Docker

TRAVAUX PRATIQUES

Thierry Vaira
LaSalle BTS SN IR

Présentation

Docker est une plate-forme permettant aux développeurs et aux administrateurs système de créer, d'exécuter et de partager des applications avec des **conteneurs**. L'utilisation de conteneurs pour déployer des applications est appelée **conteneurisation**.

La conteneurisation est de plus en plus populaire car les conteneurs sont :

- **Flexible** : même les applications les plus complexes peuvent être conteneurisées.
- **Léger** : les conteneurs exploitent et partagent le noyau hôte (efficaces en termes de ressources système vs machines virtuelles).
- **Portable** : créer localement, déployer sur le cloud et exécuter n'importe où.
- **Faible Couplage** : les conteneurs sont hautement autonomes et encapsulés (remplacement ou mise à niveau sans perturber les autres).
- **Évolutif** : augmenter et distribuer automatiquement des répliques de conteneurs dans un centre de données.
- **Sécurisé** : les conteneurs appliquent des contraintes et des isolements aux processus sans aucune configuration requise de la part de l'utilisateur.

docker-io vs docker-ce

Les anciennes versions de Docker étaient appelées **docker** ou **docker-engine** ou **docker-io**.

Le paquet **docker-io** est toujours le nom utilisé par **Debian/Ubuntu** pour la version Docker fournie sur les dépôts officiels.

docker-ce est une version certifiée fournie directement par **docker.com**. Docker a une version *Entreprise Edition* (EE) et une version gratuite *Community Edition* (CE).

Remarque : La principale raison d'utiliser le nom docker-io sur la plate-forme Debian/Ubuntu était d'éviter un conflit de nom avec le binaire de la barre d'état système.

Installation de docker.io sous Ubuntu 18.04

```
$ apt-cache policy docker.io
docker.io:
  Installé : (aucun)
  Candidat : 19.03.6-0ubuntu1~18.04.1
...

$ sudo apt install docker.io
```

Installation de docker-ce sous Ubuntu 18.04

Avant d'installer Docker Community Edition (**docker-ce** de docker.com), il faudra peut-être supprimer les anciens paquets :

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

Puis :

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu bionic stable"
$ sudo apt update

$ sudo apt install docker-ce
```

Vérification

```
$ systemctl status docker
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled;)
Active: active (running) since Fri 2020-05-29 09:01:05 UTC; 56s ago
Docs: https://docs.docker.com
Main PID: 19679 (dockerd)
Tasks: 8
CGroup: /system.slice/docker.service
+-19679 /usr/bin/dockerd -H fd://--containerd=/run/containerd/containerd.sock

$ docker --version
Docker version 19.03.10, build 9424aeaee9

$ docker --help
```

no sudo (facultatif)

```
$ docker images
Got permission denied …

$ sudo usermod -aG docker ${USER}

# Redémarrer la session ou :
$ su - ${USER}

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

Docker Engine

Docker Engine est une technologie de conteneurisation open source pour créer et conteneuriser des applications. Docker Engine agit comme une application client-serveur avec :

- Un serveur avec un processus démon **`dockerd`**

```
$ ps ax | grep docker
25447 ? Ssl 21:52 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

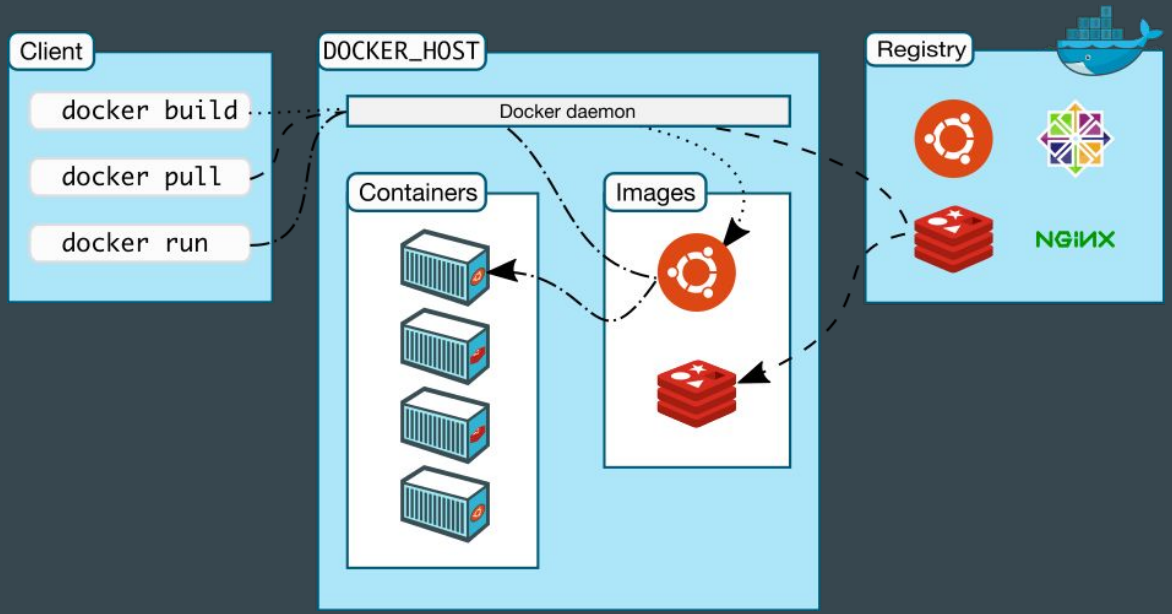
- Un client avec une interface en ligne de commande (CLI) client **`docker`**

```
$ docker --version
Docker version 19.03.10, build 9424aeaee9
$ docker --help
```

- Une API pour les programmes qui s'interfacent avec le démon Docker.

Architecture

Docker utilise une architecture **client/serveur**. Le client Docker discute avec le démon Docker, qui s'occupe de la construction, de l'exécution et de la distribution de vos **conteneurs** Docker. Le client et le démon Docker peuvent s'exécuter sur le même système, ou vous pouvez connecter un client Docker à un démon Docker distant.



Image

Une **image** est un modèle en lecture seule (*template*) avec des instructions pour créer un **conteneur** Docker. Souvent, une image est basée sur une autre image, avec une personnalisation supplémentaire.

Il est possible de créer ses propres images ou utiliser uniquement celles créées par d'autres et publiées dans un registre (*registry*).

Pour créer une image, on utilise un fichier **`Dockerfile`** avec une syntaxe simple pour définir les étapes nécessaires pour créer l'image et l'exécuter.

Chaque instruction d'un **`Dockerfile`** crée un calque dans l'image. Lorsque on modifie le **`Dockerfile`** et on reconstruit l'image, seuls les calques qui ont été modifiés sont reconstruits. Cela fait partie de ce qui rend les images si légères, petites et rapides par rapport aux autres technologies de virtualisation.

Conteneur

Un **conteneur** est une instance exécutable d'une **image**. Un conteneur est défini par son image ainsi que par les **options de configuration** fournis à la création ou au démarrage.

Il est possible de créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l'aide de la commande **`docker`**.

On peut connecter un conteneur à un ou plusieurs réseaux, y attacher du stockage ou même créer une nouvelle image en fonction de son état actuel.

Par défaut, un conteneur est relativement bien **isolé** des autres conteneurs et de sa machine hôte. Vous pouvez contrôler l'isolement du réseau, du stockage ou d'autres sous-systèmes des autres conteneurs ou de la machine hôte.

Lorsqu'un conteneur est supprimé, toute modification de son état qui n'est pas stockée dans un stockage persistant disparaît.

Technologie

Docker est écrit en **Go** et profite de plusieurs fonctionnalités du noyau Linux :

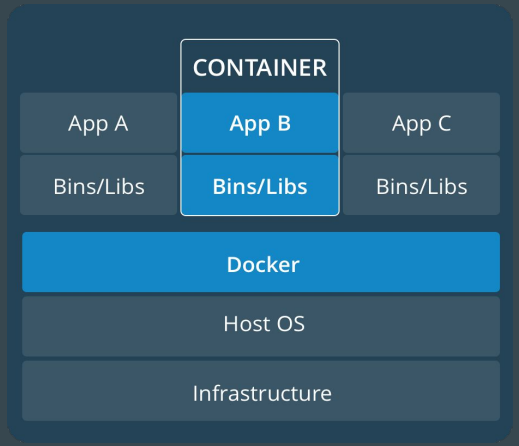
- Espaces de noms (namespaces) : Docker utilise les espaces de noms pour **isoler** l'espace de travail du conteneur. Chaque aspect d'un conteneur s'exécute dans un espace de noms distinct et son accès est limité à cet espace de noms. Docker Engine utilise des espaces de noms suivants sous Linux : pid, net, ipc, mnt et uts.
- Groupes de contrôle (cgroups) : Docker utilise les groupes de contrôle (cgroups) sur Linux pour **partager** les ressources matérielles disponibles avec les conteneurs et d'appliquer éventuellement des **limites et des contraintes**.
- **UnionFS** : Docker utilise le système de fichiers Union (et plusieurs variantes notamment AUFS, btrfs, vfs et DeviceMapper) pour fournir les blocs de construction des conteneurs. UnionFS fonctionne en créant des couches, ce qui le rend très léger et rapide.

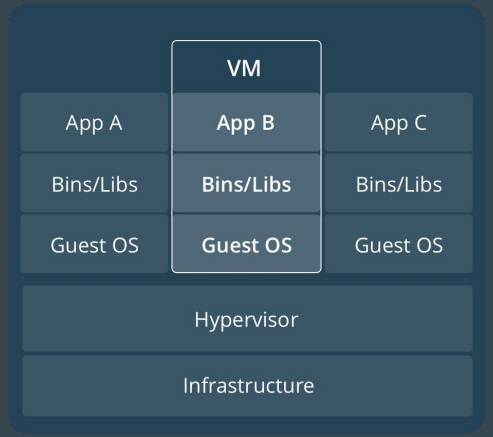
Docker Engine combine les espaces de noms, les groupes de contrôle et UnionFS dans un *wrapper* appelé format de conteneur. Le format de conteneur par défaut est **`libcontainer`**.

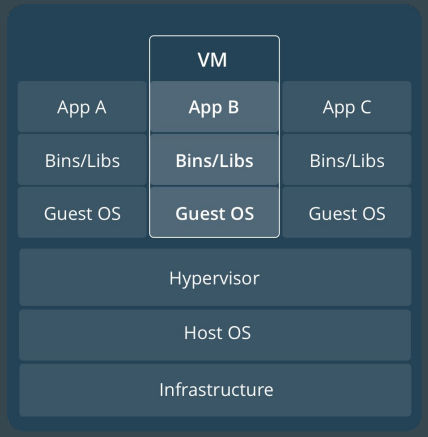
Comparaison

Un **conteneur** s'exécute nativement sous Linux et partage le noyau de la machine hôte avec d'autres conteneurs. Il exécute simplement un processus ce qui le rend léger.

Une **machine virtuelle** (VM) exécute un système d'exploitation «invité» complet avec un accès virtuel aux ressources de l'hôte via un hyperviseur. En général, les machines virtuelles “consomment” plus de ressources que les besoins réelles de l'application mais offrent une meilleur isolation.







Liens

- Docker Engine :
  - https://docs.docker.com/engine/
  - https://docs.docker.com/engine/install/
  - https://docs.docker.com/engine/reference/commandline/container_run/
- Guide de démarrage :
  - https://docs.docker.com/get-started/part1/
  - https://docs.docker.com/get-started/part2/
  - https://docs.docker.com/get-started/part3/
- **Dockerfile** :
  - https://docs.docker.com/engine/reference/builder/
  - https://docs.docker.com/get-started/part2/#sample-dockerfile
- Docker Hub : https://hub.docker.com/

Plan

- ❏ I. Commandes de base
- ❏ II. Prise en main : Hello World
- ❏ III. Test Ubuntu
- ❏ IV. Surveillance
- ❏ V. Dockerfile
- ❏ VI. Création et publication d’image
- ❏ VII. Image/Conteneur avec des options
- ❏ VIII. GUI
- ❏ IX. Exemple Dockerfile

I. Commandes de base

```
$ docker --help

# List images
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE

# List containers
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

# Display system-wide information
$ docker info

# Search the Docker Hub for images
$ docker search hello-wordl
NAME                          DESCRIPTION STARS OFFICIAL AUTOMATED
variusscent/hello-wordl                   0
q869454748/hello-wordl-nginx              0
```

I. Commandes de base : Nettoyage

```
# Arrêter tous les conteneurs
$ docker stop $(docker ps -aq)

# Supprimer tous les conteneurs
$ docker rm $(docker ps -aq)

# Supprimer toutes les images
$ docker rmi $(docker images -q)
```

II. Hello world [1 / 2]

(https://hub.docker.com/)

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

II. Hello world [2 / 2]

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              bf756fb1ae65        4 months ago        13.3kB

$ docker ps -a
CONTAINER ID IMAGE       COMMAND   CREATED       STATUS                     PORTS NAMES
9856ea6e063a hello-world "/hello"  7 minutes ago Exited (0) 7 minutes ago          trusting_tharp

# Supprimer le conteneur
$ docker rm 9856ea6e063a

# Supprimer l'image
$ docker rmi hello-world

# Un conteneur avec un nom et qui se supprime automatiquement (pas l'image)
$ docker run --rm --name=mon-conteneur hello-world
```

III. Ubuntu [1 / 3] (https://hub.docker.com/_/ubuntu?tab=tagsker.com/)

```
$ docker search ubuntu
NAME    DESCRIPTION                                     STARS  OFFICIAL  AUTOMATED
ubuntu  Ubuntu is a Debian-based Linux operating sys…   10946            [OK]
```

```
$ docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
...
docker.io/library/ubuntu:16.04
```

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              16.04               005d2078bdfa        5 weeks ago         125MB
```

```
$ docker run -it --name=mon-ubuntu-1604 ubuntu:16.04 /bin/bash
```

```
root@5b84d6832fed:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.6 LTS (Xenial Xerus)"
```

III. Ubuntu [2 / 3]

```
root@5b84d6832fed:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
...
```

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              16.04               005d2078bdfa        5 weeks ago         125MB
```

```
$ docker ps -a
CONTAINER ID IMAGE         COMMAND     CREATED       STATUS
5b84d6832fed ubuntu:16.04 "/bin/bash" 8 minutes ago Exited (0) 15 seconds ago
```

III. Ubuntu [3 / 3]

```
$ docker start mon-ubuntu-1604
mon-ubuntu-1604

$ docker exec -it mon-ubuntu-1604 /bin/bash
root@5b84d6832fed:/# exit

$ docker ps -a
CONTAINER ID IMAGE        COMMAND     CREATED        STATUS         ...
5b84d6832fed ubuntu:16.04 "/bin/bash" 11 minutes ago Up 2 minutes   ...

$ docker stop mon-ubuntu-1604
mon-ubuntu-1604

$ docker ps -a
CONTAINER ID IMAGE        COMMAND     CREATED        STATUS                     ...
5b84d6832fed ubuntu:16.04 "/bin/bash" 12 minutes ago Exited (0) 2 seconds ago   …

# Supprimer le conteneur et l’image
```

IV. Surveillance

```
$ docker ps -a

# Visualiser l'activité d'un conteneur
$ docker logs -ft d05da7223292

# Visualiser les statistiques CPU/Mémoire/Réseau
$ docker stats d05da7223292
```

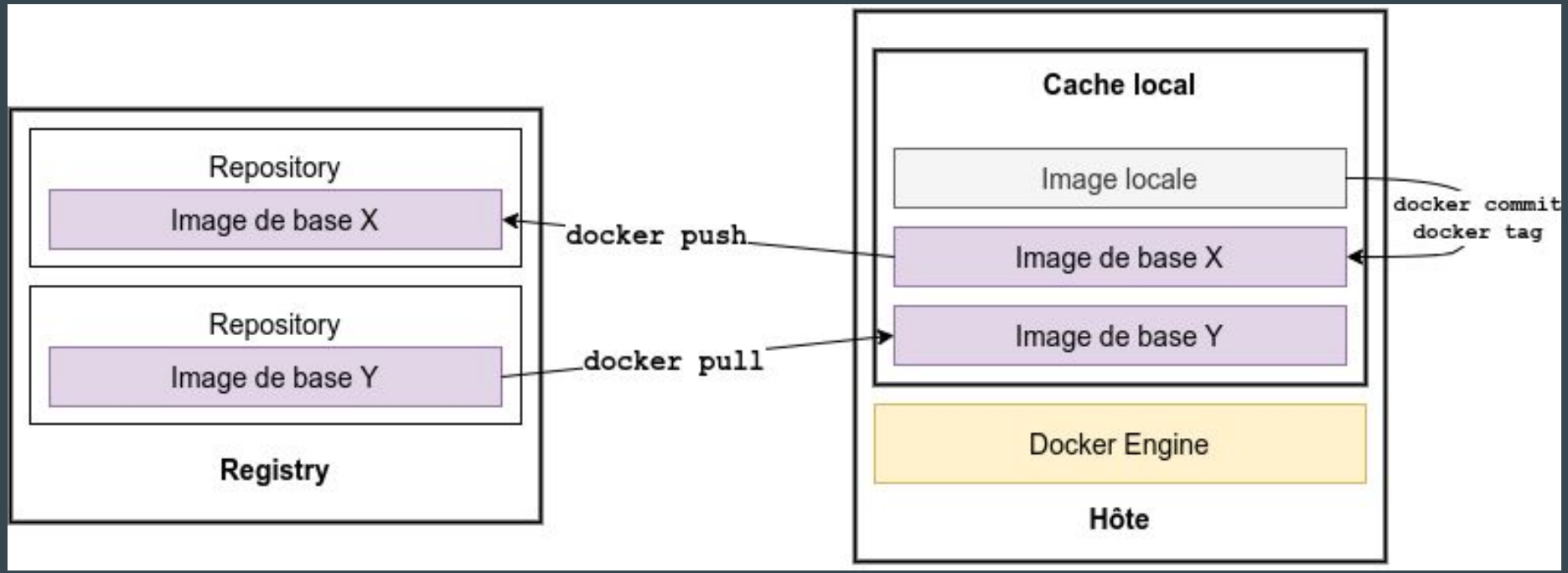
V. Dockerfile [1 / 2]

```
$ vim Dockerfile
FROM ubuntu:20.04                                   # choix de l'image
RUN apt-get update && apt-get install -y g++ make   # exécuter des commandes
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
RUN mkdir /root/test
CMD /bin/bash                                       # commande au démarrage

# Créer une image 'test'
$ docker build -t test .
Sending build context to Docker daemon  6.872GB
Step 1/4 : FROM ubuntu:20.04
20.04: Pulling from library/ubuntu
...
Step 4/4 : CMD /bin/bash
Successfully built 4bedb2cd7cf5
Successfully tagged test:latest

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
test                latest              4bedb2cd7cf5        About a minute ago   258MB
ubuntu              20.04               005d2078bdfa        5 weeks ago          73.9MB
```

V. Dockerfile [2 / 2]

```
$ docker run -it test

root@c731673bd837:/# g++ --version
g++ (Ubuntu 9.3.0-10ubuntu2) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

root@c731673bd837:/# make --version
GNU Make 4.2.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

VI. Modification d'image (commit/tag/push) [1 / 3]



VI. Modification d’image (commit/tag/push) [2 / 3]

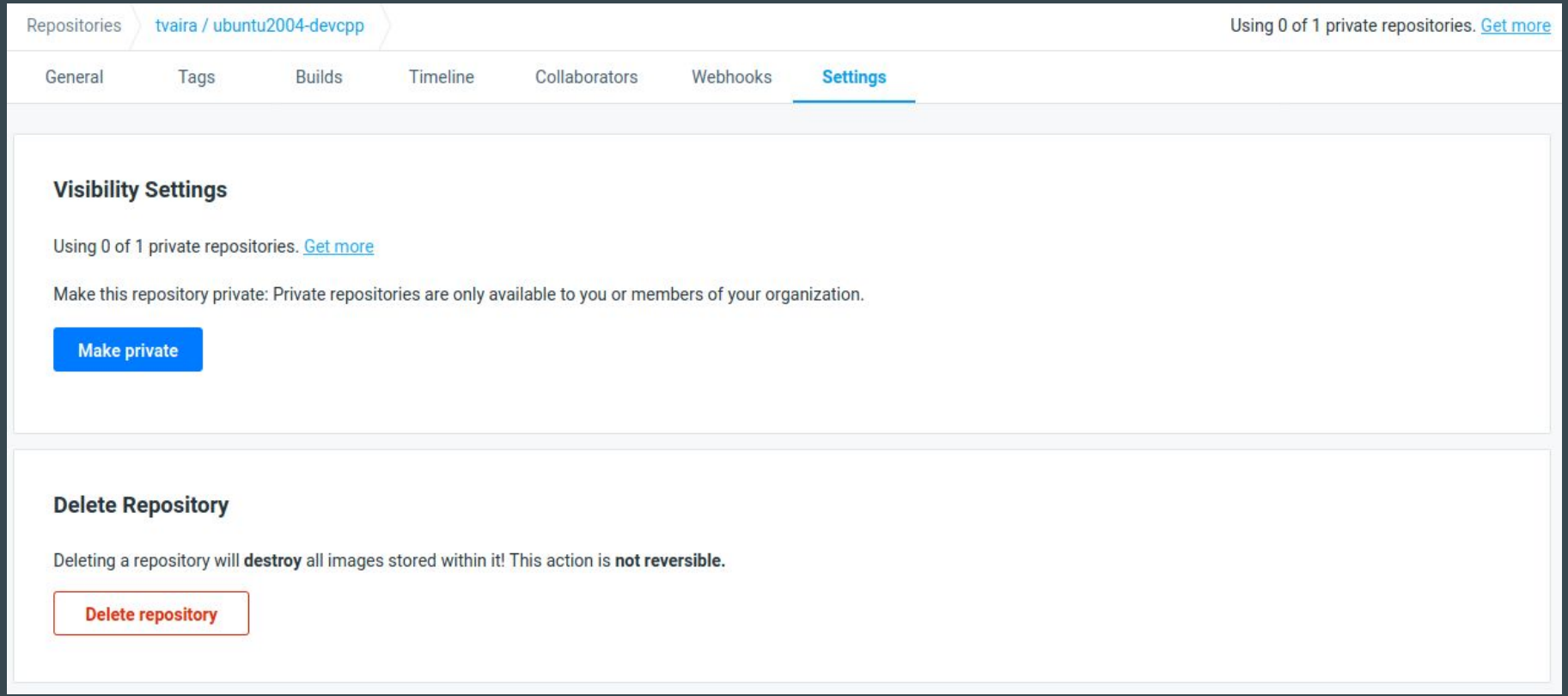
```
# Modification du conteneur
root@c731673bd837:/# apt update
root@c731673bd837:/# apt install libcppunit-dev bcpp subversion

# Lister le CONTAINER ID
$ docker ps -a

# Commit d’une nouvelle image
$ docker commit -m "ajout cppunit, bcpp et subversion" -a "tv" c731673bd837
tv/ubuntu2004-devcpp

# Lister les images
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
tv/ubuntu2004-devcpp    latest              063f852a18e7        15 seconds ago      297MB
test                    latest              c731673bd837        46 minutes ago      258MB
ubuntu                  20.04               1d622ef86b13        5 weeks ago         73.9MB
```

VI. Modification d’image (commit/tag/push) [3 / 3]

```
# Se connecter à Docker Hub https://hub.docker.com/
$ docker login -u tvaira
Password: *********
WARNING! Your password will be stored unencrypted in /home/tv/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded

# Tag d’une nouvelle image (tv -> tvaira)
$ docker tag tv/ubuntu2004-devcpp tvaira/ubuntu2004-devcpp

# Transmet l’image à Docker Hub https://hub.docker.com/
$ docker push tvaira/ubuntu2004-devcpp
```

Docker Hub (https://hub.docker.com/) [1 / 2]

Docker Hub (https://hub.docker.com/) [2 / 2]



VII. Image/Conteneur avec des options [1 / 2]

CLI

$ docker container run [OPTIONS] IMAGE [CMD] [ARG...]

Les options importantes :

```
--name              Assign a name to the container
--interactive , -i  Keep STDIN open even if not attached
--tty , -t          Allocate a pseudo-TTY
--env , -e          Set environment variables
--rm                Automatically remove
--user , -u         Username or UID (<name|uid>[:<group|gid>])
--volume , -v       Bind mount a volume
--workdir , -w      Working directory inside the container
```

Voir aussi :

```
# Copy files/folders between a container and the local filesystem
$ docker container cp
```

Dockerfile

CMD (*only one*)

ENV

USER
VOLUME
WORKDIR

ADD
COPY

Voir aussi : RUN, ENTRYPOINT, SHELL

VII. Image/Conteneur avec des options [2 / 2]

```
$ docker run -it --user=$(id -u $USER):$(id -g $USER)
--workdir="/home/$USER" --volume="/home/$USER:/home/$USER"
--volume="/etc/group:/etc/group:ro" --volume="/etc/passwd:/etc/passwd:ro"
--volume="/etc/shadow:/etc/shadow:ro"
--volume="/etc/sudoers.d:/etc/sudoers.d:ro" test /bin/bash

tv@b94a9d926fae:~$ id
uid=1026(tv) gid=65536(tv) groups=65536(tv)

tv@b94a9d926fae:~$ pwd
/home/tv

tv@b94a9d926fae:~$ exit

$ cat /etc/group | grep -E '^tv'
tv:x:65536:
$ cat /etc/passwd | grep -E '^tv'
tv:x:1026:65536:Vaira Thierry,,,:/home/tv:/bin/bash
```

VIII. GUI [1 / 4]

```
$ vim Dockerfile
FROM ubuntu:20.04
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y locales qt5-default qt5-qmake qtcreator
RUN sed -i -e 's/# en_US.UTF-8 UTF-8/fr_FR.UTF-8 UTF-8/' /etc/locale.gen && \
    dpkg-reconfigure --frontend=noninteractive locales && \
    update-locale LANG=fr_FR.UTF-8
ENV LANG fr_FR.UTF-8
ENV LANGUAGE fr_FR:fr
ENV LC_ALL fr_FR.UTF-8
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
CMD /usr/bin/qtcreator
```

```
# Créer une image ‘ubuntu2004-qtcreator’
$ docker build -t ubuntu2004-qtcreator .
...
```

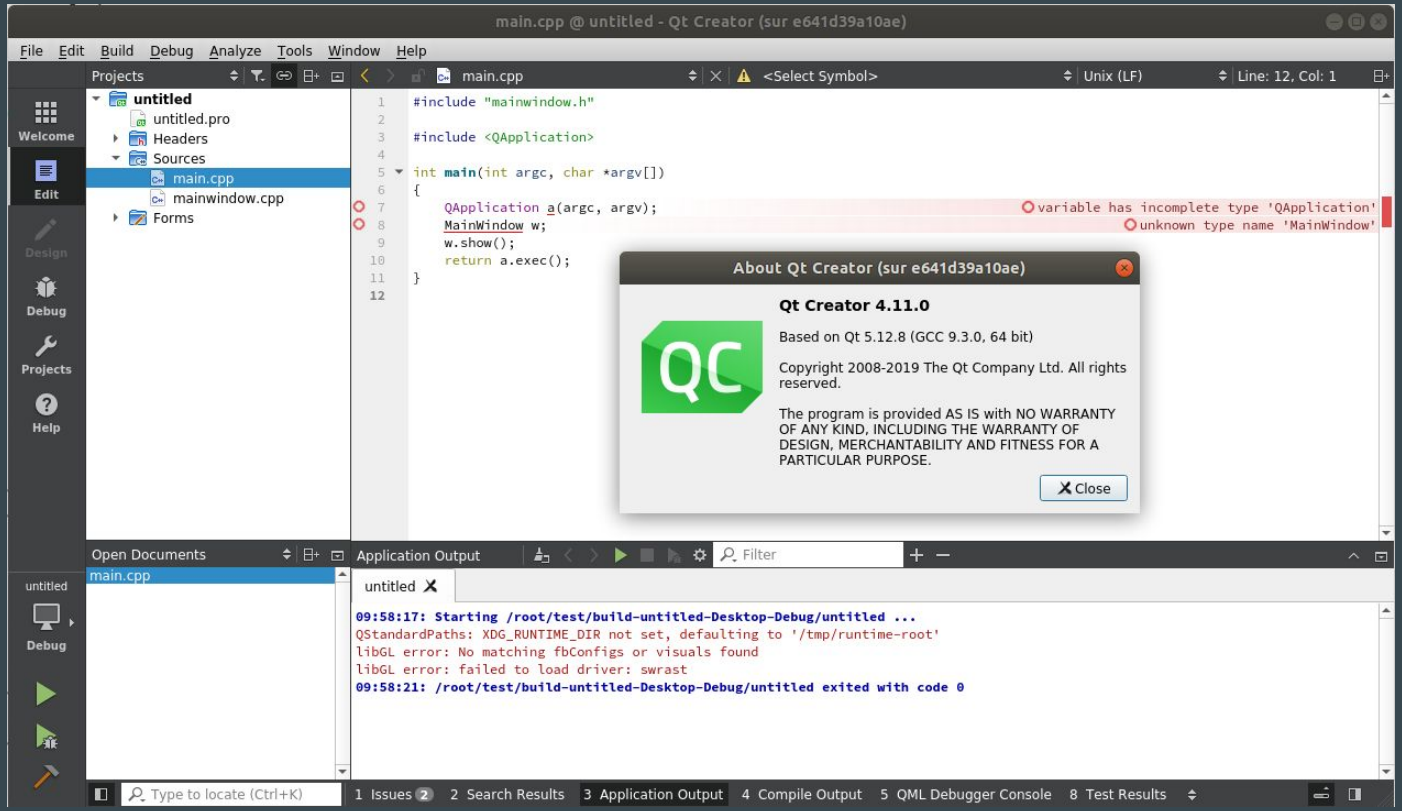
VIII. GUI [2 / 4]

```
$ xhost +local:docker     # autoriser les connexions au serveur X
# ou :
$ xhost +local:root

$ docker run -it --env="DISPLAY" --env="QT_X11_NO_MITSHM=1"
--volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" ubuntu2004-qtcreator
```

Pour le support des cartes **nvidia** dans Docker : https://nvidia.github.io/nvidia-docker/

```
$ sudo apt-get purge nvidia-docker
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
$ curl -s -L
https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee
/etc/apt/sources.list.d/nvidia-docker.list

$ sudo apt-get update
$ sudo apt-get install nvidia-docker2
$ sudo pkill -SIGHUP dockerd
```

VIII. GUI [3 / 4]


main.cpp @ untitled - Qt Creator (sur e641d39a10ae)
File Edit Build Debug Analyze Tools Window Help
Projects
untitled
untitled.pro
Headers
Sources
main.cpp
mainwindow.cpp
Forms
Welcome
Edit
Design
Debug
Projects
Help
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
QApplication a(argc, argv);
MainWindow w;
w.show();
return a.exec();
}
variable has incomplete type 'QApplication'
unknown type name 'MainWindow'
About Qt Creator (sur e641d39a10ae)
Qt Creator 4.11.0
Based on Qt 5.12.8 (GCC 9.3.0, 64 bit)
Copyright 2008-2019 The Qt Company Ltd. All rights reserved.
The program is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
Close
Open Documents
main.cpp
Application Output
untitled
09:58:17: Starting /root/test/build-untitled-Desktop-Debug/untitled ...
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
09:58:21: /root/test/build-untitled-Desktop-Debug/untitled exited with code 0
Type to locate (Ctrl+K)
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 QML Debugger Console 8 Test Results

VIII. GUI [4 / 4]

```
# Create a file named qtcreator-4.11 in /usr/bin
$ sudo touch /usr/bin/qtcreator-4.11

# Make it executable
$ sudo chmod +x /usr/bin/qtcreator-4.11

# Edit it
$ sudo vim /usr/bin/qtcreator-4.11
#!/bin/sh
set -eu
xhost +local:docker
docker run -it --rm --name=qtcreator-4.11 --user=$(id -u $USER):$(id -g $USER)
--workdir="/home/$USER" --volume="/home/$USER:/home/$USER"
--volume="/etc/group:/etc/group:ro" --volume="/etc/passwd:/etc/passwd:ro"
--volume="/etc/shadow:/etc/shadow:ro" --volume="/etc/sudoers.d:/etc/sudoers.d:ro"
--env="DISPLAY" --env="QT_X11_NO_MITSHM=1" --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw"
ubuntu2004-qtcreator /usr/bin/qtcreator "$@"
xhost -local:docker
```

IX. Exemple Dockerfile [1 / 2]

```
FROM ubuntu:20.04
LABEL maintainer="tvaira@free.fr"
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y sudo locales qt5-default qtbase5-private-dev
qt5-qmake qtcreator g++ fonts-ubuntu
RUN apt-get install -y libqwt-qt5-dev libqt5multimedia5 qtmultimedia5-dev
libqt5quickcontrols2-5 libqt5serialport5-dev libqt5sql5-mysql libqt5webkit5-dev
libqt5charts5-dev libqt5bluetooth5 qtconnectivity5-dev libqt5gamepad5-dev libcppunit-dev
RUN apt-get install -y bash-completion vim git subversion doxygen-gui graphviz
RUN sed -i -e 's/# en_US.UTF-8 UTF-8/fr_FR.UTF-8 UTF-8/' /etc/locale.gen && \
    dpkg-reconfigure --frontend=noninteractive locales && \
    update-locale LANG=fr_FR.UTF-8
ENV LANG fr_FR.UTF-8
ENV LANGUAGE fr_FR:fr
ENV LC_ALL fr_FR.UTF-8
RUN echo "Set disable_coredump false" >> /etc/sudo.conf
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
...
```

IX. Exemple Dockerfile [1 / 2]

```
...
ENV USERNAME iris
ENV PASSWORD password
# Remplacer avec votre UID/GID
RUN export uid=1026 gid=65536 && \
    useradd --uid ${uid} --create-home $USERNAME && \
    echo "$USERNAME:$PASSWORD" | chpasswd && \
    usermod --shell /bin/bash $USERNAME && \
    usermod -aG sudo $USERNAME && \
    echo "$USERNAME ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers.d/$USERNAME && \
    chmod 0440 /etc/sudoers.d/$USERNAME && \
    groupmod --gid ${gid} $USERNAME && \
    chown ${uid}:${gid} -R /home/$USERNAME && \
    mkdir -p /home/$USERNAME/Qt_Projets && chown ${uid}:${gid} -R /home/$USERNAME/Qt_Projets
USER $USERNAME
ENV HOME /home/$USERNAME
WORKDIR /home/$USERNAME/Qt_Projets
CMD /usr/bin/qtcreator
```