Utilisation de Docker (2/2)

Pré-requis. Ce sujet nécessite le sujet :
- Docker

1 Des images paramétrées

Nous avons vu, dans la séances précédentes, que l'image **MySQL** est paramétrée (`MYSQL_DATABASE` par exemple). Nous allons faire de même pour notre application Spring-Boot avec la directive `ENV` :

Construire une image paramétrée

```
## Créer un répertoire
mkdir myapp-env

## Préparer le WAR dans le répertoire
wget http://tinyurl.com/jlmassat2/cca/spring-app.war
mv -v spring-app.war myapp-env

## Créer le script de démarrage
cat > myapp-env/start.sh <<FIN
echo "Log␣\$MYAPP_NAME␣in␣\$MYAPP_LOG"
java -jar spring-app.war &> \$MYAPP_LOG
FIN

## Créer le dockerfile
cat > myapp-env/Dockerfile <<FIN
FROM my-java:latest
LABEL desc="My␣SpringBoot␣App␣with␣Env"
WORKDIR /app
COPY spring-app.war start.sh /app/
ENV MYAPP_LOG="/var/tmp/myapp.log"
ENV MYAPP_NAME="myapp"
EXPOSE 8081
ENTRYPOINT ["/bin/bash", "/app/start.sh"]
FIN

## Construire une image
docker build -t myapp-env:latest myapp-env
docker images
```

Lancer une image paramétrée

```
## Tester en mode démon
docker run --name myapp-env -dt -p 9100:8081 \
    -e MYAPP_NAME=moviesapp \
    -e MYAPP_LOG=/tmp/moviesapp.log \
    myapp-env:latest

## Vérifier le nouveau fichier de trace
sleep 10s
docker exec myapp-env cat /tmp/moviesapp.log | head

## Stopper
docker stop myapp-env
docker rm myapp-env
```

2 Surveiller les conteneurs

Dans les exercices précédents nous avons surveillé le bon lancement d'un conteneur avec le script `wait-for-it`. C'est un bricolage (bien pratique) mais il est préférable que le conteneur surveille et déclare lui-meme sont état de santé. Commençons par ajouter une clause `HEALTHCHECK` au dockerfile et reconstruisons l'image.

Construction d'une image auto-surveillée

```
## Ajouter au dockerfile
cat >> myapp-env/Dockerfile <<FIN
HEALTHCHECK --interval=4s --timeout=3s --retries=100 \
  CMD curl -f http://localhost:8081/movies/ || exit 1
FIN

## Reconstruire une image
docker image rm myapp-env:latest
docker build -t myapp-env:latest myapp-env
docker images
```

Lancement d'une image auto-surveillée

```
## Tester en mode démon
docker run --name myapp-env -dt -p 9100:8081 \
    -e MYAPP_NAME=moviesapp \
    -e MYAPP_LOG=/tmp/moviesapp.log \
    myapp-env:latest

## Attendre que le conteneur soit healthy - a la main
until [ "$status" = healthy ]; do
    echo waiting; sleep 2s
    status="$(docker␣inspect␣--format='{{.State.Health.Status}}'␣myapp-env)"
done

## Un outil pour attendre
wget https://github.com/jordyv/wait-for-healthy-container/raw/refs/heads/master/wait-for-healthy-container.sh
mv -v wait-for-healthy-container.sh /usr/bin/wait-for-healthy-container
chmod a+x /usr/bin/wait-for-healthy-container

## Attendre que le conteneur soit healthy - plus simple
wait-for-healthy-container myapp-env

## Stopper
docker stop myapp-env
docker rm myapp-env
```

⏩ **Travail à faire :** Pour bien suivre le déroulement, vous pouvez aussi utiliser `docker stats`.

3 Les volumes

Nous avons stocké les traces, mais quand le conteneur s'arrête, les données sont perdues. Nous allons prévoir un volume de stockage (`docker volume`) afin d'externaliser les traces et ainsi les conserver.

```
## Créer un volume partagé
docker volume create my-logs
ls -l /var/lib/docker/volumes/my-logs

## Lister les volumes
docker volume ls

## Tester en mode démon -- avec le volume
docker run --name myapp-env -dt -p 9100:8081 \
    -v my-logs:/mnt \
    -e MYAPP_NAME=moviesapp \
    -e MYAPP_LOG=/mnt/moviesapp.log \
    myapp-env:latest

## Attendre que le conteneur soit healthy
wait-for-healthy-container myapp-env

## Verifier localement les logs
tail -5 < /var/lib/docker/volumes/my-logs/_data/moviesapp.log

## La même chose en docker
docker run --name my-cat -v my-logs:/mnt alpine cat /mnt/moviesapp.log |
   tail -5

## Stopper
docker stop myapp-env my-cat
docker rm  myapp-env my-cat

## Supprimer le volume
docker volume rm my-logs
```

4 Démarrage automatique

Il est absolument nécessaire de pouvoir faire démarrer automatiquement un conteneur (pour que le service soit opérationnel). Nous pourrions un service (au sens de `systemd`, mais la solution ci-dessous est plus simple :

```
## Redémarrage automatique (si le conteneur est déjà lancé)
docker update --restart always myapp-env

## ou bien lancement avec redémarrage automatique
docker run -d --restart always myapp-env
```

⏩ **Travail à faire :** Après redémarrage de votre VM, vérifiez que le conteneur est lancé.

5 Limiter les ressources

Dans docker, nous pouvons restreindre les ressources allouées à un conteneur (plus d'informations). Dans l'exemple ci-dessous, nous limitons la CPU (option `--cpus`) :

```
## Boucle consommatrice de CPU - à régler pour 5 secondes
LOOP='for((x=0;x<5000000;x++));␣do␣true;␣done'

## Boucle dans la machine invitée
time bash -c "$LOOP"

## Boucle dans docker
time docker run -it almalinux bash -c "$LOOP"

## Boucle avec 1/2 CPU
time docker run -it --cpus="0.5" almalinux bash -c "$LOOP"
```

⏩ **Travail à faire :** Exécutez à nouveau ces exemples avec, dans un autre terminal, la commande `htop` afin de surveiller la consommation de CPU.

⏩ **Travail à faire :** Vous pouvez arrêter votre VM et lui allouer deux CPUs (paramètres de VirtualBox). Après redémarrage, essayez ces deux exemples qui lancent des boucles en parallèle :

```
## Boucle consommatrice de CPU - à régler pour 5 secondes
LOOP='for((x=0;x<5000000;x++));␣do␣true;␣done'

## Deux boucles en //
DLOOP="{␣$LOOP␣&}␣;␣{␣$LOOP␣&};␣wait;␣wait;␣echo␣fini"

## Deux boucles avec 1 CPU
time docker run -it --cpus="1.0" almalinux bash -c "$DLOOP"

## Deux boucles avec 1,5 CPU
time docker run -it --cpus="1.5" almalinux bash -c "$DLOOP"
```