Utilisation d'openMPI avec Singularity

Singularity est une technologie qui vous permet d'exécuter une ou des application linux à l'intérieur d'un environnement isolé et reproductible, appelé conteneur, qui partage le noyau linux de la machine sur laquelle vous êtes. Un conteneur ressemble à une machine virtuelle, sauf qu'il n'embarque pas le noyau de la machine, juste les fichiers et les processus, ce qui lui permet de se lancer en quelque secondes et de ne pas avoir de "surcouche" entre les programmes à l'intérieur et à l'extérieur du conteneur.

Dans ce plongeon, nous allons apprendre à créer un conteneur pour effectuer des calculs avec Openmpi sur un cluster de calcul.

- prérequis : connaissance basique de Linux, connaissance basique d'opemMPI et du travail sur un cluster de calcul.
- plongeon préalable recommandé : Utilisation basique de Singularity (Singularity_01_Run/README.md)
- hauteur du plongeoir : 3m.
- maître(s) nageur(s) : Martin Souchal (https://annuaire.in2p3.fr/agents/Y249U291Y2hhbCBNYXJ0aW4sb3U9cGVvcGxlLGRjPWluMnAzLGRjPWZy/show)

Utilisation de singularity version 2.6, disponible ici : Singularity Website (https://www.sylabs.io/docs/).

Dans ce plogeon nous allons installer un conteneur Ubuntu sur une machine CentOS 7. Il est possible d'adapter ce TP sur n'importe quelle autre OS Linux récent, il faudra remplacer yum install par le gestionnaire de paquet de la distribution hôte.

Ce TP à été rédigé pour Linux exclusivement. Une partie du TP nécessite les droits administrateurs (sudo)

Mon premier conteneur

Création de l'image

Comme avec docker, la création d'un conteneur passe par l'écriture d'un fichier décrivant la configuration du conteneur. Singularity et Docker utilisent des formats de fichiers différents, toutefois il est possible de transformer un conteneur Docker en conteneur Singularity. Nous allons commencer par créer un conteneur avec la méthode Singularity. Nous verrons comment passer d'un conteneur Docker à un conteneur Singularity dans un second temps.

Il faut installer la base du systeme d'exploitation que nous voulons avoir à l'intérieur du conteneur. Pour cela on a deux solutions :

- on écrit un fichier de "recette" qui va contenir une description de ce que nous voulons dans le conteneur.
- on crée le conteneur à partir d'un fichier Dockerfile ou d'une image docker.

1) Fichier recette singularity

Commencez par créer un fichier texte nommé example.def et indiquons que nous voulons utiliser la dernière version d'Ubuntu :

```
Bootstrap: docker
From: ubuntu:latest
```

Dans Singularity 2.6, par défaut l'image crée est immuable pour permettre la reproductibilité. Il faut donc ajouter l'option --sandbox lorsqu'on teste un nouveau conteneur. Testons la création de notre conteneur décrit dans le fichier example.def :

Si vous n'avez pas de droits superutilisateur (sudo), vous pouvez passer au point 2).

```
sudo singularity build --sandbox example.img example.def
```

Pour ajouter plus de logiciels dans le conteneur on peut lancer des commandes directement à l'intérieur du conteneur :

```
sudo singularity shell --writable example.img
Singularity: Invoking an interactive shell within container...
Singularity example.img:~> apt-get update
Singularity example.img:~> apt-get install wget
```

ou en utilisant la variable %post dans le fichier de recette :

```
%post
apt-get update && apt-get -y install wget build-essential
```

Le fichier example.def ressemble maintenant à ca :

```
Bootstrap: docker
From: ubuntu:latest

%post
apt-get update && apt-get -y install wget build-essential
```

Nous avons demandé d'installer les outils de compilations standard pour pouvoir en disposer dans le conteneur. Relancons le build :

```
sudo singularity build --sandbox example.img example.def
```

On peut écrire d'importe quelle commande bash dans le fichier de bootstrap. Par exemple, créons un répértoire dans le conteneur qui contiendra nos données :

```
Bootstrap: docker
From: ubuntu:latest

%post
apt-get update && apt-get -y install wget build-essential
mkdir /data
```

2) Depuis Docker

Si vous n'avez pas les droits d'administration sur votre machine, vous pouvez passer par une image docker existante (ici nous allons chercher une image Docker sur le Docker Hub) :

```
singularity build --sandbox example.img docker://sysmso/docker-openmpi
```

Nous avons maintenant un conteneur prêt à l'emploi qui contient un compilateur et un répertoire de travail. Nous pouvons commencer à travailler dans le conteneur.

Travailler dans le conteneur

Nous avons maintenant un conteneur basique qui contient les outils de compilations. Nous pouvons alors entrer dans le conteneur et lancer un shell pour voir le résultat :

```
singularity shell example.img
Singularity: Invoking an interactive shell within container...
Singularity.example.img> $
Singularity.example.img> $ gcc --version
gcc (Ubuntu 7.3.0-16ubuntu3) 7.3.0
Copyright (C) 2017 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.
Singularity.example.img> $ exit
```

Pour sortir du conteneur, tapez exit. La version de gcc dans le conteneur est la 7.3. Par défaut vous lancez un shell dans votre conteneur en lecture-seule, donc vous ne pourrez pas installer de logiciels supplémentaire dans votre conteneur :

```
singularity shell example.img
Singularity.example.img> $ apt install vim
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libgpm2 libpython3.5 vim-common vim-runtime
Suggested packages:
  gpm ctags vim-doc vim-scripts vim-gnome-py2 | vim-gtk-py2 | vim-gtk3-py2 | vim-athena-py2 | vim-
nox-py2
The following NEW packages will be installed:
  libgpm2 libpython3.5 vim vim-common vim-runtime
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 7682 kB of archives.
After this operation, 35.1 MB of additional disk space will be used.
W: Not using locking for read only lock file /var/lib/dpkg/lock
W: chmod 0700 of directory /var/cache/apt/archives/partial failed - SetupAPTPartialDirectory (30:
Read-only file system)
```

Pour lancer un shell dans un conteneur en écriture, il faut utiliser l'option --writable, a condition que le conteneur ait été crée avec l'option --sandbox :

```
singularity shell --writable example.img
Singularity.example.img> $ apt install vim
```

La commande ne devrait pas marcher, car vous n'êtes pas root.

Attention, avec singularity, vous ne pouvez pas changer d'utilisateur dans le conteneur : l'utilisateur hors du conteneur est le même dans le conteneur. Pour être root dans votre conteneur, il faut donc être root sur la machine hôte.

Vous pouvez executer les commandes suivantes dans le conteneur pour voir la différence avec votre système :

```
singularity shell -p example.img
Singularity.example.img> $ top
```

Si vous executez la commande top en dehors du container, vous avez beaucoup plus de processus. A l'intérieur du conteneur vous êtes isolés du reste de la machine avec l'option -p.

Par défaut lorsque vous lancez un shell dans votre conteneur, vous avez accès à tous les fichiers de votre machine locale :

```
touch toto.txt
singularity shell example.img
Singularity.example.img> $ ls
Singularity.example.img> $ toto.txt
```

Si vous voulez lancer un shell sans que le conteneur puisse avoir accès aux fichiers locaux, vous pouvez utiliser l'option -H pour utiliser un autre repertoire comme /home :

```
mkdir -p /tmp/temp_home_${USER}
singularity shell -H /tmp/temp_home_${USER}:/home/${USER} example.img
Singularity.example.img> $ ls
```

L'intêret du conteneur est de produire un résultat reproductible dans n'importe quel environnement : il est par conséquent important de s'isoler au maximum du système de fichier local (des fichiers cachés pourraient avoir une influence sur le code que vous exécutez).

Pour ajouter des fichiers à l'intérieur de votre container, vous pouvez rajouter une section %files (l'équivalent d'un cp entre l'hote et le conteneur) :

```
%files
tp-singularity/mpi-ping.c /data/mpi-ping.c
```

Votre fichier singularity complet ressemble maintenant à ça :

```
Bootstrap: docker
From: ubuntu:latest

%files
tp-singularity/mpi-ping.c /data/mpi-ping.c

%post
apt-get update && apt-get -y install wget build-essential
mkdir /data
```

Relancez le build, et votre fichier sera inséré dans le conteneur :

```
sudo singularity build --sandbox example.img example.def
singularity shell example.img
Singularity.example.img> $ ls /data/mpi-ping.c
```

Nous avons maintenant un conteneur qui contient nos outils de compilation et notre fichier à compiler.

Executer votre container sur un cluster de calcul

Créons un container avec une application et son environnement

Le conteneur que nous avons crée contient Ubuntu avec notre environnement de compilation. Allons plus loin et ajoutons une application MPI compilée dans le container.

Pour cela nous allons avoir besoin d'installer openmpi dans le container. Dans ubuntu, pour avoir le paquet openmpi il faut utiliser le repository universe.

Voilà ce que donne le fichier singularity final avec toutes les dépendances requises (appelons-le example2.def) :

```
Bootstrap: docker
From: ubuntu:latest

%runscript
mpicc /mpi-ping.c


%files
tp-singularity/mpi-ping.c /mpi-ping.c

%environment
VARIABLE=MEATBALLVALUE
export VARIABLE

%labels
AUTHOR souchal@apc.in2p3.fr

%post

apt-get update && apt-get -y install software-properties-common wget build-essential sgml-base rsync
xml-core openssh-client
add-apt-repository universe
apt-get update && apt-get -y install cmake git gfortran openmpi-common openmpi-bin libopenmpi-dev
apt-get clean

mkdir /data
```

Pour information, voici l'équivalent de ce fichier avec Docker :

```
FROM ubuntu:latest

RUN apt-get update && \
    apt-get install -y wget software-properties-common build-essential sgml-base rsync xml-core
openssh-client && \
    apt-get clean

RUN add-apt-repository universe && \
    apt-get update && \
    apt-get -y install cmake git gfortran openmpi-common openmpi-bin libopenmpi-dev && \
    apt-get clean

RUN mkdir /data

ENTRYPOINT ["echo","Le runscript est la commande par défaut du conteneur !"]

ADD ./mpi-ping.c /mpi-ping.c
```

On peut maintenant contruire le container :

```
sudo singularity build example2.img example2.def
```

Le runscript peut être executé avec la commande run :

```
singularity run example2.img
```

On peut ensuite compiler le fichier mpi-ping.c avec le compilateur inclus dans le conteneur :

```
singularity exec example2.img mpicc tp-singularity/mpi-ping.c -o ./mpi-ping
```

Une fois la compilation terminée, on lance le test avec mpirun et le fichier que nous venons de compiler :

```
mpirun -np 20 singularity exec example2.img ./mpi-ping
```

La commande mpirun doit être disponible sur la machine hôte, c'est cette commande qui va dispatcher le conteneur sur les CPU physiques de la machine. De plus il faut que la version locale de MPI soit compatible avec celle utilisée pour compiler le fichier dans le conteneur.

Lancer un programme conteneurisé en batch

Il faut envoyer votre conteneur sur le cluster, par exemple via scp.

- Avec Torque Scheduler, avec 8 coeurs sur le même noeud :

```
#!/bin/bash
#PBS -N SINGULARITY
#PBS -l nodes=1:ppn=8,mem=2gb,walltime=24:00:00
export SCRATCH="/scratch/$USER.$PBS_JOBID"
mpirun /usr/local/bin/singularity exec /home/${USER}/example2.img /usr/bin/mpi-ping >> output.txt
```

Le résultat :

```
mpi-ping: ping-pong
nprocs=8, reps=10000, min bytes=0, max bytes=0 inc bytes=0
4 pings 5
6 pings 7
0 pings 1
2 pings 3
  4 pinged   5:        0 bytes      0.36 uSec     0.00 MB/s
  2 pinged   3:        0 bytes      0.36 uSec     0.00 MB/s
  0 pinged   1:        0 bytes      0.37 uSec     0.00 MB/s
  6 pinged   7:        0 bytes      0.36 uSec     0.00 MB/s
```

- Avec Torque Scheduler, avec 16 coeurs sur 2 noeuds différents :

```
#!/bin/bash
#PBS -N SINGULARITY
#PBS -l nodes=2:ppn=4,mem=2gb,walltime=24:00:00
export SCRATCH="/scratch/$USER.$PBS_JOBID"
mpirun /usr/local/bin/singularity exec /home/${USER}/example2.img /usr/bin/mpi-ping >> output.txt
```

Le résultat :

```
mpi-ping: ping-pong
nprocs=16, reps=10000, min bytes=0, max bytes=0 inc bytes=0
4 pings 5
6 pings 7
0 pings 1
2 pings 3
10 pings 11
12 pings 13
14 pings 15
8 pings 9
  0 pinged   1:        0 bytes      0.56 uSec     0.00 MB/s
  6 pinged   7:        0 bytes      0.54 uSec     0.00 MB/s
  4 pinged   5:        0 bytes      0.52 uSec     0.00 MB/s
  2 pinged   3:        0 bytes      0.52 uSec     0.00 MB/s
  8 pinged   9:        0 bytes      0.52 uSec     0.00 MB/s
 14 pinged  15:        0 bytes      0.52 uSec     0.00 MB/s
 10 pinged  11:        0 bytes      0.53 uSec     0.00 MB/s
 12 pinged  13:        0 bytes      0.52 uSec     0.00 MB/s
```

Sortie de bain

Merci d'envoyer quelques commentaires aux maîtres nageurs :

- Environnement de travail opérationnel ? oui [ ], non [ ]
- Vous aviez les pre-requis ? oui [ ], non [ ]
- Comment jugez-vous la difficulté du plongeon ? trop simple [ ], adapté [ ], trop compliqué [ ]
- Durée du plongeon ? trop court [ ], 15-20 minutes [ ], trop long [ ]
- Pourquoi avez-vous choisi ce plongeon ? :
- Signalement de typos, erreurs, etc :
- Commentaires libres :
