LOG3000 – Processus du génie logiciel

Hiver 2021

TP No.4

Groupe 2 - Équipe no3

1960266 – Yanis Toubal

1947497 – Yuhan Li

Soumis à : Isabella Vieira Ferreira

Jeudi 8 avril 2021

3.1 Lancer votre premier conteneur

1-2.

```
C:\Users\yanis>docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
9aae54b2144e: Pull complete
Digest: sha256:826f70e0ac33e99a72cf20fb0571245a8fee52d68cb26d8bc58e53bfa65dcdfa
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest

C:\Users\yanis>docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
alpine        latest    302aba9ce190   3 days ago    5.61MB
hello-world   latest    d1165f221234   3 weeks ago   13.3kB

C:\Users\yanis>
```

Figure 1.

3.

```
C:\Users\yanis>docker run alpine ls -l
total 56
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 bin
drwxr-xr-x    5 root     root           340 Mar 29 14:24 dev
drwxr-xr-x    1 root     root          4096 Mar 29 14:24 etc
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 home
drwxr-xr-x    7 root     root          4096 Mar 25 16:57 lib
drwxr-xr-x    5 root     root          4096 Mar 25 16:57 media
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 mnt
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 opt
dr-xr-xr-x  177 root     root             0 Mar 29 14:24 proc
drwx------    2 root     root          4096 Mar 25 16:57 root
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 run
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 sbin
drwxr-xr-x    2 root     root          4096 Mar 25 16:57 srv
dr-xr-xr-x   11 root     root             0 Mar 29 14:24 sys
drwxrwxrwt    2 root     root          4096 Mar 25 16:57 tmp
drwxr-xr-x    7 root     root          4096 Mar 25 16:57 usr
drwxr-xr-x   12 root     root          4096 Mar 25 16:57 var
```

Figure 2.

4-5.

```
C:\Users\yanis>docker run alpine echo "hello from alpine"
hello from alpine

C:\Users\yanis>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\yanis>docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED              STATUS                          PORTS     NAMES
df92b2d1dca0   alpine        "echo 'hello from al…"   About a minute ago   Exited (0) About a minute ago             elegant_newton
255e1512c92f   alpine        "ls -l"                  3 minutes ago        Exited (0) 3 minutes ago                  dazzling_wiles
811fadcc8b92   hello-world   "/hello"                 8 minutes ago        Exited (0) 8 minutes ago                  compassionate_faraday
```

Figure 3.

3.2 Exécuter une application Web statique avec Docker

1.

```
C:\Users\yanis>docker run -d dockersamples/static-site
Unable to find image 'dockersamples/static-site:latest' locally
latest: Pulling from dockersamples/static-site
fdd5d7827f33: Pull complete
a3ed95caeb02: Pull complete
716f7a5f3082: Pull complete
7b10f03a0309: Pull complete
aff3ab7e9c39: Pull complete
Digest: sha256:daa686c61d7d239b7977e72157997489db49f316b9b9af3909d9f10fd28b2dec
Status: Downloaded newer image for dockersamples/static-site:latest
a13f5b2b519035efe791cb76c99ce575798e0414b0f61d11fa024ae7e5579943
```

Figure 4.

2-3.

```
C:\Users\yanis>docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED          STATUS          PORTS             NAMES
a13f5b2b5190   dockersamples/static-site   "/bin/sh -c 'cd /usr…"   54 seconds ago   Up 53 seconds   80/tcp, 443/tcp   dreamy_torvalds

C:\Users\yanis>docker stop a13f5b2b5190
a13f5b2b5190

C:\Users\yanis>docker rm a13f5b2b5190
a13f5b2b5190
```

Figure 5.

4-5.

```
C:\Users\yanis>docker run --name static-site -e AUTHOR="Yuhan Li" -d -P dockersamples/static-site
d49ab2c249b772a506341b9f430fea3877d5ec8da411995e1922dd8df3fb740b

C:\Users\yanis>docker port static-site
443/tcp -> 0.0.0.0:49153
80/tcp -> 0.0.0.0:49154
```

Figure 6.

6.

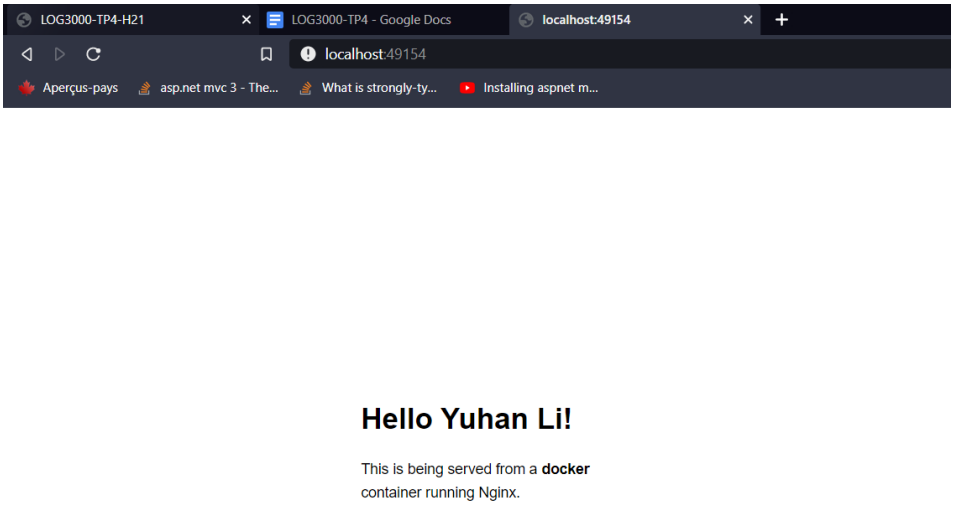

Figure 7.

3.3 Exécuter une application Web dynamique avec docker

1.

```
C:\Users\yanis\Desktop\Poly\H2021\LOG3000\Labs\LOG3000\TP4>docker images
REPOSITORY                  TAG       IMAGE ID       CREATED       SIZE
alpine                      latest    302aba9ce190   3 days ago    5.61MB
hello-world                 latest    d1165f221234   3 weeks ago   13.3kB
dockersamples/static-site   latest    f589ccde7957   5 years ago   191MB

C:\Users\yanis\Desktop\Poly\H2021\LOG3000\Labs\LOG3000\TP4>mkdir flask-app

C:\Users\yanis\Desktop\Poly\H2021\LOG3000\Labs\LOG3000\TP4>cd flask-app

C:\Users\yanis\Desktop\Poly\H2021\LOG3000\Labs\LOG3000\TP4\flask-app>
```

Figure 8.

2.

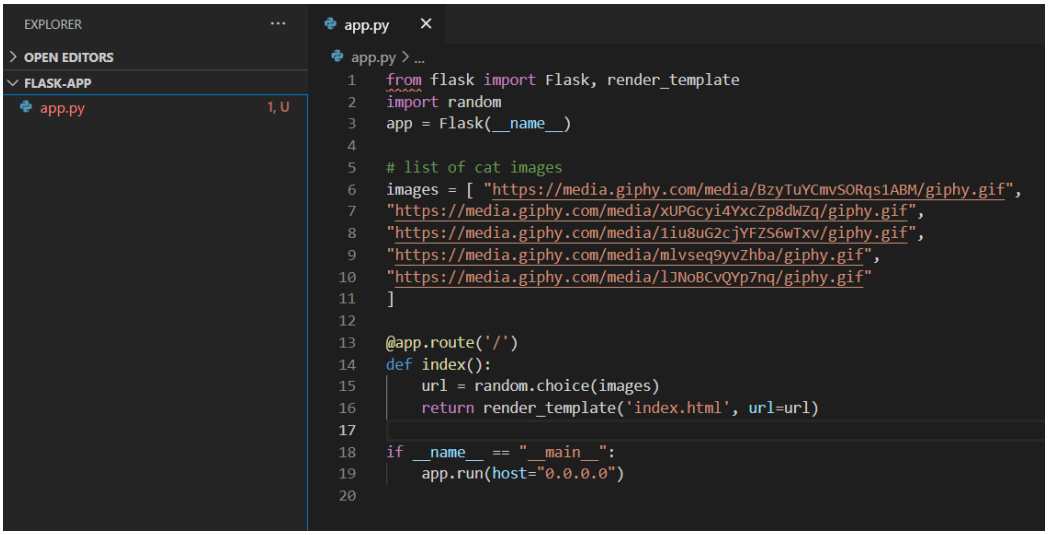

Figure 9.

3.

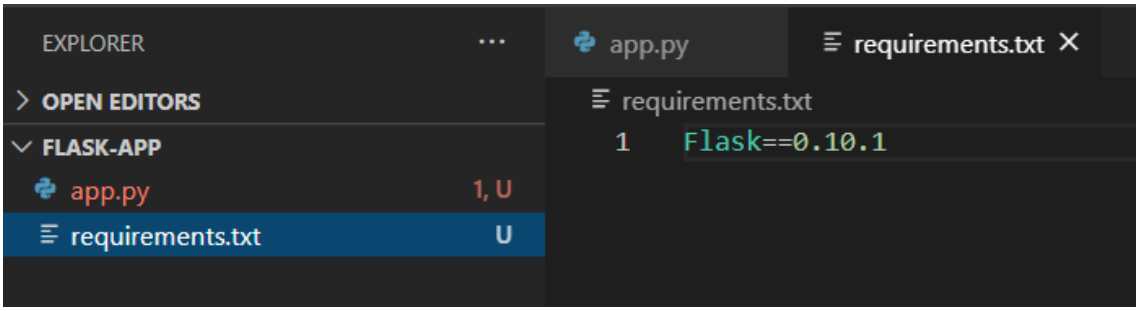

Figure 10.

4.

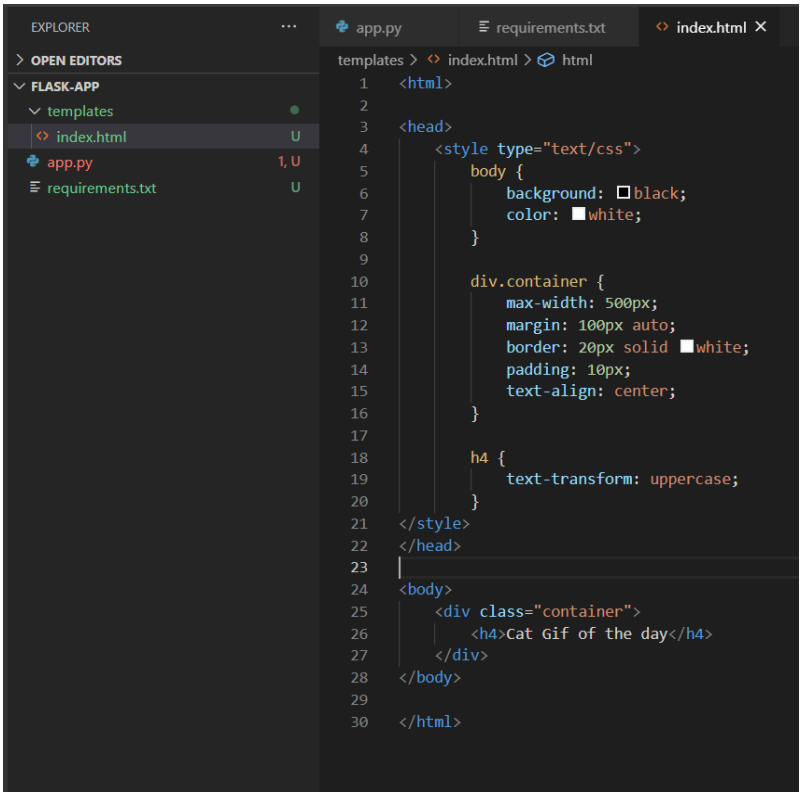

Figure 11.

5.

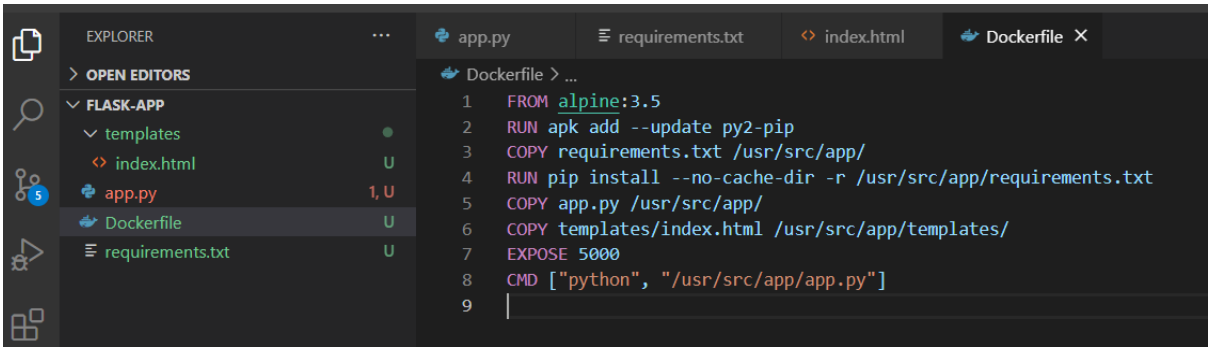

Figure 12.

6.

```
C:\Users\yanis\Desktop\Poly\H2021\LOG3000\Labs\LOG3000\TP4\flask-app>docker build -t lampe14/myfirstapp .
[+] Building 16.8s (7/10)
 => [internal] load build definition from Dockerfile                                                          0.1s
 => => transferring dockerfile: 318B                                                                          0.0s
 => [internal] load .dockerignore                                                                             0.1s
 => => transferring context: 2B                                                                               0.0s
 => [internal] load metadata for docker.io/library/alpine:3.5                                                 1.4s
 => [1/6] FROM docker.io/library/alpine:3.5@sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec   2.1s
 => => resolve docker.io/library/alpine:3.5@sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec  0.0s
 => => sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec 433B / 433B                    0.0s
 => => sha256:f7d2b5725685826823bc6b154c0de02832e5e6daf7dc25a00ab00f1158fabfc8 528B / 528B                    0.0s[+[[+] Building
 17.7s (8/10)
 => [internal] load build definition from Dockerfile                                                            0.1s
 => => transferring dockerfile: 318B                                                                            0.0s
 => [internal] load .dockerignore                                                                               0.1s
 => => transferring context: 2B                                                                                 0.0s
 => [internal] load metadata for docker.io/library/alpine:3.5                                                   1.4s
 => [1/6] FROM docker.io/library/alpine:3.5@sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec     2.1s
 => => resolve docker.io/library/alpine:3.5@sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec    0.0s
 => => sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec 433B / 433B                      0.0s
 => => sha256:f7d2b5725685826823bc6b154c0de02832e5e6daf7dc25a00ab00f1158fabfc8 528B / 528B                      0.0s
 => => sha256:f80194ae2e0ccf0f098baa6b981396dfbffb16e6476164af72158577a7de2dd9 1.51kB / 1.51kB                  0.0s
[+] Building 20.0s (11/11) FINISHED                                                                               s => [internal
] load build definition from Dockerfile                                                              0.1s
 => => transferring dockerfile: 318B                                                                          0.0s3s => [internal
] load .dockerignore                                                                                 0.1s
 => => transferring context: 2B                                                                               0.0s1s => [internal
] load metadata for docker.io/library/alpine:3.5                                                     1.4s
 => [1/6] FROM docker.io/library/alpine:3.5@sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec   2.1s0s => => resolv
e docker.io/library/alpine:3.5@sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec     0.0s
 => => sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec 433B / 433B                    0.0s3s => => sha256
:f7d2b5725685826823bc6b154c0de02832e5e6daf7dc25a00ab00f1158fabfc8 528B / 528B                        0.0s
 => => sha256:f80194ae2e0ccf0f098baa6b981396dfbffb16e6476164af72158577a7de2dd9 1.51kB / 1.51kB                0.0s1s => => sha256
:8cae0e1ac61cead281f41115cc0ebd39117f7e54dffc8fd5e05a7590dca3cd4e 1.97MB / 1.97MB                   1.7s
 => => extracting sha256:8cae0e1ac61cead281f41115cc0ebd39117f7e54dffc8fd5e05a7590dca3cd4e                     0.3s3s => [internal
] load build context                                                                                 0.1s2s => => transferring contex
t: 1.33kB                                                                               0.0s
 => [2/6] RUN apk add --update py2-pip                                                                        6.3s
 => [3/6] COPY requirements.txt /usr/src/app/                                                                 0.1s
 => [4/6] RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt                                     7.3s
 => [5/6] COPY app.py /usr/src/app/                                                                           0.2s
```

Figure 13.

7.

```
C:\Users\yanis\Desktop\Poly\H2021\LOG3000\Labs\LOG3000\TP4\flask-app>docker run -p 8888:5000 --name flask-app lampe14/myfirstapp
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [29/Mar/2021 16:48:50] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [29/Mar/2021 16:48:52] "GET /favicon.ico HTTP/1.1" 404 -
```

Figure 14.

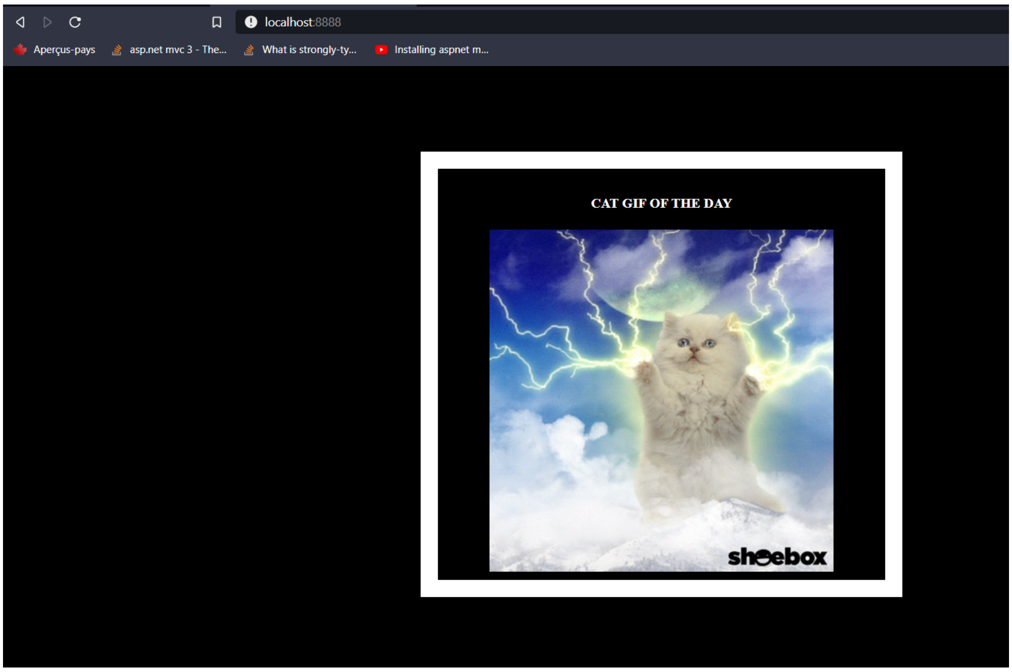

Figure 15.

3.4 «Dockerize» une application

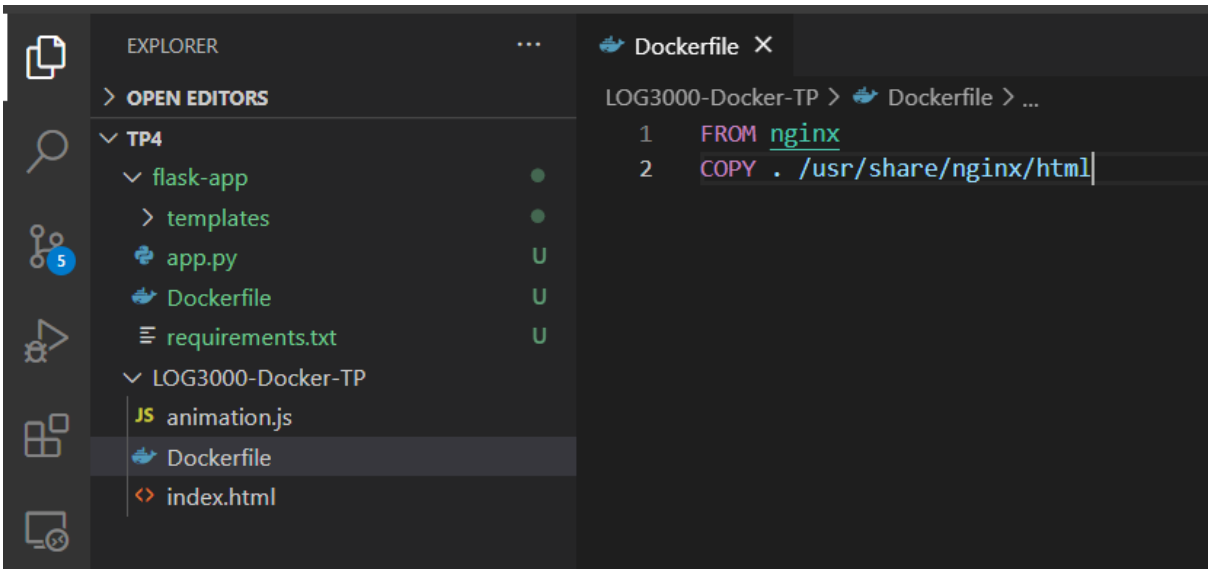

Figure 16.

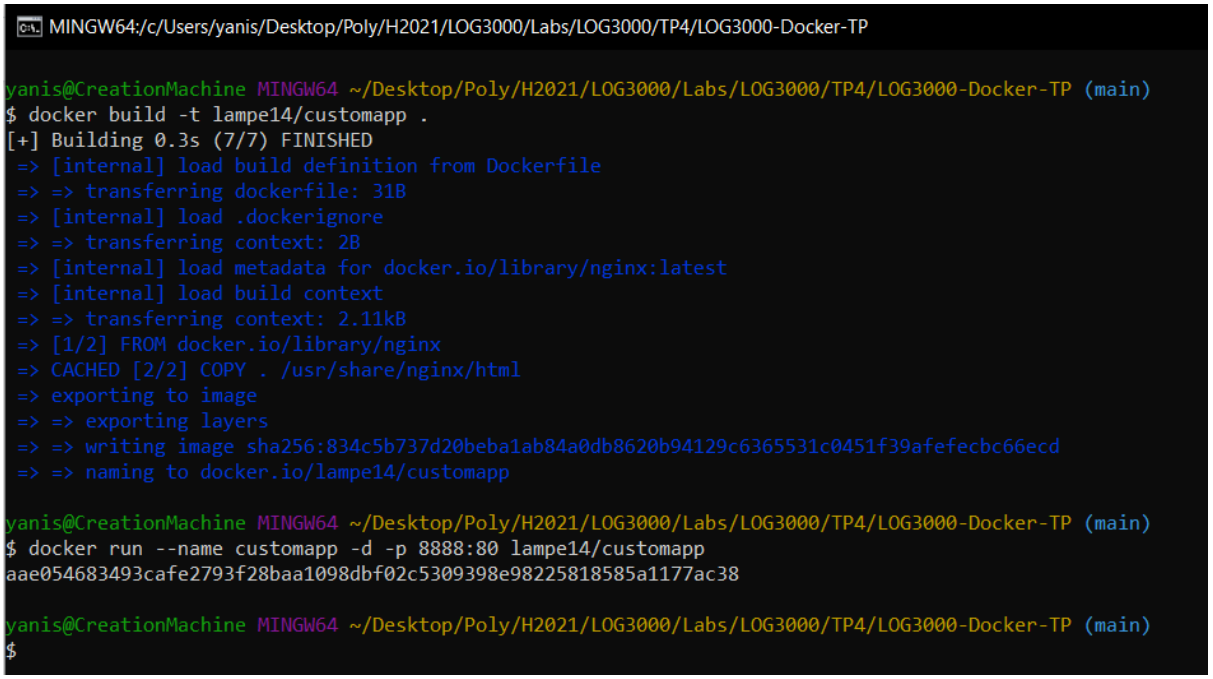

Figure 17.

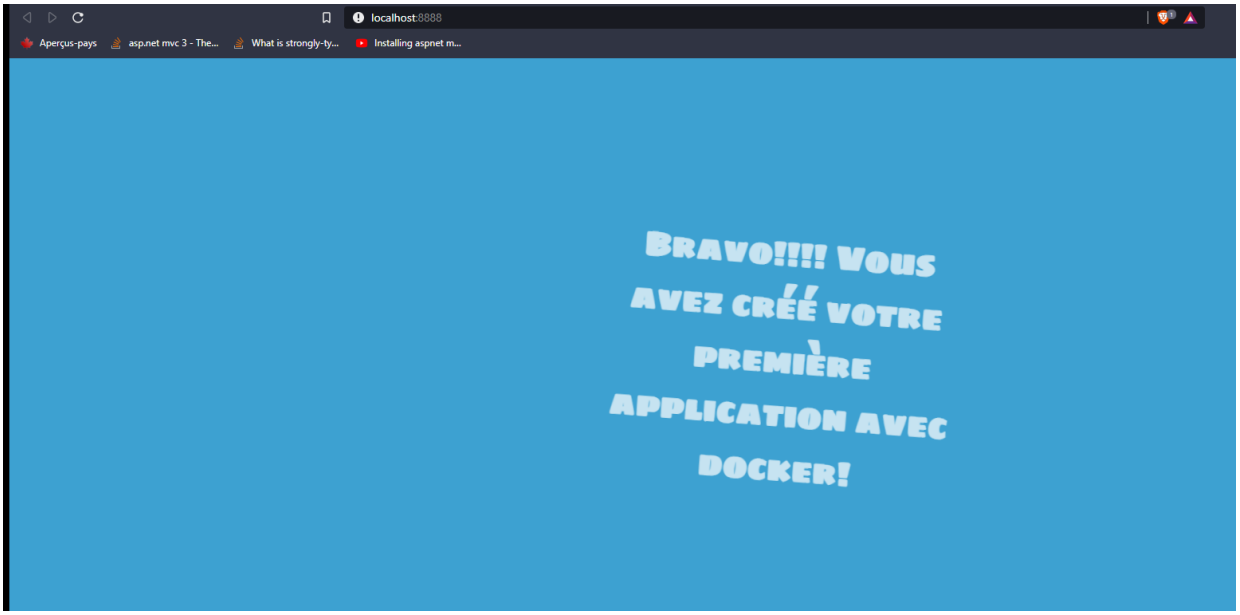
localhost:8888
Aperçus-pays
asp.net mvc 3 - The...
What is strongly-ty...
Installing aspnet m...
BRAVO!!!! VOUS AVEZ CRÉÉ VOTRE PREMIÈRE APPLICATION AVEC DOCKER!

Figure 18.

4. Questions

4.1 Questions d'analyse sur la section 3.1

1. Expliquez avec vos propres mots ce que fait la commande pull alpine .

Alpine est une distribution Linux. Dans le contexte de Docker, alpine est une *image* basée sur la distribution Linux. Lorsque nous faisons pull alpine, nous allons récupérer l'image Alpine du registre Docker et l'enregistrer dans notre système. [1] Nous pouvons également confirmer cela en observant l'affichage obtenu à la figure 1. avec le statut qui indique: « Downloaded newer image for alpine:latest».

2. Expliquez avec vos propres mots ce qui se passe derrière l'écran lorsque vous exécutez la commande docker run .

En faisant docker run, le client Docker va contacter le daemon Docker. Plus concrètement, cette commande va faire en sorte que le daemon crée un conteneur à partir d'une image donnée. Dans notre cas, il s'agit de l'image alpine qui est disponible localement. Puis, le daemon va lancer une commande (fournie en paramètre) dans ce conteneur et va finalement diffuser la sortie de cette commande vers le client Docker. [2] À la figure 2, cette commande s'agissait de « ls l-», ce qui liste tous les fichiers ou répertoires (*ls*) correspondant au dernier conteneur créé ainsi que tous ses états (*-l*). Nous pouvons voir sur cette figure que nous avons bel et bien reçu sur notre terminal une liste qui correspond à la sortie de cette commande. [3]

3. Expliquez avec vos propres mots ce qui se passe derrière l'écran lorsque vous exécutez la commande docker run alpine echo "hello from alpine". S'agit-il d'une sortie de docker ou de Linux Alpine?

Comme expliqué à la question 2. ci-haut, on lance une commande dans le conteneur lorsqu'on effectue *docker run*. Ici, cette commande s'agit de *echo* qui est une commande Linux servant à afficher une ligne de texte (chaîne de caractère) passé en argument. Ainsi, le client va lancer *echo* dans le conteneur Alpine, et la sortie de cette commande, soit l'affichage du texte « hello from alpine », sera renvoyée au client. [2] À la figure 3, il est justement possible de percevoir l'affichage de cette chaîne dans le terminal.

4. Quelle est la différence entre une image et un conteneur?

Une image représente une application et son environnement virtuel à un moment précis dans le temps. En effet, une image correspond à un fichier qui n'est pas modifiable et qui contient le code source, les librairies, les dépendances, ainsi que tous les autres outils nécessaires pour lancer l'application en question. Toutefois, il n'est pas possible de lancer ou exécuter une image, car elle n'a pas d'état en particulier; c'est plutôt un *template.* [4]

Un conteneur, lui, peut être vu comme étant l'instance d'une image en cours d'exécution, ou autrement dit un environnement virtualisé pendant son exécution. Il est possible d'avoir plusieurs conteneurs en cours d'exécution provenant de la même image. [4]

5. Quels sont les avantages d'un conteneur par rapport à une machine virtuelle?

Les conteneurs sont différents des machines virtuelles, car ils déploient un niveau de virtualisation différent. Contrairement aux machines virtuelles, où la virtualisation a lieu dans le matériel

(*hardware*), les conteneurs sont virtualisés dans la couche d'application. C'est pourquoi, les conteneurs peuvent utiliser une seule machine, partitionner son noyau, virtualiser le système d'exploitation et exécuter des processus isolés, tout ça indépendamment du système d'exploitation associée. Donc, grâce aux conteneurs, plusieurs charges de travail peuvent être exécutées sur une seule instance de système d'exploitation. Cela rend alors les conteneurs plus rapides, flexibles, légers et portatifs comparativement aux machines virtuelles. [5]

4.2 Questions d'analyse sur la section 3.2

6. Expliquez avec vos propres mots chaque paramètre utilisé à l'étape 4 de la section 3.2.

Tout d'abord, il y a le paramètre «--name» qui représente le nom qu'on attribue au conteneur que l'on va démarrer. Dans notre cas, il s'agit du nom *static-site* qui précède directement le paramètre. Ensuite, il y a le paramètre «-e» qui permet d'ajouter des variables d'environnement dans notre cas on ajoute la variable d'environnement «AUTHOR» afin d'afficher sa valeur sur le site Web statique dans le container. En continuant, il y a l'option -d qui permet de lancer le container en mode détaché et donc de ne pas afficher l'application directement dans le terminal dans lequel on lance la commande. Pour finir, il y a le paramètre «-P» qui indique que docker doit «pull» l'image spécifiée en paramètre (*dockersamples/static-site* dans notre cas) avant de «run» (exécuter) le container. [3]

4.3 Questions d'analyse sur la section 3.3

7. Expliquez la sortie de la commande docker images. Comment obtenir une version spécifique d'une image?

Cette commande affiche les images créées les plus récentes, ainsi que leur répertoire, *tag* et taille. On peut observer cela dans la figure 8. Ainsi, une image sera listé plusieurs fois si elle comporte plusieurs noms de répertoire ou de *tag*. Avec la colonne *tag*, il est possible d'avoir différentes versions de la même image et donc on peut faire une recherche en utilisant le nom du répertoire et le *tag* pour trouver une image avec une version spécifique en utilisant la commande:
docker images <Nom du répertoire>:<Version de l'image>. [3]

8. Expliquez avec vos propres mots la différence entre base images et child images.

Les «base images» sont des images de référence. Elles peuvent provenir d'images officielles, d'images officielles qu'on a personnalisées, ou encore d'images complètement personnalisées. Elles sont utilisées comme base pour créer des «child images». Dans ce même ordre d'idée, les «child images» sont des images qui sont construites à partir d'une «base image». [3] Dans la figure 12, la «base image» est la version 3.5 de alpine alors que l'image qu'on construit dans la figure 13 avec le «docker build» est une «child image».

9. Expliquez avec vos propres mots la différence entre official images et user images.

Les «official images» sont des images de Docker qui ont étaient soigneusement révisées avant d'être publiées. Les personnes qui révisent et qui maintiennent ses images sont une équipe chez Docker ou encore la communauté Docker sur Github à travers l'open source. Les «user images» qu'on a, elles, proviennent de monsieur et madame tout le monde et peuvent être publiées par n'importe qui. Les «official images» sont jugés meilleurs pour les débutants parce qu'elles sont bien documentées et elles sont optimisées pour les usages généraux. [3] Dans le cas de la version 3.5 de alpine, il s'agit

d'une image officielle en observant son profil sur le Dockerhub. En recherchant «alpine» sur ce site, on peut également trouver de nombreuses «user images». [6]

10. Expliquez avec vos propres mots ce qu'est un Dockerfile.

Il s'agit d'un fichier spécial qui contient des instructions successives que Docker peut interpréter afin de créer une image. Le fichier doit respecter une structure et une syntaxe particulière. En effet, chaque ligne d'instruction doit être sous la forme de <INSTRUCTION> <arguments> alors que les lignes de commentaire sont, tout simplement, celles qui commencent par '#'. La première instruction d'un Dockerfile doit obligatoirement être un «FROM» pour spécifier l'image de base avec comme argument le nom de l'image de base. La convention spécifie que les instructions doivent être en majuscule, mais cela n'est pas une obligation. Au lancement de la commande «docker build», il faut spécifier le chemin du répertoire contenant le fichier Dockerfile et celui-ci sera exécuté. On peut observer que dans la figure 13, on utilise un '.' pour spécifier le répertoire courant.

11. Expliquez chaque ligne du Dockerfile créé à l'étape 5 de la section 3.3.

En observant la figure 12, il est possible de voir chacune des lignes du Dockerfile.

La ligne 1: L'instruction «FROM» met en place l'image de base qui est la version 3.5 de alpine dans notre cas. [3]

La ligne 2: L'instruction «RUN» permet d'exécuter une commande. Dans notre cas, la commande exécutée sera «apk add --update py2-pip» qui provient de l'image de base (alpine:3.5). Cette commande ajoute le *package* py2-pip et l'option «--update» va télécharger la dernière version du *package*. [3]

La ligne 3: L'instruction «COPY» va prendre le fichier (1 paramètre) et le copier dans le répertoire de destination (2e paramètre) et donc la commande va placer le fichier «requirements.txt» sous le répertoire /usr/src/app. [3]

La ligne 4: Cette ligne va exécuter un «pip install» qui permet d'installer des librairies python. L'option «--no-cache-dir» permet de réduire la taille de l'image en ne sauvegardant que de façon temporaire les librairies installées. Ensuite, le paramètre «-r» permet de spécifier la location du fichier contenant les requirements c'est-à-dire le fichier contenant les librairies à installer. Comme on peut le voir dans la figure 10, la seule librairie à installer est la version 0:10:1 de Flask.

La ligne 5: Cette ligne copie le fichier app.py dans le répertoire /usr/src/app.

La ligne 6: Cette ligne copie le fichier index.html dans le répertoire /usr/src/app/templates/.

La ligne 7: L'instruction «EXPOSE» permet de spécifier que le "container" écoute le port spécifié. Dans notre cas, le «container» va écouter le port 5000. [3]

La ligne 8: L'instruction «CMD» permet d'exécuter une commande avec des paramètres. Ici, on exécute la commande «python» avec en paramètre le chemin vers «app.py» celui revient à appeler python /usr/src/app/app.py. Ce qui exécute le fichier app.py.

12. Expliquez ce qui se passe lorsque vous exécutez la commande docker build -t <YOUR_USERNAME>/flask-app

La commande «build» permet de construire l'image à partir du fichier Dockerfile dans notre cas. Ensuite, le paramètre «-t» permet de spécifier le répertoire (Docker) dans lequel sera entreposé l'image. Donc, comme on l'a entré à la figure 13, ce qui suit le paramètre «-t» (lampe14/flask-app)

sera le nom du répertoire. Enfin, le '.' à la toute fin de la commande signifie que l'on va prendre le contenu du répertoire courant (Ordinateur) afin de construire l'image.

4.4 Questions d'analyse sur la section 3.4

13. Décrivez chaque ligne du Dockerfile que vous avez créé. Ajoutez des captures d'écran et la sortie des commandes pour appuyer votre explication. Ajoutez également une image de votre navigateur montrant l'URL et le site Web en cours d'exécution.
N'oubliez pas de soumettre le code source que vous avez développé.

Ligne 1: Comme expliqué à la question 11 de la section 4.3, l'instruction «FROM» définit quelle image de base nous prenons. Dans notre cas, il s'agit du serveur Web nginx. Comme la version n'est pas précisée, Docker va prendre la *latest* version, c'est-à-dire la version la plus récente.

Ligne 2: La commande «COPY» ajoute des fichiers à partir du répertoire actuel du client Docker. Ici, nous copions le contenu du répertoire courant contenant «index.html» et «animation.js» dans le répertoire «/usr/share/nginx/html» qui est le répertoire à partir duquel «nginx» affiche le site Web.

4.5 Question de rétroaction

Nous travaillons à l'amélioration continue des travaux pratiques de LOG3000. Cette question peut être répondue très brièvement.

1. Combien de temps avez-vous passé au travail pratique, en heures-personnes, en sachant que
Deux personnes travaillant pendant trois heures correspondent à six heures-personnes.
Nous avons consacré environ 4 heures par personne sur ce TP, donc un total de 6 heures au total.
Selon nous c'est un temps tout à fait adéquat.

2. Quelles difficultés avez-vous rencontré lors de ce laboratoire?
Nous n'avons pas rencontré de difficultés majeures, mais seulement quelques légères difficultés dans la compréhension des commandes et des instructions en lien avec Docker. Cela est toutefois normal, puisqu'il s'agit d'une nouvelle technologie que nous n'avions jamais vu. Il y a assez d'informations et de documentations disponibles en ligne pour nous aider à affronter ces petits pépins.

Références

[1] https://training.play-with-docker.com/ops-s1-hello/

[2]https://grigorkh.medium.com/docker-for-beginners-part-2-running-your-first-container-7cb1ef829f79

[3] https://docs.docker.com/

[4] https://phoenixnap.com/kb/docker-image-vs-container

[5]https://go4hosting.in/knowledgebase/docker/difference-between-docker-image-and-container

[6] https://hub.docker.com/