

Docker

Lancement de conteneurs, modes attachés et détachés ...

Nous allons exécuter un serveur mysql dans un conteneur, en utilisant l'image officielle disponible sur le [docker hub](#). La configuration du serveur se fait (entre autres) en passant au conteneur des variables d'environnement.

- lancez mysql dans un conteneur avec la commande `docker run [OPTIONS] IMAGE [COMMAND] [ARG...]`. Vous devrez spécifier que :
 - le mot de passe de l'utilisateur root, avec la variable d'environnement `MYSQL_ROOT_PASSWORD` (option `-e CLE=VALEUR`)
 - l'exécution doit se faire en mode détaché, avec la possibilité de se rattaché par la suite (options `-dit`)
 - le port interne au conteneur (3306) doit être redirigé vers un port de la machine hôte (3306 aussi, sauf en cas de conflit) (option `-p port_hote:port_container`)
 - l'image servant à construire le container (`mysql`)
- vérifiez que le container est lancé et son état avec les commandes `docker ps`, `docker logs <container_id>` et `docker container inspect <container_id>`

On veut maintenant accéder au serveur mysql s'exécutant dans le conteneur. Il suffit pour cela de spécifier l'url du serveur (`localhost:<port_hote>`) à n'importe quel client mysql (en ligne de commande, MySQLWorkbench, ...). Nous allons utiliser le client en ligne de commande inclut dans le container.

- avec `docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]`, exécutez la commande `mysql -uroot -p` dans le container. Vous devrez spécifier :
 - mode interactif (options `-it`)
 - le container dans lequel exécuter la commande (l'id de votre container)
 - la commande à exécuter (`mysql -uroot -p`) Après avoir entré votre mot de passe, vous avez accès à la console mysql (`quit;` pour quitter).

Arrêtez et supprimez votre container.

Création d'images avec un `Dockerfile`

Dans le repertoire de votre choix, clonez le dépôt git (url : `<url depot message>`) correspondant au microservice `message`. Nous allons "conteneuriser" ce microservice, c'est à dire créer une image permettant de l'exécuter dans un conteneur.

- créez, dans le répertoire du projet, un fichier `Dockerfile` avec les étapes suivantes :
 - à partir d'une image `openjdk:11-jdk` (instruction `FROM`)
 - copiez le livrable (le `.jar`) (instruction `COPY`)
 - définissez la commande à exécuter dans le container (instruction `ENTRYPOINT`). La commande est la suivante : `java -jar <fichier.jar>`
- construisez l'image avec la commande `docker build [OPTIONS] PATH | URL | -`, en indiquant :
 - avec l'option `-t`, le nom à donner à l'image

- le repertoire à utiliser comme contexte pour la construction de l'image (le repertoire du projet, là où vous avez enregistré le Dockerfile)
- vérifiez que l'image à bien été créée avec la commande `docker image ls`
- executer l'image, en spécifiant :
 - mode détaché
 - la redirection de port (l'application utilise le port 8081)
 - le nom de l'image à executer
- vérifiez que l'application est accessible, en vous connectant avec votre navigateur sur l'url `localhost:<port_hote>`
- arrêtez et supprimez votre container.

On pourra améliorer notre dockerfile, en suivant les suggestions faites [ici](#).

Mise en place et utilisation d'un registre

Nous allons maintenant créer un registre pour enregistrer nos images. Ce registre sera en local et non sécurisé. Vous trouverez dans la [documentation](#) des indications pour configurer un registre pour une utilisation en production.

- lancez un container avec l'image `registry:2` en mode détaché, avec la redirection de port (5000)
- ajoutez, avec la commande `docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]` un tag additionnel pour votre image. le premier parametre correspond au nom de votre image, le second sera de la forme `registry_url/new_name[:tag]`
- envoyez l'image sur le registre docker avec la commande `docker push [OPTIONS] NAME[:TAG]`

Création de services avec `docker-compose`

Nous voulons maintenant executer notre application complète (base de donnée et les trois microservices) grâce à `docker-compose`. Nous allons pour cela :

- créer des images pour chacun des microservices, puis les publier sur le registre
- écrire un fichier `docker-compose.yml` décrivant les containers à lancer et leurs configurations
- lancer les services avec `docker-compose`.

Création des images pour les microservices

Reproduire, pour les microservices `name` et `hello` les étapes faites précédemment. Vous pouvez utiliser le même Dockerfile, et simplement refaire pour chaque microservice :

- création de l'image avec `docker build [OPTIONS] PATH | URL | -`
- publication de l'image avec `docker push [OPTIONS] NAME[:TAG]`

Fichier `docker-compose.yml`

Créez un fichier `docker-compose.yml` et indiquer :

- sa version (3)
- les services :
 - message :

- basé sur l'image que vous avez créée
- redirection de port 8081 -> 8081
- name
 - basé sur l'image que vous avez créée
 - redirection de port 8082 -> 8082
 - dépend de name_db
- name_db
 - basé sur l'image mysql
 - variables d'environnement pour spécifier le mot de passe de l'utilisateur root (mettez **root**) et la création d'une base de données **test**
 - volumes pour rendre persistant les données
- hello
 - basé sur l'image que vous avez créée
 - redirection de port 8083 -> 8083
 - dépend de name et message
- les volumes : le volume utilisé par name_db

Lancement des services

Lancez les services avec la commande **docker-compose up**. Vous pouvez voir quels services sont exécutés **docker-compose ps**, puis arrêter les services avec **docker-compose down**.

Pour aller plus loin, on pourrait créer un network, associer tous nos services à ce network et ne rendre visible de l'extérieur que hello.