Lancement de conteneurs, modes attachés et détachés ...

Lancement de conteneurs

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'execution doit se faire en mode détaché, avec la possibilité de se ratacher par la suite (options `-dit`)
  - le port interne au conteneur (3306) doit être redirigé vers un port de la machine hote (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>`

Executer une commande dans un container

On veut maintenant accéder au serveur mysql s'executant 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, MySqlWorckbench, ...). Nous allons utiliser le client en ligne de commande inclut dans le container.

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

Arretez et supprimez votre container.

Creation d'images et registres

Création d'images avec un `Dockerfile`

Dans le repertoire de votre choix, clonez le dépot git (url : `<url depot message>`) correspondant au microservice `message`. Puis créez le livrable avec la commande `mvn package`. Nous allons "conteneuriser" ce microservice, c'est à dire créer une image permettant de l'executer 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 executer 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>`
- arretez et supprimez votre container.

On pourra améliorer notre dockerfile, en suivant les suggestions faites ici.

Mise en place et utilisation d'un registre

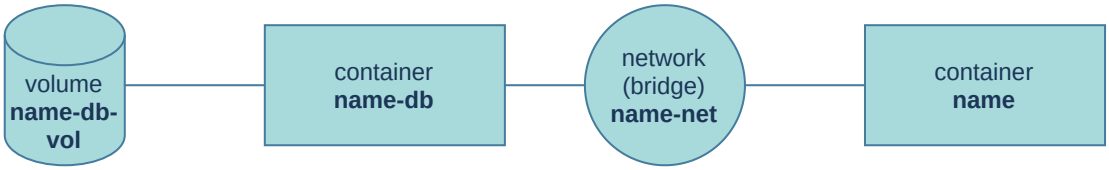
Nous allons maintenant créer un registre pour enregistrer nos images. Ce registre sera en local et non sécurisé. Vous trouverez dans la documentation des indications pour configurer un registre pour une utilisation en production.

- 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]`

Utilisation des volumes et des networks

Nous souhaitons maintenant déployer une architecture (une peu) plus complexe, composée de deux éléments :

- une application java `name`, très proche de `message` conteneurisée précédemment
- une base de données Mysql, qui sera utilisée par `name`



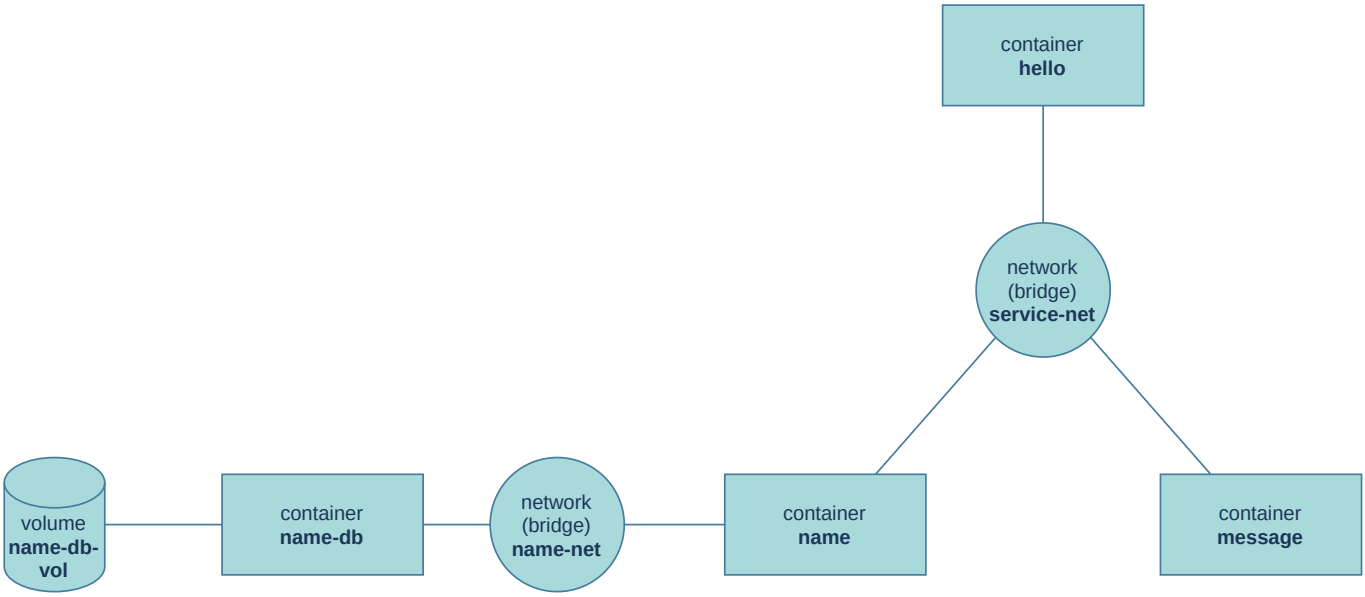
Nous allons donc :

- créer un réseau de type bridge, nommé `name-net`
- créer un volume nommé `name-db-vol`
- lancer un conteneur `name-db` basé sur l'image `mysql`, en lui passant en paramètre :
  - mode détaché (`-d`)
  - volume `name-db-vol` avec comme point de montage dans le container `/var/lib/mysql` (`--mount type=volume,src=name-db-vol,dst=/var/lib/mysql`)
  - réseau `name-net` (`--network name-net`)
  - nom du container (`--name name-db`)
  - des variables d'environnement pour configurer le container :
    - mot de passe de l'utilisateur root (exemple : `-e MYSQL_ROOT_PASSWORD=<root-password>`)
    - base de donnée à créer (exemple : `-e MYSQL_DATABASE=<base_de_donnee>`)
- créer une image pour `name`, exactement comme nous l'avons fait pour `message` (le `Dockerfile` est déjà présent dans le dépôt git)
  - faire un clone du dépôt git (`git clone <url-du-depot>`)
  - changer le répertoire de travail (`cd name`)
  - construire le livrable (`mvn package`)
  - construire l'image (`docker build name .`)
- puis lancez un conteneur `name-ms` basé sur cette image, avec comme parametres :
  - nom du container (exemple : `--name name-ms`)
  - redirection de port 8082 -> 8082
  - réseau `name-net` (`--network name-net`)
  - des variables d'environnement pour configurer le container :
    - url de la base de données (`-e spring.datasource.url=jdbc:mysql://name-db:3306/<base_de_donnee>`)
    - mot de passe de l'utilisateur root dans la base de données (exemple : `-e spring.datasource.password=<root-password>`)

Création de services avec `docker-compose`

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

- créer des images pour chacun des microservices (enfin pur `hello`, c'est déjà fait pour `name` et `message`)
- 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 le microservice `hello` les étapes faites précedemment. Le Dockerfile est déjà présent dans le dépôt git. Vous devez simplement faire : - faire un clone du dépôt git (`git clone <url-du-depot>`) - changer le répertoire de travail (`cd hello`) - construire le livrable (`mvn package`) - construire l'image (`docker build hello .`)

Vérifiez que les trois images `message`, `name` et `hello` sont présentes localement (`docker image ls`). Si ce n'est pas le cas, reproduisez les étapes ci-dessus pour chaque image manquante.

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'un 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 voirs quels services sont executés `docker-compose ps`, puis arreter 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'exterieur que hello.