Fiche de procédure Docker

Bethoule Voisin Joshua

1 : Docker ?

Docker est une plate-forme logicielle qui vous permet de concevoir, tester et déployer des applications rapidement. Docker intègre les logiciels dans des unités normalisées appelées conteneurs, qui rassemblent tous les éléments nécessaires à leur fonctionnement, dont les bibliothèques, les outils système, le code et l'environnement d'exécution.

2 : Découverte des commandes de base de Docker

Etape 1 :
Tout d'abord récupérer l'image du service de votre choix :
(exemple pour nginx : https://hub.docker.com/_/nginx)

Etape 2 :
Vérifier si l'image a bien été récupéré et qu'elle apparaît bien dans la liste :

```
docker images
```

Etape 3 :
Lancer votre container basé sur votre image en arrière plan avec un partage du port interne 80 sur le port externe de votre choix:

```
docker run -d -p port_au_choix:80 -name nom_image image
```

`-d` : exécute le container en arrière plan.
`-p` : mappe le port de votre choix au port 80 du container.
`-name mon_image` : nomme le container pour une identification plus facile.

Saisissez ensuite l'adresse de la machine dans un navigateur web avec le port saisi précédemment :

`http://IP_de_la_machine:port_choisis`

Etape 4 :
Récupérer la liste des container créé sur l'installation docker

```
docker ps -a
```

Cette commande affiche tous les containers, y compris ceux qui ne sont plus en cours d'exécution.

Etape 5 :
Vérifiez que le conteneur est bien en cours de fonctionnement en listant les processus de containers en cours.

```
docker ps
```

Cette commande liste uniquement les containers en cours d'exécution, si votre containers est présent dans la liste il est actif.

Etape 6 :
Exécuter cette commande pour créer un shell bash à l'intérieur du container :

```
docker exec -it mon_image bash
```

Pour tester le bash utiliser cette commande qui vous donnera l'OS actuelle que vous utilisez :

```
uname -a
```

Sortez ensuite du bash :

```
exit
```

Etape 7 :
Stopper votre container grâce à cette commande :

```
docker stop mon_image
```

Etape 8 :
Vérifier si celui ci est bien stopper :

```
docker ps
```

Etape 9 :
Relancer le container :

```
docker start mon_image
```

Etape 10 :
Exécuter à nouveau la commande pour verifier l'etat :

```
docker ps
```

Etape 11 :
Supprimer complètement le containers :

```
docker rm mon_image
```

(Pensez à stopper le containers pour que la suppression fonctionne)

```
docker stop mon_image
```

Etape 12 :
Vérifier ensuite que le container a belle et bien étais supprimer :

```
docker ps -a
```

Etape 13 :
Supprimer ensuite l'image télécharger plutôt :

```
docker rmi image
```

Etape 14 :
Et vérifier que celle ci a bien étais supprimer

```
docker images
```

3 : Création de vos propres images Docker

Etape 1 :
Crée en premier lieux un fichier Dockerfile contenant l'image de base (Ici comme exemple : Debian) et la commande à exécuter une fois le conteneur démarrer (Ici juste l'affichage de "Hello World !") :

Crée le fichier "Dockerfile" :

```
touch Dockerfile
```

Editer le fichier "Dockerfile" :

```
nano Dockerfile
```

Contenue du fichier "Dockerfile" :

```
FROM debian:latest
CMD ["echo", "Hello World!"]
```

Etape 2 :
Ensuite, vous pouvez construire l'image Docker en utilisant la commande suivante :

```
docker build -t nom_de_l'image .
```

`-t` : Donne un nom à l'image
`.` : Indique que le Dockerfile se trouve dans le répertoire courant

Etape 3 :
Vérifier que l'image créée est bien présente :

```
docker images
```

Si l'image créée plutôt est présente alors cela a bien fonctionné.

Etape 4 :

Exécutez l'image pour voir le message "Hello World !" s'afficher :

```
docker run nom_de_l'image
```

Etape 5 :

Maintenant nous allons lancer une image Nginx qui affichera une page HTML personnaliser :

Créé le fichier HTML :

```
touch index.html
```

Editer le fichier HTML :

```
nano index.html
```

Contenue simple pour une page HTML :

```
<html>
<head>
     <title>My Nginx Test</title>
</head>
<body>
     <h1>Hello from Nginx!</h1>
     <p>This is a simple HTML page served by Nginx inside
Docker.</p>
</body>
</html>
```

Etape 6 :

Vérifier que le fichier index.html contient bien les lignes de code HTML en utilisant une commande comme :

```
cat index.html
```

Etape 7 :
Passons à la création de l'image Nginx grâce à un dockerfile comme fait précédemment avec Debian :

Crée le fichier "Dockerfile" :

```
touch Dockerfile
```

Editer le fichier "Dockerfile" :

```
nano Dockerfile
```

Contenue du fichier "Dockerfile" :

(Utiliser NGINX comme image de base :)
```
FROM nginx:latest
```

(Copier le fichier HTML dans le répertoire de contenu par défaut de NGINX, a changer si vous n'avez pas le même :)
```
COPY index.html /usr/share/nginx/html/index.html
```

Etape 8 :
Construire l'image docker :

```
docker build -t my-nginx .
```

`-t` : Donne un nom à l'image
`.` : Indique que le Dockerfile se trouve dans le répertoire courant
`my-nginx` : nom de l'image (À changer selon vos préférences)

Etape 9 :
Vérifier que l'image créée est bien présente :

```
docker images
```

Si l'image créée plutôt (my-nginx) est présente alors cela a bien fonctionné.

Etape 10 :
Lancer un conteneur basé sur my-nginx :

```
docker run -d -p 8080:80 my-nginx
```

`-d` : lance le conteneur en mode détaché.
`-p 8080:80` : mappe le port 80 du conteneur au port 8080 de l'hôte.

Etape 11 :
Vérifier que le conteneur a bien démarré :

```
docker ps
```

Si le “Status” est “UP” alors celui-ci fonctionne bien.

Etape 12 :
Ouvrez un navigateur sur la machine hôte et accédez à l'adresse IP de la VM :

```
http://adresse_ip:8080
```

Si tout fonctionne comme prévu, vous devriez voir la page HTML que vous avez créée, avec le contenu suivant :

Hello from Nginx! This is a simple HTML page served by Nginx inside Docker.

4 – Automatisation du déploiement de vos conteneurs

Etape 1 :

Créez un fichier de configuration docker-compose.yml pour déployer le même conteneur NGINX que celui de l'étape 2.3.

Dans votre répertoire de travail, créez un fichier docker-compose.yml :

```
touch docker-compose.yml
```

Saisissez ensuite le contenue de votre fichier de configuration :

```
version: '3'
services:
  nginx:
     image: nginx:latest
     ports:
     - "8080:80"
     volumes:
     - ./index.html:/usr/share/nginx/html/index.html
```

`image: nginx:latest` : Permet d'utiliser la dernière image Nginx.

`ports:`
`- "8080:80"` : Permet de mapper le port 8080 de la machine hôte au port 80 du conteneur.

`volumes:`
`- ./index.html:/usr/share/nginx/html/index.html` : permet de monter le fichier index.html dans le répertoire par défaut de NGINX pour servir la page web.

Etape 2 :

La commande suivante permet de récupérer les images mentionnées dans le fichier docker-compose.yml sans lancer les conteneurs :

```
docker compose pull
```

Etape 3 :

Pour lancer le conteneur défini dans docker-compose.yml au premier plan (attacher le terminal au conteneur), utilisez la commande suivante :

```
docker compose up
```

Cette commande télécharge les images s'il le faut et lance le conteneur et affiche les logs en temps réel.

Ensuite, ouvrez un navigateur et accédez à l'URL suivante :

```
http://adresse_ip_VM:8080
```

Vous devriez voir la page par défaut de NGINX.

Etape 3 :

Stopper ensuite le conteneur grâce à cette commande :

`docker compose stop` ou `ctrl + c`

Etape 4 :

Pour lancer le conteneur en arrière-plan (mode détaché), utilisez l'option `-d` :

```
docker compose up -d
```

Etape 5 :
Pour afficher les logs du conteneur lancé en arrière-plan en temps réel, utilisez la commande suivante :

```
docker compose logs -f
```

`-f` : permet de voir les nouveaux logs en direct.
Sorter ensuite grace a `ctrl + c`

Etape 6 :
Pour arrêter proprement les conteneurs lancés en arrière-plan, utilisez :

```
docker compose down
```

Cette commande arrête et supprime les conteneurs ainsi que les réseaux créés lors du lancement.

Etape 7 :
Nous allons maintenant créer un fichier de configuration permettant d'obtenir une installation WordPress + PostgreSQL fonctionnelle :
Commencez par créer un répertoire de travail pour votre projet WordPress + PostgreSQL :

`mkdir wordpress-postgres` : permet de crée un dossier nommé "wordpress-postgre"
`cd wordpress-postgres` : permet d'entrer dans ce même dossier

Etape 8 :

Créer maintenant le fichier docker-compose.yml

```
touch docker-compose.yml
```

Et entrer son contenue pour Wordpress + Postgres :

```
version: '3'
services:
  db:
     image: postgres:latest
     container_name: postgres_db
     environment:
     POSTGRES_DB: wordpress
     POSTGRES_USER: wordpress_user
     POSTGRES_PASSWORD: wordpress_password
     volumes:
     - postgres_data:/var/lib/postgresql/data
     networks:
     - wordpress_network

  wordpress:
     image: wordpress:latest
     container_name: wordpress_app
     ports:
     - "8080:80"
     environment:
     WORDPRESS_DB_HOST: db:5432
     WORDPRESS_DB_USER: wordpress_user
     WORDPRESS_DB_PASSWORD: wordpress_password
     WORDPRESS_DB_NAME: wordpress
     volumes:
     - wordpress_data:/var/www/html
     depends_on:
     - db
```

```
    networks:
    - wordpress_network

volumes:
  postgres_data:
  wordpress_data:

networks:
  wordpress_network:
```

Explication du fichier docker-compose.yml :

`version: '3'` : Spécifie la version de Docker Compose.
`services` :

`db` : Le service pour PostgreSQL.

`image: postgres:latest` utilise la dernière image de PostgreSQL.

`environment :` Des variables d'environnement sont définies pour le nom de la base de données, l'utilisateur, et le mot de passe.

`volumes` : Monte un volume pour persister les données de la base de données dans /var/lib/postgresql/data.

`wordpress` : Le service pour WordPress.

`image: wordpress : latest` utilise la dernière image officielle de WordPress.

`ports:`
`- "8080:80"` Le conteneur expose le port 80 à l'extérieur via le port 8080 de l'hôte (VM).

`environment :` Les variables d'environnement permettent à WordPress de se connecter à PostgreSQL.

`volumes` : Monte un volume pour persister les données de WordPress.

`depends_on` : Assure que le conteneur de la base de données est démarré avant celui de WordPress.

`volumes` :

`postgres_data` : Pour sauvegarder les données de PostgreSQL.

`wordpress_data` : Pour sauvegarder les données du site WordPress.

`networks` :

Créer un réseau Docker pour que les services puissent communiquer entre eux.

Etape 9 :

Utilisez la commande suivante pour télécharger les images spécifiées dans le fichier docker-compose.yml sans démarrer les conteneurs :

```
docker compose pull
```

Pour démarrer les conteneurs et afficher leurs logs dans la console (au premier plan), exécutez :

```
docker compose up
```

Et enfin acceder a wordpress depuis votre navigateur :

```
http://adresse_ip_VM:8080
```

FIN