Utilisation de Docker

Docker est un outil de création de conteneurs qui permet facilement de créer des environnements d'exécution qui soient plus simples et moins coûteux que des machines virtuelles.

1 Installation

Commençons par installer **Docker** :

```
# ajout du dépôt docker
dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

# installation de docker
dnf -y install docker-ce docker-ce-cli containerd.io docker-compose-plugin

# démarrer Docker et l'activer
systemctl enable --now docker

# Utilitaire pour tester et attendre la disponibilité
wget https://raw.githubusercontent.com/vishnubob/wait-for-it/refs/heads/master/wait-for-it.sh
chmod a+rx wait-for-it.sh
mv -v wait-for-it.sh /usr/bin/wait-for-it
```

2 Les images

Dans **docker**, les images sont des systèmes minimums prêts à être exploités pour des exécutions ou des déploiements de logiciels. Quelques manipulations (commandes `docker image`) :

```
## charger les images
docker pull almalinux
docker pull ubuntu

## lister les images
docker images
```

Travail à faire : En utilisant la documentation, supprimez une image (`docker image rm`) et inspectez une autre image (`docker image inspect`. Finalement, avec `prune`, supprimez les images inutiles.

3 Les conteneurs

Nous allons maintenant, à partir des images, créer des conteneurs (commandes `docker container`) :

Quelques essais de création

```
## Une exécution
docker run almalinux /bin/echo Hello

## Vérifier que le conteneur précédent est arrêté
docker ps -a

## Vérifier AlmaLinux
docker run almalinux /usr/bin/cat /etc/redhat-release |
    fgrep -i almaLinux

## Vérifier Ubuntu
docker run ubuntu /usr/bin/apt list |
    fgrep -i bash | fgrep -i ubuntu

## Toujours le même noyau
uname -r
docker run almalinux /usr/bin/uname -r
docker run ubuntu  /usr/bin/uname -r
docker run archlinux /usr/bin/uname -r

## Lister les conteneurs créés
docker ps -a

## Stopper et supprimer les conteneurs créés
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

Conteneurs en mode interactif

```
docker run -it almalinux /bin/bash
[root@92afaafe4463 /]# uname -a
Linux a7134dda282d 5.14.0-503.33.1.el9_5.x86_64 ...
[root@92afaafe4463 /]# cat /etc/redhat-release
AlmaLinux release 9.5 (Teal Serval)
[root@92afaafe4463 /]# exit

## Visualiser les conteneurs (-a pour voir les stoppés)
docker ps -a
```

Conteneurs en arrière-plan

```
## Conteneurs en arrière-plan
docker run -itd almalinux /bin/bash

## Le visualiser
docker ps -l                # -l last

## Récupérer son ID
ID=$(docker ps -l -q)       # -q id
echo "ID␣is␣$ID"

## Modifier le conteneur en arrière-plan
docker exec $ID /bin/bash -c "echo␣hello␣>␣/tmp/hello.txt"
docker exec $ID cat /tmp/hello.txt | fgrep hello

docker stop $ID
docker rm $ID
```

4 Faire des images

Continuons en créant et sauvegardant des images :

```
## Créer un conteneur
docker run -idt almalinux /bin/bash
ID=$(docker ps -a -q -l)
echo "ID␣is␣$ID"

## Mettre a jour et ajouter httpd
docker exec $ID dnf -y update
docker exec $ID dnf -y install httpd

## Ou se trouve httpd ?
docker exec $ID /usr/bin/whereis httpd

## Créer une image
docker commit $ID my-httpd
docker images

## Créer un conteneur 'my-httpd' avec
## httpd activé et redirection des ports
docker run --name my-httpd -dt -p 8800:80 \
   my-httpd /usr/sbin/httpd -D FOREGROUND

## Changer la page par défaut
docker exec my-httpd /bin/bash \
   -c 'echo␣httpd␣on␣docker␣>␣/var/www/html/index.html'

## Vérifier
curl localhost:8800 | fgrep 'httpd␣on␣docker'

## Récupérer des informations
docker inspect my-httpd

## Stop
docker stop my-httpd
docker rm my-httpd
```

5 Utiliser des Dockerfile

Nous allons maintenant créer des images de manière automatisée en utilisant des `Dockerfile`.

5.1 Une image pour java

```
rm -fr java-docker

## Créer un répertoire
mkdir java-docker

## Créer le dockerfile
cat > java-docker/Dockerfile <<FIN

FROM almalinux
LABEL desc="My␣Java␣17"
RUN dnf -y update
RUN dnf -y install java-17-openjdk
RUN dnf -y clean all

FIN

## Construire une image
docker build -t my-java:latest java-docker
docker images

## Tester
docker run my-java:latest java -version
```

5.2 Une image pour une application Spring-Boot

Nous pouvons maintenant créer une image pour déployer notre application Spring-Boot :

```
rm -fr myapp-docker

## Créer un répertoire
mkdir myapp-docker

## Préparer le WAR dans le répertoire
wget https://jean-luc-massat.pedaweb.univ-amu.fr/ens/cca/spring-app.war
mv -v spring-app.war myapp-docker

## Créer le dockerfile
cat > myapp-docker/Dockerfile <<FIN

FROM my-java:latest
LABEL desc="My␣SpringBoot␣App"
WORKDIR /app
COPY spring-app.war /app/spring-app.war
EXPOSE 8081
ENTRYPOINT ["java", "-jar", "/app/spring-app.war"]

FIN

## Construire une image
docker build -t my-app:latest myapp-docker
docker images

## Tester en mode interactif Control-C pour stopper
docker run -it -p 9000:8081 my-app

## Tester en mode démon
docker run --name my-app -dt -p 9000:8081 my-app

## Récupérer son adresse IP
ipaddr=$(docker inspect \
  --format='{{range␣.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' my-app)
echo "IPAddr␣is␣$ipaddr"

## Pour attendre le démarrage
wait-for-it -t 0 $ipaddr:8081

## Tester my-app
curl -L http://localhost:9000 | fgrep -i 'star␣war'

## Vérifier les logs
docker container logs my-app

## Consulter les statistiques
docker container stats --no-stream
```

Note : Vous pouvez observer (avec `ip addr`) que docker a ajouté des interfaces réseau virtuelles (ainsi que des routes `ip route`) afin de pouvoir dialoguer avec les conteneurs créés.

6 Déployer avec une BD MySQL

Commençons par supprimer tous nos conteneurs :

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

Continuons en chargeant une image MySQL :

```
docker pull docker.io/library/mysql
docker images
```

Créons un réseau qui va permettre de relier les conteneurs MySql et Spring-Boot :

```
docker network create my-database
```

Créons et configurons le conteneur qui héberge notre BD :

```
docker run --name my-mysql --network my-database \
    -e MYSQL_ROOT_PASSWORD=root \
    -e MYSQL_USER=moviesuser \
    -e MYSQL_PASSWORD=moviespass \
    -e MYSQL_DATABASE=moviesdb -d mysql
```

Nous devons temporiser afin de nous assurer que la BD est opérationnelle :

```
## Récupérer adresse IP
ipaddr=$(docker inspect \
  --format='{{range␣.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' my-mysql)
echo "IPAddr␣of␣my-mysql␣is␣$ipaddr"

## Pour attendre le démarrage
wait-for-it -t 0 $ipaddr:3306
```

Créons maintenant l'image de notre application :

Construction de l'image

```
## Créer un répertoire
mkdir myappmysql-docker

## Préparer le WAR dans le répertoire
wget https://jean-luc-massat.pedaweb.univ-amu.fr/ens/cca/spring-app.war
mv -v spring-app.war myappmysql-docker

## Créer le script de démarrage
cat > myappmysql-docker/start.sh <<FIN
java \
    -Dspring.datasource.driver-class-name=com.mysql.jdbc.Driver \
    -Dspring.datasource.url=jdbc:mysql://my-mysql:3306/moviesdb \
    -Dspring.jpa.generate-ddl=true \
    -Dspring.jpa.hibernate.ddl-auto=update \
    -Dspring.datasource.username=moviesuser \
    -Dspring.datasource.password=moviespass \
    -jar spring-app.war
FIN

## Créer le dockerfile
cat > myappmysql-docker/Dockerfile <<FIN
FROM my-java:latest
LABEL desc="My␣SpringBoot␣App␣with␣MySql"
WORKDIR /app
COPY spring-app.war /app/spring-app.war
COPY start.sh /app/start.sh
EXPOSE 8081
ENTRYPOINT ["/bin/bash", "/app/start.sh"]
FIN

## Construire une image
docker build -t my-app-mysql:latest myappmysql-docker
docker images
```

⏩ Travail à faire : En vous aidant de la documentation (`docker image`), vous pouvez sauver dans une fichier `.tar` votre image (option `save` et le recharger (option `load`). Vous avez donc un moyen simple pour packager et distribuer vos images.

Création du conteneur

```
## Tester en mode interactif Control-C pour stopper
docker run -it -p 9100:8081 --network my-database my-app-mysql:latest

## Tester en mode démon
docker run --name my-app-mysql -dt -p 9100:8081 \
   --network my-database my-app-mysql:latest

## Récupérer son adresse IP
ipaddr=$(docker inspect \
  --format='{{range␣.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' \
  my-app-mysql)
echo "IPAddr␣of␣my-app-mysql␣is␣$ipaddr"

## Pour attendre le démarrage
wait-for-it -t 0 $ipaddr:8081

## Tester my-app-mysql
curl -L http://localhost:9100 | fgrep -i 'star␣war'
```

⏩ Travail à faire : Vous pouvez maintenant vérifier que les tables du conteneur MySql sont bien à jour. Pour ce faire, nous allons utiliser un conteneur MySql client :

```
## Lister les tables
docker run -i --network my-database --rm mysql mysql \
   -hmy-mysql -umoviesuser -pmoviespass moviesdb \
   <<< 'SELECT␣*␣FROM␣movie'
```

⏩ Dernière vérification :

- Modifiez les données dans votre application (en passant par un tunnel ssh).
- Stoppez le conteneur Spring-Boot.
- Relancez-le et vérifiez que les données sont bien les mêmes.