Exécuter des applications graphiques avec Docker

Table of Contents

Pour qu'une application graphique fonctionne, il faut utiliser un serveur XServer disponible dans tous les environnements de bureau Linux. Toutefois, par défaut, un conteneur Docker ne peut pas exécuter une application graphique.

Il existe différentes options pour exécuter des applications graphiques dans un conteneur Docker, telles que l'utilisation de SSH avec transfert X11 ou de VNC, mais la plus simple consiste à partager un socket X11 avec le conteneur et à l'utiliser directement.

Partager le serveur X de l'hôte avec le conteneur

- Créer un volume `--volume="$HOME/.Xauthority:/root/.Xauthority:rw"`
- partager la variable d'environnement `DISPLAY` de l'hôte avec le conteneur `--env="DISPLAY"`
- Exécuter le conteneur avec le pilote de réseau hôte avec `--net=host`

Par exemple avec un conteneur **Android Studio** en utilisant le fichier Docker suivant comme point de départ:

```
$ cat Dockerfile

FROM ubuntu:14.04
MAINTAINER Admatic Engineering
Team@ADMATIC.INENV DEBIAN_FRONTEND=noninteractiveRUN apt-get update# Default
jdk
RUN apt-get install -y default-jdk
#Dépendances 32 bits d'androïde et d'utils
Lancez apt-get install -y \
    bison \
    git \
    gperf \
    lib32gcc1 \
    lib32bz2-1.0 \
    lib32ncurses5 \
    lib32stdc ++ 6 \
    lib32z1 \
    libc6-i386 \
    libxml2-utils \
    faire \
    Zip
```

```
# Télécharger et décompresser Android Studio pour Linux
ADD
http://dl.google.com/dl/android/studio/ide-zips/1.2.1.1/android-studio-ide-1
41.1903250-linux.zip /opt/android-studio.zip
RUN cd /opt/ && unzip android-studio.zip && rm android-studio.zip
CMD /opt/android-studio/bin/studio.sh
```

construire l'image du conteneur

```
$ sudo docker build -t android-studio.
```

lancer **Android Studio** à l'intérieur du conteneur.

```
$ sudo docker run --net=host --env="DISPLAY" --
volume="$HOME/.Xauthority:/root/.Xauthority:rw" android-studio
```

Si tout se passe bien, on doit voir **Android Studio** s'exécuter depuis un conteneur Docker.

Partager le socket X11 avec le conteneur

L'idée est assez simple et on peut facilement la mettre oeuvre, il suffit de monter dans un volume du système de fichiers le socket X11 de l'hôte (/tmp/.X11-unix) sur le même chemin d'accès au conteneur et de définir la variable d'environnement DISPLAY du conteneur comme affichage du serveur X de L'hôte, donc lorsque l'application du conteneur envoie les instructions de rendu, il les envoie au serveur X hôte.

Pour pouvoir lancer une application X depuis un conteneur Docker, il faut réaliser les opérations suivantes :

- Donner à Docker les droits d’accès au serveur X (sur le système hôte, bien entendu) : `xhost +local:docker`
- Lancer le conteneur avec (au moins) les options `-e DISPLAY=$DISPLAY et -v /tmp/.X11-unix/:/tmp/.X11-unix`

Par exemple avec un conteneur **Firefox** en utilisant le fichier Docker suivant comme point de départ:

```
From Ubuntu: 14.04
RUN apt-get update && apt-get install -y firefox

# Replace 1000 with your user / group id
RUN export uid=1000 gid=1000 && \
    mkdir -p /home/developer && \
    echo "developer:x:${uid}:${gid}:Developer,,,:/home/developer:/bin/bash"
>> /etc/passwd && \
    echo "developer:x:${uid}:" >> /etc/group && \
    echo "developer ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/developer && \
    chmod 0440 /etc/sudoers.d/developer && \
```

```
    chown ${uid}:${gid} -R /home/developer

USER developer
ENV HOME /home/developer
CMD /usr/bin/firefox
```

construire l'image du conteneur

```
$ docker build -t firefox. et exécutez le conteneur avec:
```

lancer **Firefox** à l'intérieur du conteneur.

```
docker run -ti --rm \
       -e DISPLAY=$DISPLAY \
       -v /tmp/.X11-unix:/tmp/.X11-unix \
       firefox
```

Si tout se passe bien, on doit voir **Firefox** s'exécuter depuis un conteneur Docker.

Forward X11 via une connexion SSH à l'hôte du conteneur

- Indiquer à `xhost` d'autoriser le transfert depuis l'ID du conteneur:

```
$ sudo docker run -it -d \
    --net = hôte \
    --env = "AFFICHER" \
    --env = "QT_X11_NO_MITSHM = 1" \
    --volume = "/tmp/.X11-unix:/tmp/.X11-unix:rw" \
    --device = "/dev/video0:/dev/video0" \
    --volume = "/chemin/vers/sharedDockerFiles: /root/sharedDockerFiles" \
    --volume = "/etc/machine-id:/etc/machine-id" \
    yourdockerrepo/image:tag \
    [image-du-conteneur]
```

```
$ export containerId=$(docker ps -l -q)
$ sudo xhost + local: `sudo docker inspect --format = '{{.Config.Hostname}}'
$ containerId`
$ sudo docker start $containerId
```

- Copier `.Xauthority` de l'hôte principal vers le répertoire `sharedDockerFiles`:

```
sudo cp ~/.Xauthority /chemin/vers/sharedDockerFiles
```

- Démarrer le conteneur et vérifier que la variable **$DISPLAY** du conteneur est identique à celle de l'hôte. la sortie `echo $DISPLAY` devrait être identique dans l'hôte et dans le conteneur,si ce n'est pas le cas, redémarrer le conteneur en utilisant `export DISPLAY={sortie de la`

`variable $DISPLAY de l'hôte}`

- copier le fichier .Xauthority dans le dossier partagé du conteneur

```
sudo cp /root/sharedDockerFiles/.Xauthority ~/
```

pour copier .Xauthority dans un conteneur au début de la session ssh avant d'utiliser l'interface graphique utiliser la syntaxe suivante:
`sudo docker exec -i nom_cipient bash -c 'cat> ~/.Xauthority' < ~/.Xauthority`

- Editer le fichier `/etc/ssh/ssh_config` du conteneur sous Host (nécessaire une fois) pour inclure:

```
ForwardX11 yes
X11Forwarding yes
```

- Redémarrer le conteneur et lancer l'application graphique