TP : Docker

Partie 1 : les bases (2h00)

L'objectif de cette séance est de vous faire découvrir Docker et comment l'utiliser pour déployer un projet Symfony

Les blocs de commentaires sont réservés à des question à répondre.

Concepts

Pour cette première partie, il va falloir définir les différents concepts :

1. LXC, c'est quoi ?
2. Docker, c'est quoi ?
3. Qu'est-ce qu'un container ?
4. Quelle est la différence entre une Image et un Container ?

Hello container

Pour la premère étape on va lancer un Container qui va afficher Hello World.

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

Si tout se passe bien, il affiche le contenu suivant:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:f9dfddf63636d84ef479d645ab5885156ae030f611a56f3a7ac7f2fdd86d7e4e
Status: Downloaded newer image for hello-world:latest

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

Plusieurs étapes sont importantes dans ce process :

```
Unable to find image 'hello-world:latest' locally
```

Docker n'a pas trouvé d'image locale appelée `hello-world:latest`.

À quoi correspond le `:latest` du nom de l'image ?

```
latest: Pulling from library/hello-world
```

```
1b930d010525: Pull complete
Digest:
sha256:f9dfddf63636d84ef479d645ab5885156ae030f611a56f3a7ac7f2fdd86d7e4e
Status: Downloaded newer image for hello-world:latest
```

Qu'est ce qu'un Digest ?

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

Pour voir les containers qui tournent actuellement, vous pouvez faire

```
sudo docker ps
# alias de
sudo docker container ls
```

Pourquoi ne voit-on pas notre container ?

Pour voir la totalité des containers créés sur la machine, lancez la commande suivante:

```
sudo docker ps -a
```

Vous devriez voir le contenu suivant:

```
CONTAINER ID        IMAGE                         COMMAND
CREATED             STATUS                       PORTS       NAMES
40c279dd8e71        hello-world                   "/hello"                 9
minutes ago       Exited (0) 9 minutes ago                 keen_chaplygin
```

Le résultat de cette commande indique plusieurs choses :

- CONTAINER ID: identifiant de conteneur unique
- IMAGE: l'image utilisée
- COMMAND: la commande lancée dans le conteneur
- CREATED: la date de création
- STATUS: l'état du conteneur
- PORTS: les ports exposés (ici rien)
- NAMES: les noms donnés au conteneur (par défaut ce nom est aléatoire)

Nous allons lancer de nouveau un container mais cette fois en lui spécifiant un nom :

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

retournez voir la liste de tous les conteneurs et le "name" à changé.

Que se passe-t-il si j'essaie de réexecuter la même commande une seconde fois ? Pourquoi ?

Hello nginx

Pull de l'image

pour la suite de ce TP, on va avoir besoin de télécharger l'image nginx. Pour changer, executez la commande suivante:

```
docker pull nginx:1.17
```

Pour voir les images téléchargées sur la machine, vous pouvez éxecuter la commande :

```
sudo docker images
# alias de
sudo docker image ls
```

De la même manière que `docker container ls` on retrouve une liste d'images docker

```
REPOSITORY                                                  TAG
IMAGE ID            CREATED             SIZE
nginx                                                       1.17
ed21b7a8aee9        13 days ago         127MB
hello-world                                                 latest
fce289e99eb9        15 months ago       1.84kB
```

Avec les champs suivants :

- REPOSITORY: le nom de l'image
- TAG: son tag/version
- IMAGE ID: son identifiant unique
- CREATED: sa date de création

Tooltip: `sudo -s` pour éviter d'avoir à préfixer toutes ses commandes par `sudo`

À partir de ce moment, on va créer plusieurs instances nginx.

Container Nginx

lancez la commande suivante

```
docker run -d --name mysite nginx:1.17
```

un nouvel argument a été ajouté :

- `-d` : en mode détaché

Qu'est ce que le mode détaché?

en affichant la liste des containers qui tournent, on obtient les informations suivantes:

```
# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
1fe32a24c152        nginx:1.17          "nginx -g 'daemon of…"   3 seconds
ago       Up 2 seconds        80/tcp              mysite
```

On va arrêter le container en faisant la commande suivante :

```
docker stop mysite
```

En faisant la commande `docker ps` on ne voit plus le container, mais avec l'argument `-a` on le retrouve en status éteint.

On va également le supprimer en faisant:

```
docker rm mysite
```

Le container n'existe plus.

Pour gagner du temps, on va rajouter l'option `--rm` qui va supprimer automatiquement le container dès qu'il est arrêté :

```
docker run --rm -d --name mysite nginx:1.17
```

Accéder au container via l'adresse IP

:warning: n'ayant pas de mac, je n'ai pas pu tester cette partie.

Pour pouvoir accéder au container, il faut récupérer son adresse IP pour ça on va inspecter le container:

```
docker container inspect mysite
```

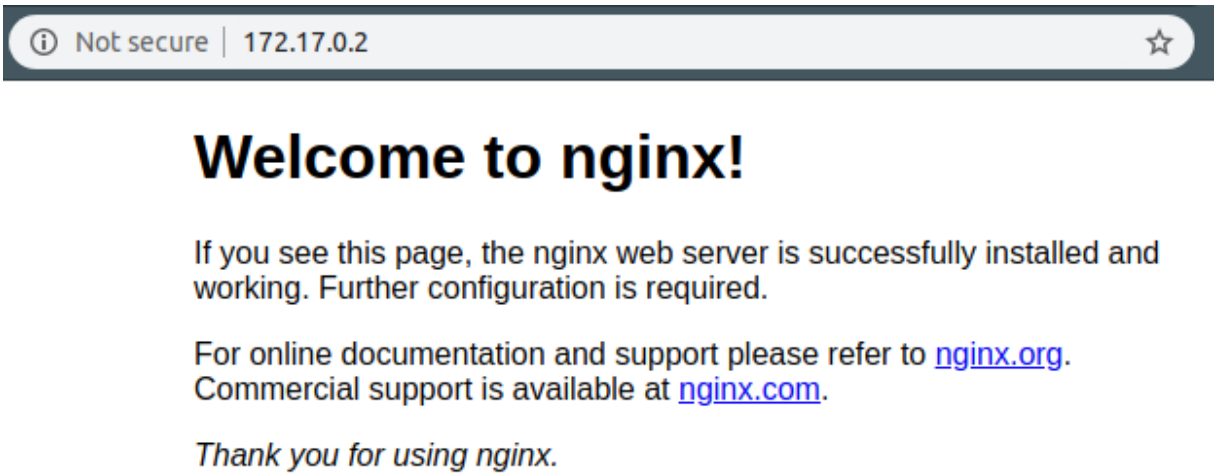
Cela va sortir toutes les informations au sujet du container.

Quelles sont les informations qui sont renvoyées lorsqu'on inspecte le conteneur?

La partie du json qui nous interesse est la clé `NetworkSettings.Networks.bridge.IPAddress`.

Tooltip: la commande `docker inspect mysite -f "{{.NetworkSettings.Networks.bridge.IPAddress}}"` permet de gagner du temps :-)

Dans votre navigateur allez à l'adresse IP indiquée. Si tout se passe bien, vous aurez la page suivante :



C'est la seule fois dans tout ce TP que vous allez utiliser l'adresse IP d'un container directement !

Accéder au container via l'adresse IP locale

On va arrêter et supprimer le container précédent. On va ajouter l'argument `-P` (P majuscule) à la commande comme suit:

```
docker run --rm -d -P --name mysite nginx:1.17
```

on optient quelque chose de ce genre :

```
# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                   NAMES
10b5fc5ddefe        nginx:1.17          "nginx -g 'daemon of…"   3 seconds
ago       Up 2 seconds        0.0.0.0:32769->80/tcp   mysite
```

L'information importante est l'information renseignée par le port, ici `32769`. Cela permet d'aller directement sur l'adresse http://localhost:32769 et d'obtenir le même résultat.

En spécifiant le port cette fois

On va arrêter et supprimer le container précédent. On souhaite définir nous même le port d'execution, pour cela on va modifier l'instruction `-P` par `-p 8081:80`

```
docker run --rm -d -p 8081:80 --name mysite nginx:1.17
```

rendez-vous maintenant sur l'adresse http://localhost:8081

Quelle est la différence entre -P et -p ?

Custom web page

On va maintenant déployer une page HTML de notre choix avec Docker

Créez un dossier dans un répertoire de votre ordinateur et allez dans le dossier avec votre terminal.

Exécutez la commande suivante:

```
mkdir -p $PWD/site && echo '<h1>hello World</h1>' > $PWD/site/index.html
```

Cette commande va créer un sous-dossier `site` dans le répertoire courant avec un fichier index.html.

Tooltip: $PWD - variable d'environnement du dossier courant

Pour tester `echo $PWD`

À partir de maintenant, on va modifier la commande de lancement du Container pour partager le dossier `site` avec le dossier `/usr/share/nginx/html/`

```
docker run  --rm -d -p 8081:80 -v $PWD/site:/usr/share/nginx/html/  --name mysite nginx:1.17
```

retournez sur http://localhost:8081

Partie 2 : Docker compose (2h00)

Docker-compose est une solution permettant d'orchestrer plusieurs conteneurs docker.

Qu'est-ce que docker-compose ? Quels sont les 4 principales informations contenues dans un fichier docker-compose ? Qu'est-ce qu'un orchestrateur ?

Pour cet exemple, nous allons utiliser un fichier docker-compose.yaml pour lancer un service Nginx, avec PHP-FPM et une base de données MySQL pour faire tourner une application Symfony.

Clone du projet

Pour commencer, on va cloner le projet suivant :

```
git clone https://gogs.h91.co/Courses/symfony-mini-blog.git
```

On va travailler uniquement dans ce dossier, donc ouvrir un terminal à cet endroit.

Modifier le fichier .gitignore pour retirer les 3 lignes suivantes :

```
docker-compose.yml
Dockerfile
nginx.conf
```

Créer ensuite un projet sur github et changer la remote comme suit :

```
git remote set-url origin https://github.com/USERNAME/REPOSITORY.git
```

Docker Compose

MySQL

Pour cette première étape, nous allons faire tourner un serveur MySQL.

Voici le squelette de docker-compose.yml :

```yaml
version: "3.4"
volumes:
  dbstorage: ~
services:
  database:
    image: mysql:8.0
    volumes:
          - dbstorage:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_DATABASE=db
    ports:
      - 13306:3306
```

Expliquez chaque parties du docker-compose.yaml

Pour démarrer le conteneur, exécutez la commande :

```
docker-compose up
```

En utilisant phpstorm essayez d'accéder à la base de données.

Comment se connecter à cette base de données ?

PHPMyAdmin

pour faciliter l'accès à la base de données on va ajouter un service phpmyadmin

```yaml
phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      - PMA_HOST=database
      - PMA_USER=root
      - PMA_PASSWORD=password
    ports:
      - 18080:80
```

Rendez-vous à l'adresse http://localhost:18080/

Pourquoi ne faut-il pas deployer ce phpmyadmin en production ?

Arrêtez l'execution (Ctrl+C) et relancez la commande précédente en ajoutant l'argument `-d`.

La liste des commandes utiles pour docker :

- `docker-compose up -d`
- `docker-compose down`
- `docker-compose up -d --force-recreate`
- `docker-compose stop database`
- `docker-compose ps`
- `docker-compose logs -f --tail=10 phpmyadmin`

Définir ce que chaque commandes faits

PHP-FPM

L'image PHP que nous allons utiliser s'appelle php fpm. Le problème de cette image c'est qu'elle vient sans aucune des extensions PHP de base, et pour cela, il va nous falloir créer notre propre image à partir de l'image PHP-FPM de base.

Créez un fichier nommé `Dockerfile` à la racine du projet :

```
FROM php:7.4-fpm

ADD https://getcomposer.org/download/1.10.22/composer.phar
/usr/bin/composer

RUN chmod a+x /usr/bin/composer

RUN apt-get update && \
    apt-get install -y git libcurl3-dev libzip-dev

RUN docker-php-ext-install pdo pdo_mysql curl zip
WORKDIR /my_app
```

Qu'est ce qu'un Dockerfile ? Que fait celui-ci?

Expliquez le principe d'héritage de docker.

Ajoutez maintenant le service `php` déclaré de cette manière

```
php:
  build: .
  volumes:
    - ./:/my_app
```

Nginx

Pour Nginx on va reprendre la config de l'exercice précédent mais au format docker-compose.yml.

```
docker run  --rm -d -p 8081:80 -v $PWD/site:/usr/share/nginx/html/  --name
mysite nginx:1.17
```

Voici la configuration de l'exercice d'avant :

```
version: "3.4"
services:
  nginx:
    image: nginx:1.17
    container_name: mysite
    ports:
      - 8081:80
    volumes:
      - ./site:/usr/share/nginx/html/
```

à partir de maintenant, l'exercice consiste à transformer ce docker-compose pour l'adapter à notre exercice.

en premier temps créer un fichier `nginx.conf` avec le contenu suivant:

```
server {
    listen 80;
    root    /my_app/public;
    include /etc/nginx/default.d/*.conf;

    index index.php index.html index.htm;

    client_max_body_size 64m;
    charset utf-8;

    location / {
        try_files $uri /index.php$is_args$args;
    }
    error_page 404 /index.php;

    location ~ ^/index\.php(/|$) {
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        fastcgi_param DOCUMENT_ROOT $realpath_root;
        include fastcgi_params;
        internal;
    }

    location ~ \.php$ {
        return 404;
    }
}
```

Voici les consignes:

- retirer le dossier `./site` du partage
- partager le fichier `./nginx.conf` avec le dossier ``/etc/nginx/conf.d/default.conf`
- partager le dossier public du projet avec le dossier `/my_app/public`
- remplacer le port exposé avec le port `80` du conteneur pour le port `18081`

Configuration

Si tout fonctionne, il est possible de voir le site à la page : http://localhost:18081/

On va maintenant installer les dépendences du projet via docker.

Pour ça, on va executer la commande suivante :

```
docker-compose exec php composer install
```

Cela va installer toutes les dépendences de composer. On va également executer les migrations de la même manière

```
bin/console doctrine:migration:migrate
```

Enfin

Si tout fonctionne, il est possible de voir le site à la page : http://localhost:18081

Il est possible de tester cette manifique application de blog.

Pour aller plus loin

Voici quelques questions pour pousser le sujet un peu plus loin

- Qu'est-ce que docker swarm ?
- Qu'est-ce que Kubernetes ? Quel lien avec Docker ?