

Cours Inf3523 – Architecture et Génie des Logiciels

Lab_4 : Introduction à docker

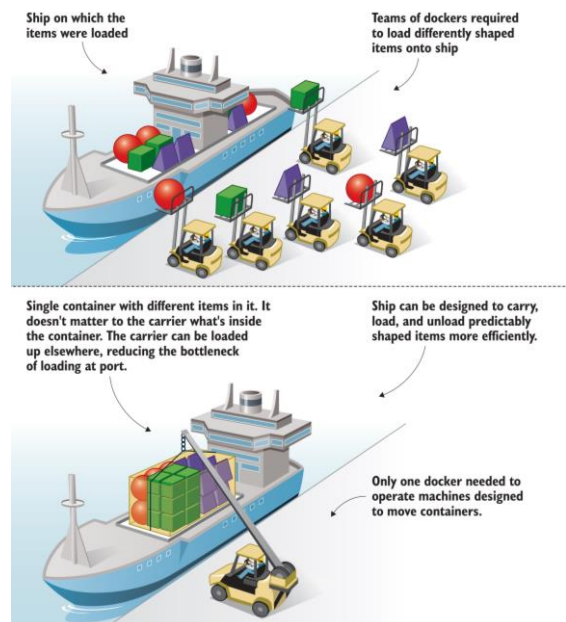
*Ce lab a pour but de vous initier à travailler avec docker. Il va présenter brièvement docker pour ensuite présenter un exercice qui va permettre de déployer notre application **carbackend** en utilisant docker.*

INTRODUCTION

Docker est une plateforme qui vous permet de « construire, déployer et exécuter n'importe quelle application, n'importe où ». Il est désormais considéré comme une méthode standard pour résoudre l'un des aspects les plus coûteux du développement logiciel : le déploiement. Avant l'arrivée de Docker, le processus de développement impliquait généralement des combinaisons de diverses technologies pour gérer le déplacement du logiciel, telles que les machines virtuelles, les outils de gestion de la configuration, les systèmes de gestion de packages, etc. Tous ces outils devaient être gérés et entretenus par des ingénieurs spécialisés, et la plupart avaient leurs propres méthodes de configuration uniques.

QU'EST CE QUE DOCKER

Pour comprendre ce qu'est Docker, il est plus facile de commencer par une métaphore plutôt qu'une explication technique, et la métaphore de Docker est puissante. Un docker était un travailleur qui déplaçait des marchandises commerciales dans et hors des navires lorsqu'ils accostaient dans les ports. Il y avait des boîtes et des objets de différentes tailles et formes, et les dockers expérimentés étaient appréciés pour leur capacité à adapter les marchandises dans les navires à la main de manière rentable (voir figure). Engager des personnes pour déplacer des marchandises n'était pas bon marché, mais il n'y avait pas d'alternative.



Cela devrait sembler familier à quiconque travaille dans le domaine du logiciel. Beaucoup de temps et d'énergie intellectuelle sont consacrés à faire rentrer métaphoriquement des logiciels de formes étranges dans des navires métaphoriques de tailles différentes, pleins d'autres logiciels de formes étranges, afin qu'ils puissent être vendus à des utilisateurs ou à des entreprises ailleurs.

AVANTAGES DE DOCKER

REMPLACEMENT DES MACHINES VIRTUELLES (VM)

Docker peut être utilisé pour remplacer les machines virtuelles (VM) dans de nombreuses situations. Si vous ne vous souciez que de l'application et non du système d'exploitation, Docker peut remplacer la VM, et vous pouvez laisser à quelqu'un d'autre le soin de s'occuper du système d'exploitation. Non seulement Docker est plus rapide à démarrer qu'une VM, il est plus léger à déplacer, et grâce à son système de fichiers en couches, vous pouvez partager plus facilement et plus rapidement les modifications avec d'autres. Il est également solidement ancré dans la ligne de commande et est facilement scriptable.

PROTOYPAGE DE LOGICIEL

Si vous souhaitez expérimenter rapidement avec un logiciel sans perturber votre configuration existante, Docker peut vous fournir un environnement assez rapidement.

EMPAQUETAGE DE LOGICIEL

Parce qu'une image Docker n'a pratiquement aucune dépendance pour un utilisateur Linux, c'est un excellent moyen d'emballer un logiciel. Vous pouvez créer votre image et être sûr qu'elle peut s'exécuter sur n'importe quelle machine Linux moderne, pensez à Java sans avoir besoin d'une JVM.

FACILITATION DE L'ARCHITECTURE DE MICROSERVICES

Docker facilite la décomposition d'un système complexe en une série de parties composables, ce qui vous permet de raisonner sur vos services de manière plus discrète.

MODÉLISATION DES RÉSEAUX

Étant donné que vous pouvez lancer des centaines (voire des milliers) de conteneurs isolés sur une seule machine, la modélisation d'un réseau est assez facile. Cela peut être idéal pour tester des scénarios du monde réel sans exploser votre budget.

ACTIVATION DE LA PRODUCTIVITÉ FULL-STACK HORS LIGNE

Étant donné que vous pouvez regrouper toutes les parties de votre système dans des conteneurs Docker, vous pouvez les orchestrer pour fonctionner sur votre ordinateur portable et travailler en déplacement, même hors ligne.

RÉDUCTION DES SURCOÛTS DE DÉBOGAGE

Les négociations complexes entre différentes équipes concernant le logiciel livré sont monnaie courante dans l'industrie. Docker vous permet de définir clairement (même sous forme de script) les étapes pour déboguer un problème sur un système aux propriétés connues, ce qui simplifie grandement la reproduction des bogues et de l'environnement, et les sépare généralement de l'environnement hôte fourni.

DOCUMENTATION DES DÉPENDANCES LOGICIELLES ET DES POINTS DE CONTACT

En construisant vos images de manière structurée, prêtes à être déplacées vers différents environnements, Docker vous oblige à documenter explicitement les dépendances de votre

logiciel à partir d'un point de départ de base. Même si vous décidez de ne pas utiliser Docker partout, cette documentation peut vous aider à installer votre logiciel ailleurs.

ACTIVATION DE LA LIVRAISON CONTINUE

La livraison continue (CD) est un paradigme de livraison de logiciels basé sur un pipeline qui reconstruit le système à chaque changement, puis le déploie en production (ou "en direct") via un processus automatisé (ou partiellement automatisé).

IMAGE ET CONTENEUR

Une façon de considérer les images et les conteneurs est de les voir comme analogues à des programmes et des processus. De la même manière qu'un processus peut être considéré comme une "application en cours d'exécution", un conteneur Docker peut être vu comme une image Docker en cours d'exécution.

Si vous êtes familier avec les principes de la POO, une autre façon de considérer les images et les conteneurs est de voir les images comme des classes et les conteneurs comme des objets. De la même manière que les objets sont des instances concrètes de classes, les conteneurs sont des instances d'images. Vous pouvez créer plusieurs conteneurs à partir d'une seule image, et ils sont tous isolés les uns des autres de la même manière que les objets le sont. Tout ce que vous modifiez dans l'objet n'affectera pas la définition de la classe, ce sont fondamentalement des entités différentes.

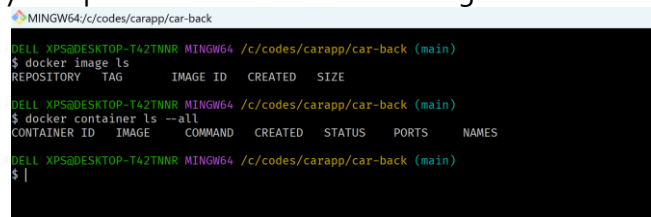
TRAVAUX PRATIQUES

Exercice 1 :

Dans ce TP, nous allons utiliser le backend du cours Inf3522 – Développement d'Applications N-tiers.

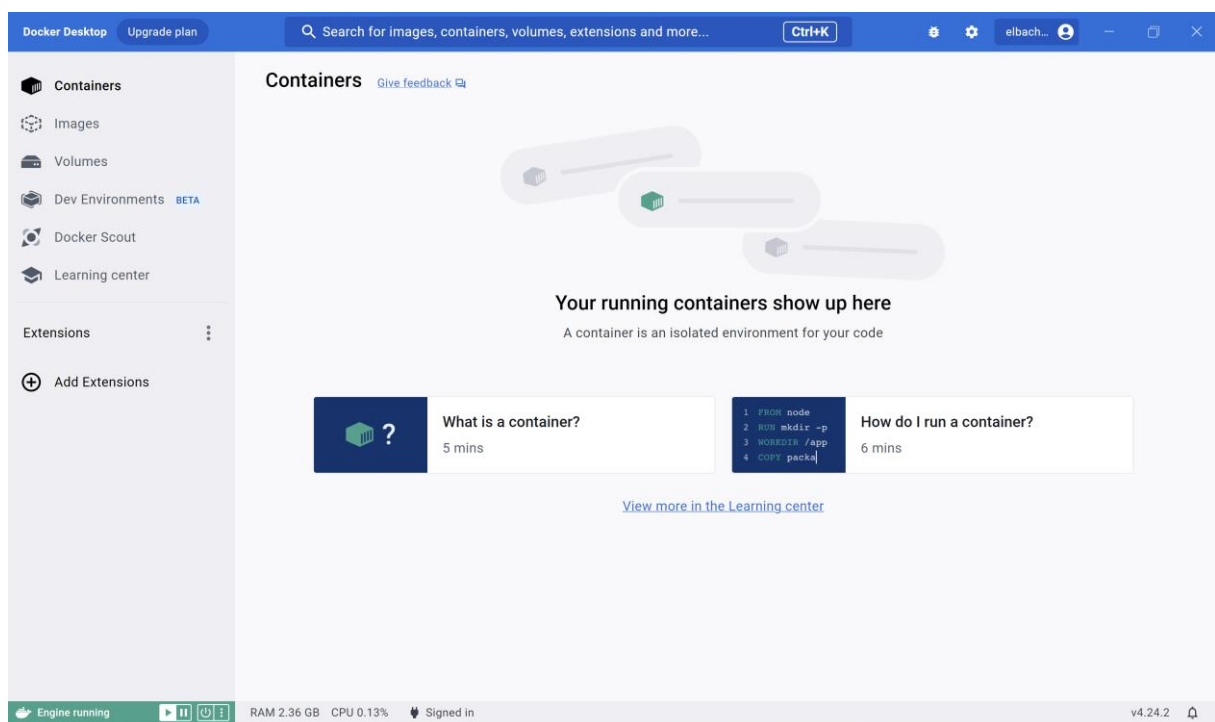
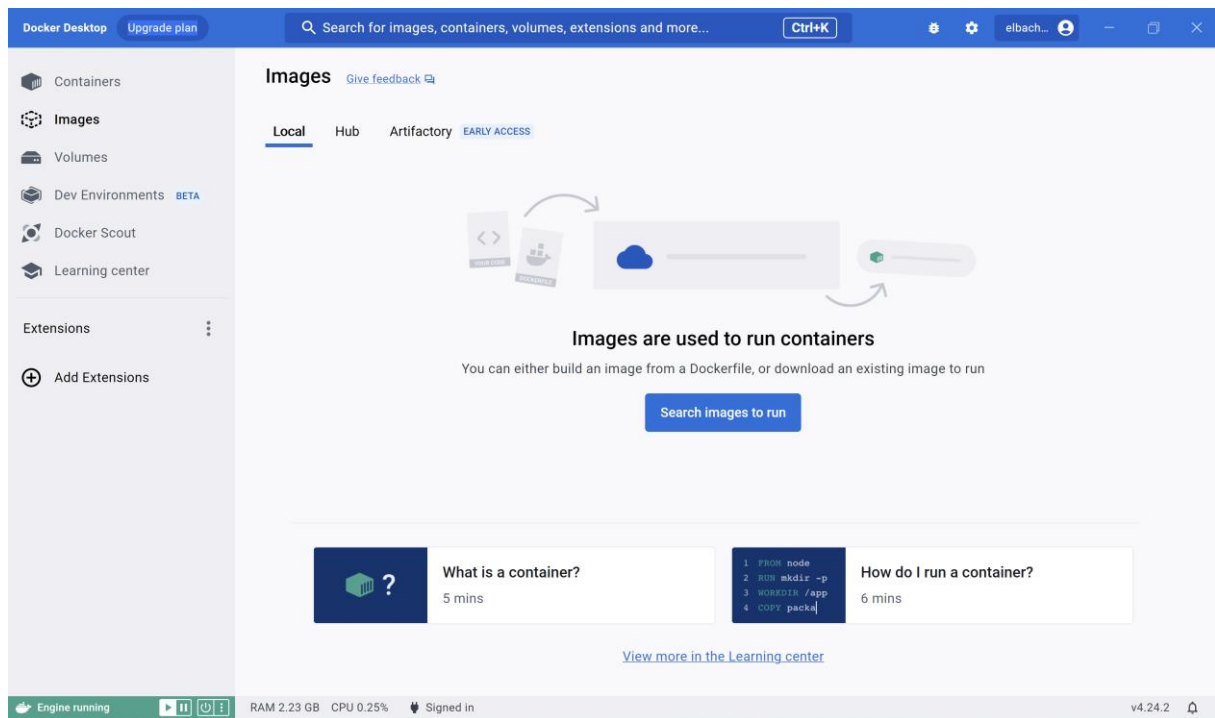
Installer docker et docker desktop

Dans docker nous voyons que nous n'avons aucune image et aucun conteneur.



```
MINGW64/c/codes/carapp/car-back
DELL XPS8DESKTOP-T42TNR MINGW64 /c/codes/carapp/car-back (main)
$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
DELL XPS8DESKTOP-T42TNR MINGW64 /c/codes/carapp/car-back (main)
$ docker container ls --all
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
DELL XPS8DESKTOP-T42TNR MINGW64 /c/codes/carapp/car-back (main)
$ |
```

La même information peut être obtenue en regardant docker desktop



Editer **pom.xml** en ajoutant les lignes 84-86 puis la ligne 89.

```

79
80     <build>
81         <plugins>
82             <plugin>
83                 <groupId>org.springframework.boot</groupId>
84                 <artifactId>spring-boot-maven-plugin</artifactId>
85                 <configuration>
86                     <executable>true</executable>
87                 </configuration>
88             </plugin>
89         </plugins>
90         <finalName>my-car-app</finalName>
91     </build>
92 </project>
93

```

Exportez une image de mariadb en effectuant la commande ci-dessous :

docker pull mariadb:latest

Vérifiez que l'image exportée est bien présente

docker image ls

Exécutez l'image mariadb pour créer le conteneur cardb avec les infos ci-après :

docker run --name cardb -p 3306:3306 -e MARIADB_ROOT_PASSWORD=root -e MARIADB_DATABASE=cardb -d mariadb

Vérifiez que le conteneur cardb créé à partir de l'image mariadb est bien présent

docker ps

Editer application.properties comme suit :

```
spring.datasource.url=jdbc:mariadb://${MARIADB_HOST:localhost}:${MARIADB_PORT:3306}/cardb
spring.datasource.username=${MARIADB_USER:root}
spring.datasource.password=${MARIADB_PASSWORD:root}
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
spring.data.rest.basePath=/api
```

Effectuez les commandes pour créer le fichier **my-car-app.jar** qui servira à créer l'image comme spécifié dans le fichier **Dockerfile** ci-dessous.

mvn clean

mvn clean install

Créez un fichier nommé **Dockerfile** dans la racine de votre application et mettez-y le contenu ci-dessous :

```
FROM openjdk:21
VOLUME /tmp
EXPOSE 8080
ARG JAR_FILE
COPY target/my-car-app.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Créez l'image docker de notre backend nommé carbackend comme suit :

docker build -t carbackend .

Vérifiez que l'image exportée est bien présente

docker image ls

Afin de lier le conteneur de notre BD et celui du backend, nous avons besoin de créer un réseau que nous nommons **carbackend-network** comme suit :

docker network create carbackend-network

Ensuite nous connectons notre conteneur cardb au réseau car-network que nous venons de créer comme suit :

docker network connect carbackend-network cardb

La commande ci-dessous permet d'inspecter le réseau et de s'assurer que cardb est bien connecté à ce réseau (regardez le hash dans le champ containers et faire docker ps pour comparer) :

docker network inspect carbackend-network

Enfin nous créons le conteneur nommé carapp de notre backend en utilisant le conteneur cardb ainsi que le réseau carbackend-network comme suit :

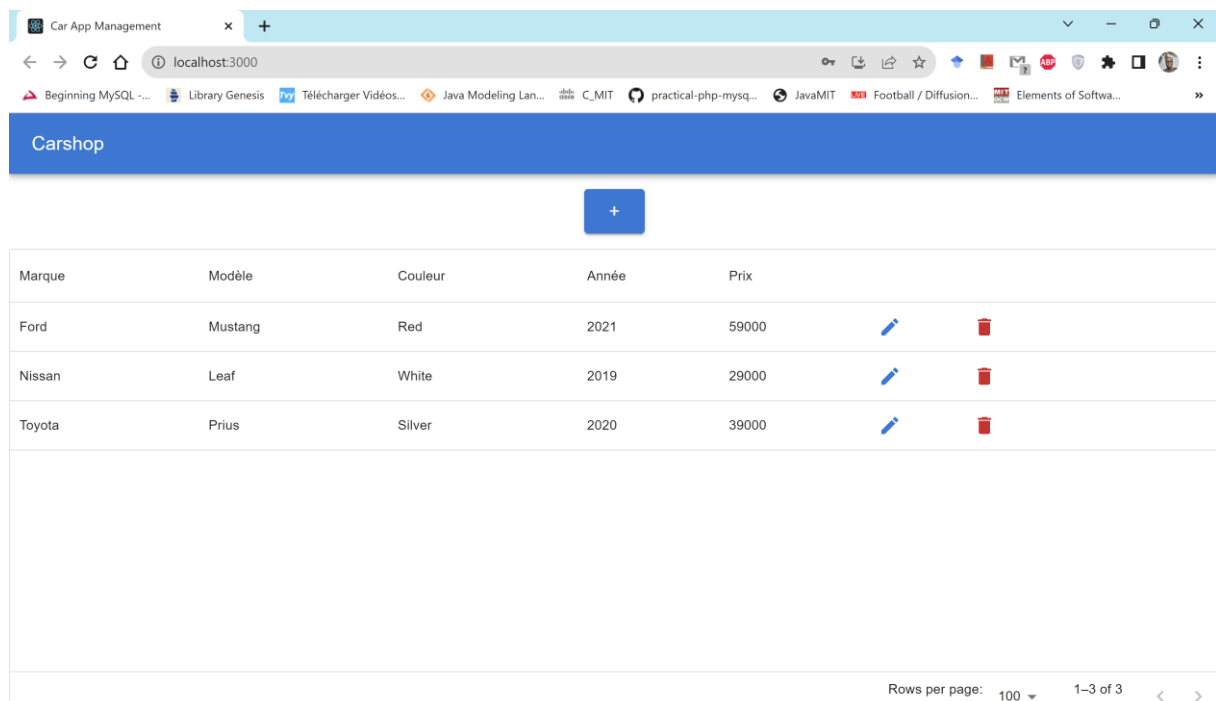
docker run -p 8080:8080 --name carapp --net car-network -e MARIADB_HOST=cardb -e MARIADB_USER=root -e MARIADB_PASSWORD=root -e MYSQL_PORT=3306 carbackend

Si nous réaffichons les conteneurs en cours d'exécution, nous verrons les deux conteneurs créés ci-haut :

docker ps

Nous pouvons également regarder les images créées ainsi que les conteneurs dans docker desktop.

Il reste à lancer notre frontend en utilisant la commande **npm start** :



Pour arrêter les conteneurs nous faisons les commandes :

docker container stop carapp

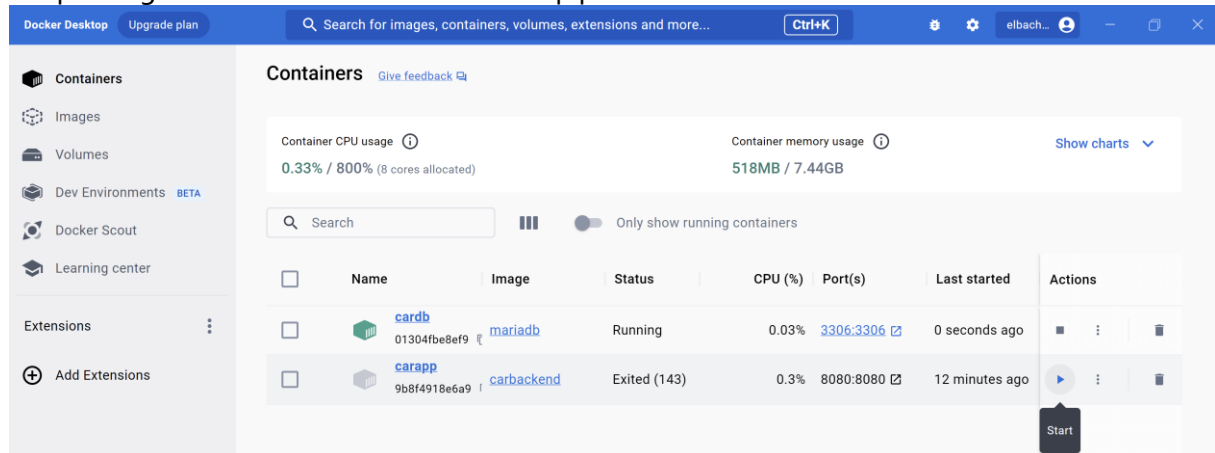
docker container stop cardb

Pour les démarrer, nous utilisons les commandes :

docker container start cardb

docker container start carapp

On peut également utiliser docker desktop pour lancer ou arrêter les conteneurs :



Exercice 2 :

Les images que je vous ai remises sont respectivement une image de la BD mariadb et une image de notre backend.

De ce fait, vous n'avez pas besoin d'avoir ni un serveur mysql ni spring boot voire java pour exécuter la partie backend de notre application. Il vous suffit d'avoir docker afin de pouvoir exécuter les images et ainsi créer des conteneurs qu'il vous suffit de lancer, puis d'autre part d'allumer votre frontend.

Assurez-vous d'éteindre votre serveur backend, même la BD si possible la mettre hors-ligne afin de lancer l'application avec juste votre frontend et les deux images docker (sans connexion internet également).

Copiez les images dans un dossier de votre choix, puis effectuez la commande :

docker load -i <path to image tar file>

où <path to image file> correspond au chemin d'accès de votre image. Par exemple si je veux charger l'image **car_db_build_01.tar**, se trouvant dans le disque local C, dans le sous-dossier **codes/dockers**, je vais faire la commande ci-dessous pour récupérer l'image de notre bd (il vous suffit de faire de même pour avoir l'image du backend).

docker load -i C:/codes/dockers/car_db_build_01.tar

Créez les conteneurs comme dans l'exercice 1 puis lancez votre application frontend.

NB : pour exporter une image (que vous pourrez partager) faire la commande :

docker save -o <generated tar file name> <repository_name:tag_name>

Par exemple si je veux exporter une image de mariadb avec comme nom **car_db_build_01.tar**, je fais la commande ci-dessous :

docker save -o car_db_build_01.tar mariadb:latest