ANDRIAMANANTENA Chrissy

FICHE DE PROCÉDURE

Docker est une plateforme de virtualisation légère et de gestion de conteneurs conçue pour simplifier le développement, le déploiement et la gestion d'applications. Elle repose sur la technologie de conteneurisation, qui permet d'emballer une application et toutes ses dépendances, y compris les bibliothèques et les fichiers système, dans un conteneur isolé et autonome.

Docker est un outil révolutionnaire qui offre aux développeurs et aux opérateurs informatiques la possibilité de créer des conteneurs, des unités logicielles autonomes, où une application et ses composants sont encapsulés, ce qui garantit une portabilité et une reproductibilité exceptionnelles. Ces conteneurs sont basés sur des images, qui sont des instantanés préconfigurés d'applications et d'environnements, facilitant ainsi la gestion des dépendances et des configurations.

Les avantages de Docker sont multiples :

Isolation : Chaque conteneur est isolé, ce qui signifie que les applications et les ressources s'exécutent de manière indépendante, sans interférer les unes avec les autres. Cela garantit la stabilité et la sécurité de l'environnement d'exécution. **Portabilité** : Les conteneurs Docker fonctionnent de manière cohérente sur différents environnements, que ce soit un poste de développement, un serveur de production ou même dans le cloud. Il en résulte une uniformité dans les déploiements et une facilité de migration. **Rapidité** : Docker permet un démarrage rapide des applications grâce à l'élimination de la surcharge associée aux machines virtuelles. Les conteneurs partagent le noyau du système d'exploitation de l'hôte, ce qui les rend extrêmement efficaces. **Scalabilité** : Vous pouvez facilement mettre à l'échelle les applications en ajoutant ou en supprimant des conteneurs en fonction des besoins, ce qui permet une gestion efficace des pics de charge. **Gestion des versions** : Les images Docker permettent de gérer les différentes versions de vos applications et de revenir à des versions précédentes en cas de problème, facilitant ainsi la gestion des mises à jour. **Écosystème riche** : Docker dispose d'un écosystème florissant de conteneurs préconfigurés prêts à l'emploi, ce qui accélère le processus de développement et de déploiement. **Intégration continue et déploiement continu (CI/CD)** : Docker s'intègre facilement dans les pipelines CI/CD, ce qui permet une automatisation complète du développement et du déploiement. **Gestion centralisée** : Docker offre des outils de gestion centralisée tels que Docker Compose et Kubernetes, permettant de gérer efficacement de grands clusters de conteneurs.

En résumé, Docker est une technologie de conteneurisation qui révolutionne la manière dont les applications sont développées, déployées et gérées. Elle offre une isolation, une portabilité, une rapidité et une flexibilité inégalées, ce qui en fait un outil indispensable pour les développeurs et les opérateurs cherchant à améliorer l'efficacité de leurs processus de développement et de déploiement d'applications.

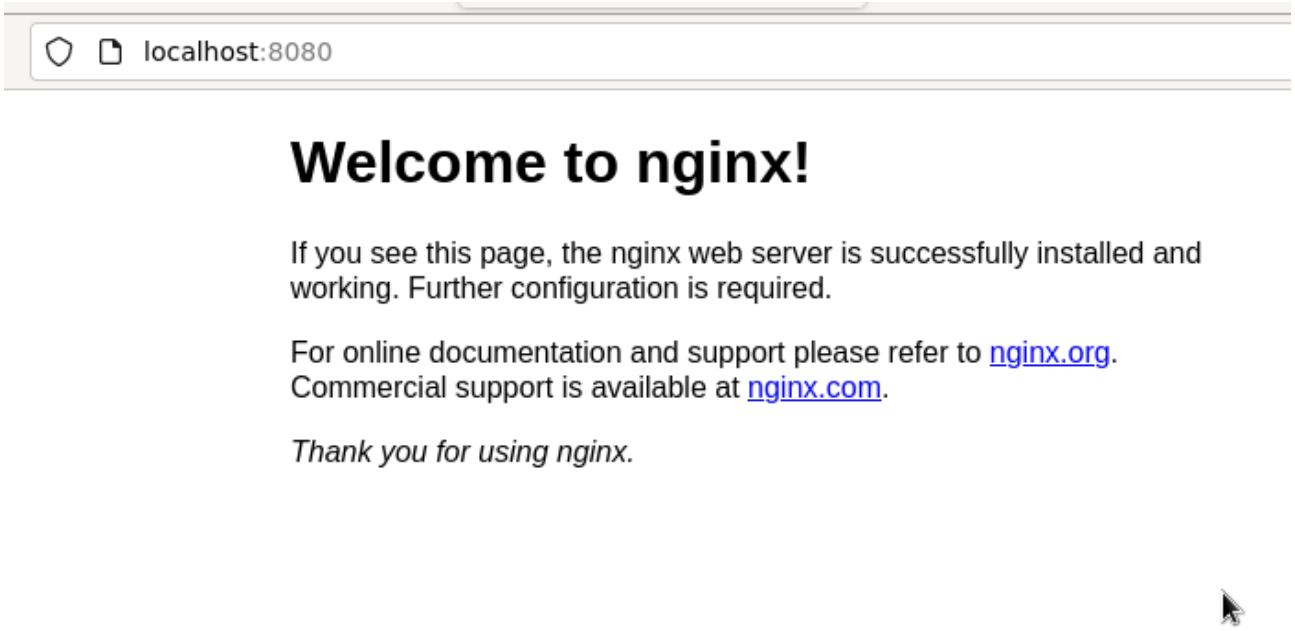
La découverte des commandes de base de Docker :

1. Récupérer une image de service : Docker pull image

(dans le cas du TP : docker pull nginx)

2. Pour vérifier les images récupérés : Doker images

3. `docker run -rm -p 8080:80 nginx`



4. La commande pour lister les conteneurs :

docker contenair ls

```
root@debian-xfce:~# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                   NAMES
3174ed7e1b5f   nginx     "/docker-entrypoint.…"   28 minutes ago   Up 28 minutes   0.0.0.0:8080->80/tcp, :::8080->80/tcp   amazing_mirzakhani
```

ID : 3174ed7e1b5f

Name : amazing_mirzakhani

5. La commande pour voir les conteneurs en cours :

Docker ps

6. Tout d'abord, pour exécuter un terminal interactif Bash à l'intérieur du conteneur NGINX :

docker exec -it mon_conteneur_nginx bash puis uname -a

```
Execute a command in a running container
root@debian-xfce:~# docker exec -it amazing_mirzakhani bash
root@3174ed7e1b5f:/# uname -a
Linux 3174ed7e1b5f 6.1.0-11-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.38-4 (2023-0
8-08) x86_64 GNU/Linux
```

7. Pour stoper le conteneur NGINX : faire la commande docker stop nom_conteneur

8. Pour revérifier que le conteneur a bien été stoper : docker ps

```
root@debian-xfce:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

9. Pour relancer le contenur NGINX : Docker run -d -p 8080:80 nginx

10. Après vérifications, ils n'ont ni le me ,nom, ni le même statut

11. Pour supprimer complétement le conteneur NGINX, il faut d'abord stoper le conteneur puis le supprimer à l'aide de la commande docker rm nom_du_conteneur

12. Il a bien dispau de la liste après cette commande

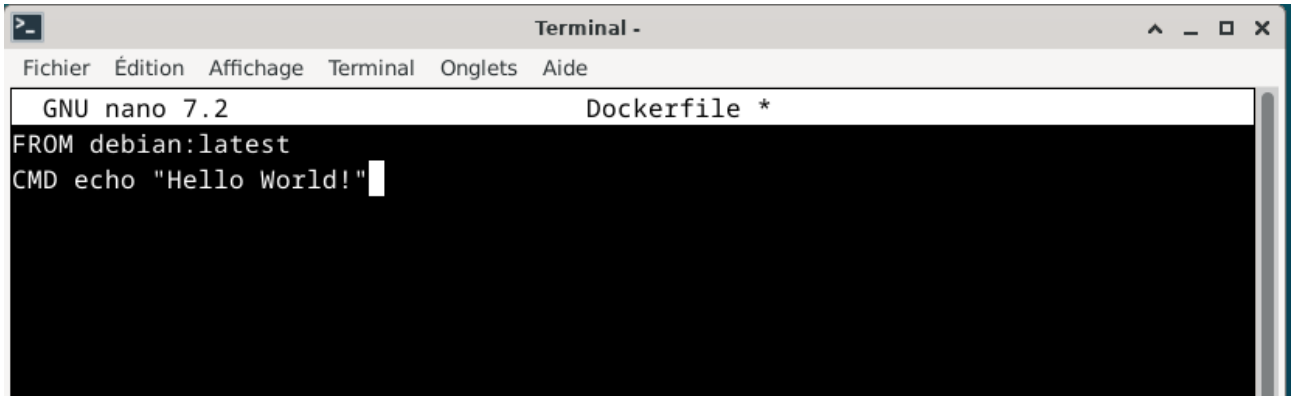
13. Pour supprimer l'image de NGINX : docker rmi nginx (il faut que tout les conteneurs soit supprimés pour ne pas avoir de message d'erreur).

14. Docker images pour revérifier les images de docker.

```
Deleted: sha256:9c7a54a9a43cca047013b82af109fe963fde787f63f9e016fdc3384500c2823d
Deleted: sha256:01bb4fce3eb1b56b05adf99504dafd31907a5aadac736e36b27595c8b92f07f1
root@debian-xfce:~# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

ETAPE 3 – Création des propres images Docker

1) Créer le Dockerfile avec la commande « nano my-hello-world»



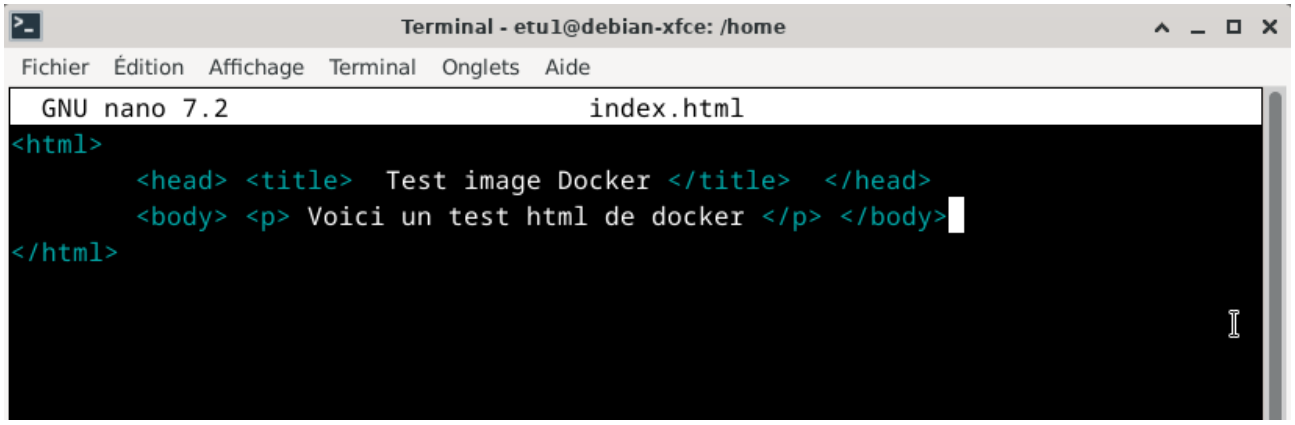
Construction de l'image docker :

```
root@debian-xfce:~# docker build -t my-hello-world
```

Lancement avec la commande « docker build -t my-hello-world »

2) Créer le fichier HTML avec la commande « nano index.html »

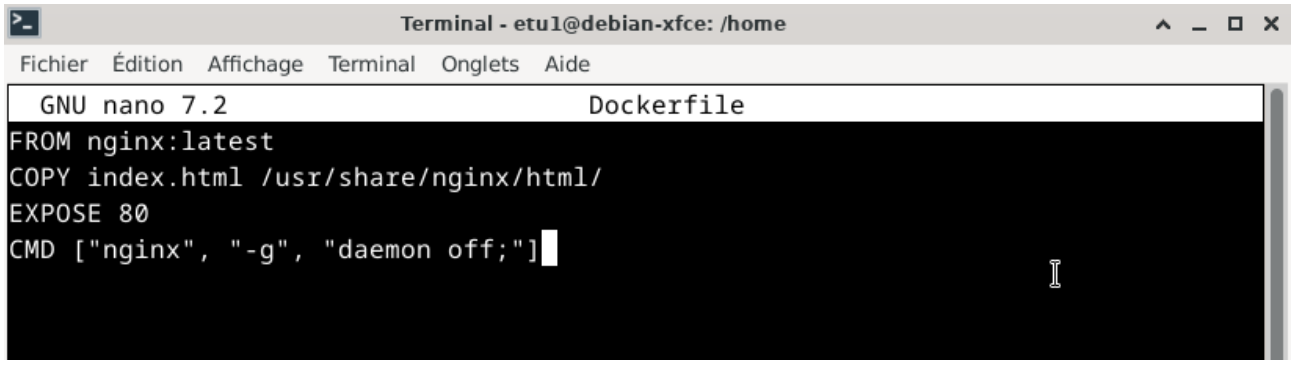
3) ajout de quelques lignes d'HTML :



4) Pour vérifie que le fichier HTML créé est vien présent dans le dossier :

cd chemin du dossier puis ls

5) Création du Dockerfile



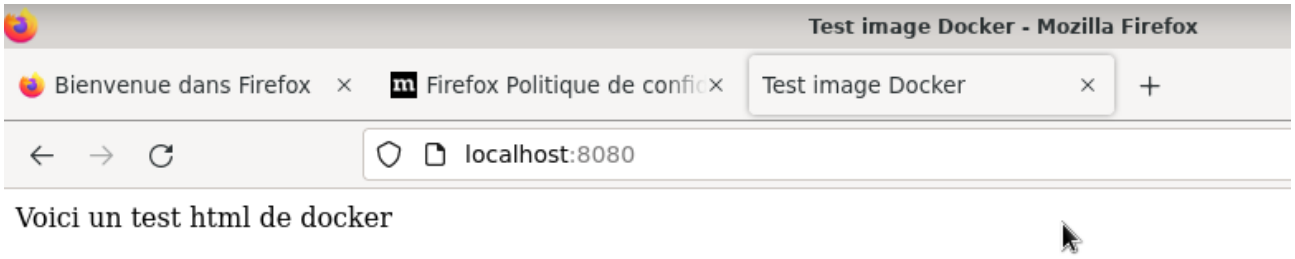
6)

exécution un conteneur basé sur cette image avec la commande suivante

```
root@debian-xfce:/home/etu1# docker run -d -p 8080:80 my-nginx
cdcd3efed6ca670d24276c63581259c7699cbce377d7bb99e2dd4ac36e1a9769
```

7)

Vérification dans le navigateur de la machine host



ETAPE 4 – Création des propres images Docker

Fichier de configuration :

version: '3.8'

services:

wordpress:

depends_on:

- "postgres"

image: ntninja/wordpress-postgresql:latest

restart: always

ports:

- 80:80

environment:

WORDPRESS_DB_HOST: postgres

WORDPRESS_DB_USER: postgres

WORDPRESS_DB_PASSWORD: postgres

WORDPRESS_DB_NAME: postgres

volumes:

- ./wp:/var/www/html

postgres:

image: postgres:10.5

restart: always

```
 environment:
 - POSTGRES_DB=postgres
 - POSTGRES_USER=postgres
 - POSTGRES_PASSWORD=postgres
 ports:
 - '5432:5432'
 volumes:
 - ./postgres-data:/var/lib/postgresql/data

volumes:
 db:
```

3) Lancer un conteneur précis :

Docker run –p 8080:80 –e “name”=name

4) Arréter un conteneur :

Docker stop “nom_conteneur”

5) Lancer un conteneur en arrière-plan :

Docker run –p 8080:80 –e “name”=name

6) Récupérer les logs :

Docker compose logs “id_conteneur”

7) Fermer le conteneur

Docker stop “nom_conteneur”

8) Création du fichier (yaml) :

```
version: '3'
services:
  wordpress:
    image: wordpress:latest
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress_password
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress_data:/var/www/html
    depends_on:
      - db
    networks:
      - wp_network

  db:
    image: postgres:latest
    environment:
      POSTGRES_DB: wordpress
      POSTGRES_USER: wordpress
      POSTGRES_PASSWORD: wordpress_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - wp_network

networks:
  wp_network:
    driver: bridge

volumes:
  wordpress_data:
  postgres_data:
```