

Utilisation de Docker Compose

⚠ Pré-requis. Ce sujet nécessite les sujets :

- Docker
- Docker 2

1 Introduction

i Docker compose. Après avoir travaillé avec Docker, les images et les conteneurs, il est évident que l'installation d'un logiciel complexe nécessite souvent plusieurs opérations de déploiement et la mise en place de plusieurs conteneurs. **Docker compose** permet justement de spécifier, fabriquer et lancer plusieurs conteneurs. Nous allons découvrir ces fonctionnalités au travers d'un exemple.

2 Préparation

Comme pour la construction d'image, **Docker compose** a besoin d'un répertoire, d'un fichier de configuration (`compose.yaml`) et de ressources afin de déployer une application. **Attention** : dans un fichier YAML, l'indentation (assurée par deux espaces) marque les relations père-fils entre les paramètres.

i Note : Vous trouverez une description précise de toutes les options du fichier de configuration dans cette documentation.

Préparer une application vide

```
## Créer un répertoire et le choisir
mkdir movies-app
cd movies-app

## Créer le fichier compose.yaml
cat > compose.yaml <<FIN

name: movies-app

# Cette application sera composée de trois parties
include:
  - database.yaml
  - web.yaml
  - front.yaml

# La partie réseau pour connecter les parties
networks:
  mynet:
    driver: bridge

FIN

## Les trois parties sont vides
cat /dev/null > database.yaml
cat /dev/null > web.yaml
cat /dev/null > front.yaml
```

Nous allons utiliser les commandes `docker compose build` pour construire (mais il n'y a rien à construire), `docker compose ps` pour lister (mais rien à lister) et `docker compose rm` pour supprimer (idem).

Construire l'application - rien pour l'instant

```
## Construire les images
docker compose build

## Lister les services
docker compose ps

## Stopper les services
docker compose rm --stop --force
```

3 La partie données

Nous allons ajouter une base de données à notre application. Nous retrouvons les paramètres que nous utilisons sur la ligne de commande dans les exercices précédents.

Note : L'image `mysql` n'offre pas de `healthcheck`. Nous devons donc le définir.

La partie MySql

```
## Créer un répertoire pour la DB
mkdir -p /var/movies/db

## Créer le fichier database.yaml
cat > database.yaml <<FIN

services: ## Définition des services

  ## La partie données
  database:
    image: mysql
    ## Toujours redémarrer
    restart: always
    ## Les paramètres de ma BD
    environment:
      MYSQL_ROOT_PASSWORD: rootpasswd
      MYSQL_USER: moviesuser
      MYSQL_PASSWORD: moviespass
      MYSQL_DATABASE: moviesdb
    ## Pour pérenniser les données dans /var/movies/db
    volumes:
      - /var/movies/db:/var/lib/mysql
    ## Pour communiquer avec les autres conteneurs
    ## qui sont dans le même réseau.
    networks:
      - mynet
    ## Pour tester l'état de ma BD
    healthcheck:
      test: /usr/bin/mysql --user=root --password=rootpasswd \
        --execute "SHOW DATABASES;"
      interval: 2s
      timeout: 20s
      retries: 100

FIN
```

Utilisons maintenant `docker compose up` pour démarrer notre application.

Lancement de l'application

```
## Lancer les services et attendre le statut healthy  
docker compose up -d --wait --build  
  
## Lister les services  
docker compose ps  
  
## Attendre les services - normalement inutile  
wait-for-healthy-container movies-app-database-1 600  
  
## Stopper les services  
docker compose rm --stop --force
```

Note : Vous remarquerez que le nom des conteneurs (par exemple `movies-app-database-1`) est construit à partir du nom de l'application (`movies-app`), du nom du service (`database`) et d'un numéro d'ordre (`1`).

4 La partie WEB

Continuons avec la partie WEB (deux instances de notre application Spring-Boot) :

La partie WEB pour compose

```
## Créer un répertoire pour les logs
mkdir -p /var/movies/logs

## Créer le dockerfile
cat > web.yaml <<FIN

services: ## Services definition

  ## Partie WEB
  web:
    build: web
    ## deux instances
    scale: 2
    expose:
      - "8081"
    restart: always
    ## Le WEB va attendre la BD
    depends_on:
      database:
        condition: service_healthy
    networks:
      - mynet
    ## Pour ne pas perdre les traces
    volumes:
      - /var/movies/logs:/var/tmp

FIN
```

Lancement de l'application

```
## Lancer les services et attendre le statut healthy
docker compose up -d --wait --build

## Lister les services
docker compose ps

## Attendre les applications WEB - normalement inutile
wait-for-healthy-container movies-app-web-1 600
wait-for-healthy-container movies-app-web-2 600

## Tester les deux services
docker run --network movies-app_mynet almalinux \
  curl -sL movies-app-web-1:8081 | fgrep '1983'
docker run --network movies-app_mynet almalinux \
  curl -sL movies-app-web-2:8081 | fgrep '1983'

## Stopper les services
docker compose rm --stop --force
```

Note : Il est possible que vous trouviez deux films en 1983. En effet, chaque instance de l'application Spring-Boot va initialiser la BD avec les trois premiers films de la série. Visiblement, cette application n'a pas été prévue pour un déploiement en plusieurs instances.

5 La partie Front Proxy

Nous avons deux applications WEB qui fonctionnent en parallèle mais il faut maintenant ajouter un proxy qui va assurer le rôle de répartition des accès (`load-balancing`). Nous allons utiliser HAProxy.

La partie Proxy

```
## Préparer les fichiers de configuration de HAProxy
wget http://tinyurl.com/jlmassat2/cca/proxy.zip
unzip proxy.zip

## Préparer le fichier front.yaml
cat > front.yaml <<FIN

# Services definition
services:

  ## La partie Proxy
  proxy:
    build: proxy/
    ports:
      - 8888:80
    restart: always
    networks:
      - mynet
    environment:
      DOMAIN_NAME: idl.xfr
      EMAIL_ADDRESS: root@idl.xfr
      BACKEND1_URL: movies-app-web-1:8081
      BACKEND2_URL: movies-app-web-2:8081
    depends_on:
      web:
        condition: service_healthy

FIN
```

►► **Travail à faire :** Lire, analyser et comprendre les fichiers qui se trouvent dans le répertoire `proxy` .

Lancement de l'application

```
## Construire/Lancer/Attendre les services
docker compose up -d --wait --build

## Lister les services
docker compose ps
```

Test du proxy

```
## Tester les services
curl -L http://localhost:8888 | fgrep 'Star'

## Vérifier le round-robbin du proxy
## SERVERID doit indiquer une app.
curl -i http://localhost:8888
## SERVERID doit indiquer l'autre app.
curl -i http://localhost:8888
```

Stopper l'application

```
## Stopper les services  
docker compose rm --stop --force
```

▶▶ **Travail à faire :** Modifier votre application afin d'avoir trois applications WEB et un bon équilibrage de la charge entre ces trois composants.