id": ObjectId("6405e6ea72a83877bf6bb624"), "name": "Jane Doe", "age": 25}, {"_id": ObjectId("6405e6ea72a83877bf6bb625"), "name": "Jim Smith", "age": 35}, {"_id": ObjectId("6405f90f60ca77a031dbdcdd"), "name": "Jane Doe", "age": 25}, {"_id": ObjectId("6405f90f60ca77a031dbdcde"), "name": "Jim Smith", "age": 35}]
```

Les volumes Docker sont des mécanismes de stockage de données pour les conteneurs Docker. Les avantages d'utiliser les volumes Docker sont les suivants :
- **Persistance des données** : Les volumes Docker sont persistants, ce qui signifie que les données stockées dans les volumes ne sont pas supprimées lorsque le conteneur est supprimé. Cela permet aux données d'être partagées entre plusieurs conteneurs et de survivre à l'arrêt et à la suppression des conteneurs.
- **Partage de données entre conteneurs** : Les volumes Docker peuvent être partagés entre plusieurs conteneurs Docker, ce qui permet de partager des données entre les conteneurs sans avoir besoin de les copier d'un conteneur à l'autre.
- **Séparation des données et du code** : En utilisant les volumes Docker, les données sont stockées à part du code dans le conteneur, ce qui permet de séparer la configuration et les données de l'application elle-même. Cela peut faciliter la mise à jour et la maintenance des applications.
- **Flexibilité** : Les volumes Docker peuvent être montés dans n'importe quel répertoire du système de fichiers du conteneur Docker, ce qui offre une grande flexibilité dans la manière dont les données sont stockées et partagées.
- **Sauvegarde et récupération des données** : Les volumes Docker peuvent être sauvegardés et restaurés facilement, ce qui permet de récupérer rapidement les données en cas de panne du système ou de perte de données.

TEST pour après rafraichissement. retest