Créer un conteneur avec Singularity

Rémy Dernat

Version 1.6, 2018-17-05

Table of Contents

From : Alexandre Dehne Garcia <alexandre.dehne-garcia@inra.fr>, Martin Souchal <souchal@apc.in2p3.fr>,

Origine : https://gitlab.in2p3.fr/souchal/tp-singularity

<u>A propos de moi</u> <color rgb="#999999">[1: http://www.isem.univ-montp2.fr/recherche/les-plate-formes/bioinformatique-labex/personnel/dernat-remy/]</color> : membre de l’ISE-M <color rgb="#999999">[2: http://www.isem.univ-montp2.fr]</color> et d’une plateforme affiliée à un LabEx <color rgb="#999999">[3: http://www.labex-cemeb.org/]</color>.

J'ai déjà présenté Singularity à 2 reprises :

- Lors des JDEVs en 2017 lors d'une session *What place for containers in the HPC world ?*
- Lors d'une journée dédiée au conteneurs en mars 2018, organisée par le CNES *HPC_Docker_et_Singularity*

Ce projet est disponible ici et peut être cloné simplement avec `git` :

```
git clone http://gitlab.mbb.univ-montp2.fr/lxdsingu/atelier-lxdsingu.git
# puis on construit le html avec asciidoctor
cd ust4hpc
asciidoctor -d book -a toc tpSingu.adoc
asciidoctor -d book -a toc bonnes-pratiques.adoc
```

Nous allons utiliser la version `2.4.6` de Singularity, disponible ici : Singularity Website.

Dans ce TP nous allons installer singularity dans une VM ubuntu sur le cloud Amazon (utilisateur/mot de passe : singularity/singu_aws). Puis nous allons construire des conteneurs singularity que nous lancerons sur le cluster Muse du meso-centre meso@LR.

Chacun aura sa VM ubuntu vide prête à à l'installation de Singularity. Référez-vous à l'etherpad pour savoir l'IP de la VM qui vous est attribuée.

Si Singularity n'est pas installé sur votre cluster alors demandez son installation à l'équipe système : http://singularity.lbl.gov/install-request

HELP: https://groups.google.com/a/lbl.gov/forum/#!forum/singularity Pour les admins : https://singularity-container.slack.com/ http://singularity.lbl.gov/faq

Chapitre I

Mon premier conteneur

- Installation de Singularity
- Vérification et principe de Singularity
- création d'un conteneur
- Travailler dans le conteneur

Installation de singularity

Se logguer sur la vm amazon

```
sudo apt-get update --fix-missing
sudo apt-get install -y squashfs-tools automake autoconf libtool python git make
libarchive13 libarchive-dev
VERSION=2.4.6
wget
https://github.com/singularityware/singularity/releases/download/$VERSION/singularity-
$VERSION.tar.gz
tar xvf singularity-$VERSION.tar.gz
cd singularity-$VERSION
./configure --prefix=/usr/local
make
sudo make install
```

L'aide de singularity est accessible via la commande suivante :

```
singularity help
USAGE: singularity [global options...] <command> [command options...] ...

GLOBAL OPTIONS:
    -d|--debug    Print debugging information
    -h|--help     Display usage summary
    -s|--silent   Only print errors
    -q|--quiet    Suppress all normal output
       --version  Show application version
    -v|--verbose  Increase verbosity +1
    -x|--sh-debug Print shell wrapper debugging information

GENERAL COMMANDS:
    help       Show additional help for a command or container
    selftest   Run some self tests for singularity install

CONTAINER USAGE COMMANDS:
    exec       Execute a command within container
    run        Launch a runscript within container
    shell      Run a Bourne shell within container
    test       Launch a testscript within container

CONTAINER MANAGEMENT COMMANDS:
    apps       List available apps within a container
    bootstrap  *Deprecated* use build instead
    build      Build a new Singularity container
    check      Perform container lint checks
    inspect    Display container's metadata
    mount      Mount a Singularity container image
    pull       Pull a Singularity/Docker container to $PWD

COMMAND GROUPS:
    image      Container image command group
    instance   Persistent instance command group


CONTAINER USAGE OPTIONS:
    see singularity help <command>

For any additional help or support visit the Singularity
website: http://singularity.lbl.gov/
```

Vérification et principes de singularity

Singularity est donc installé dans /usr/local. Sur un cluster, si nous souhaitons l'installer dans un chemin partagé, il faudra prendre quelques précautions : http://singularity.lbl.gov/admin-guide#prefix-in-special-places-localstatedir

Pour le mettre à jour, il suffira de supprimer le sous-dossier libexec qui contient tous les binaires

associés aux actions singularity.

A noter que la méthode d'installation conseillée sur un cluster reste l'installation par rpm.

Principe de fonctionnement général de singularity

L'utilisateur qui lance le conteneur est le même utilisateur dans le conteneur (ça peut être déroutant au départ quand on utilise d'autres solutions de conteneurisation).

Par défaut, quelques fichiers/dossiers sont bind montés de l'hôte dans le conteneur. Au minimum :

- le $HOME
- le /tmp et /var/tmp
- /etc/resolv.conf et /etc/hosts

Ce comportement peut être différent et modifié pour en rajouter/enlever en éditant la configuration de singularity (singularity.conf).

Singularity utilise à minima un chroot, le mount et le PID namespace et des actions avec un bit suid (voir note plus bas à ce sujet) : http://singularity.lbl.gov/docs-security#how-does-singularity-do-it

```
singularity selftest
```

Les 3 fichiers listés en derniers ont le bit setuid actif. Pour plus d'informations sur la sécurité dans Singularity : http://singularity.lbl.gov/docs-security

```
grep -i overlay /usr/local/etc/singularity/singularity.conf
> # ENABLE OVERLAY: [yes/no/try]
> # overlayfs will be tried but if it is unavailable it will be silently ignored.
> enable overlay = try
singularity shell docker://ubuntu
id
ls
mount
ls -l /
echo $SINGULARITY_CONTAINER
exit
```

il est possible d'utiliser singularity sans les bits setuid, mais singularity sera alors fortement limité.

Le but d'un conteneur singularity est de packager une ou plusieurs applications dans un conteneur et d'avoir le minimum d'impacts sur les performances et l'environnement système (pas de démon). Prévu au départ pour le HPC, les développeurs ont volontairement limité le nombre de fonctionnalités d'un conteneur pour laisser juste l'essentiel. Il y a également quelques facilités d'utilisation dans le cadre du HPC :

- la couche réseau qui est dans le conteneur est la même que celle de l'hôte, et l'utilisation d'applications de type MPI est également prévue,
- l'utilisation des cartes GPU nvidia est possible, avec une option dédiée,
- l'image produite est un fichier exécutable, qui peut donc être appelée directement,
- le HOME de l'utilisateur est monté par défaut dans le conteneur,
- le build de l'image se fait en root, le reste en tant que simple utilisateur.

Création de l'image

Build

Nous pouvons éxécuter notre premier conteneur depuis le hub Singularity. Il va être téléchargé, mis en espace temporaire et lancé :

```
singularity run shub://GodloveD/lolcow
```

Nous pouvons aussi juste le télécharger :

```
singularity pull shub://GodloveD/lolcow
```

Puis le lancer :

```
singularity run GodloveD-lolcow-master-latest.simg
```

L'image se présente sous la forme d'un seul fichier exécutable au format compressé squashfs. Le format ext3 et l'utilisation d'une sandbox est également possible : http://singularity.lbl.gov/docs-build-container

Comme avec docker, la création d'un conteneur passe par l'écriture d'un fichier de recette 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. L'inverse est également possible en passant par un convertisseur de recette.

Nous allons créer maintenant notre premier conteneur. Créer le fichier de recette train.def :

```
BootStrap: docker
From: ubuntu:16.04

%help
My first train container
Run the sl train

%setup
mkdir ${SINGULARITY_ROOTFS}/data

%post
apt-get -y update
apt -y install sl

%help
Help me. The train is stuck in this container.


%labels
Maintainer dehneg
Updater Rémy Dernat <remy.dernat@umontpellier.fr>
ContainerVersion v1.5
Software sl

%environment
    export LC_ALL=en_US.utf8
    export PATH=/usr/games:$PATH

%runscript
sl
echo les arguments passés au container sont "$*"
```

Ici le build est fait depuis docker (BootStrap), mais il est également possible d'utiliser un autre dépôt voir même de construire son image from scratch avec yum ou debootstrap.

Construisez le :

```
sudo singularity build train.simg train.def
```

Puis lancez le :

```
singularity run train.simg arg1 bb arg3
./train.simg a1 BB a3
```

Ou ouvrez un shell dans le conteneur :

```
singularity shell train.simg
which sl
exit
```

Vous pouvez démarrer le train ou tout autre programme/script du conteneur :

```
singularity exec train.simg /usr/games/sl
singularity exec train.simg echo toto
singularity exec train.simg cat /.singularity.d/labels.json
```

Et bien entendu à tout moment vous pouvez demander de l'aide à votre conteneur:

```
singularity help train.simg
```

Vous pouvez monter à la volée un répertoire local dans le containeur

```
singularity shell --bind /usr:/mnt/usrHost train.simg
```

En vous aidant de la documentation http://singularity.lbl.gov/docs-recipes , modifiez la recette train.def afin d'importer un fichier soulTrain.txt dans "/data" de votre conteneur et de l'afficher à la suite du passage du train (*NB* : l'idée n'est pas de refaire un bind mount, comme ci-dessus).

Supprimez votre ancienne image avant de relancer le `build`.

soulTrain.txt :

```
Best Soul Train songs are :
1. Marvin Gaye, "Got To Give It Up"

2. Earth, Wind & Fire, "Let's Groove"

3. The O'Jays, "I Love Music"

4. Slave, "Just A Touch of Love"

5. The Spinners, "It's A Shame"
```

Chapitre II

Création d'un container R

Création d'un conteneur vide

Le plus simple est de créer un conteneur basique via une recette minimaliste *min.def* :

```
BootStrap: docker
From: ubuntu:16.04
```

Avant d'aller plus loin, nous allons désactiver l'overlayfs, sinon la taille de l'image sera fortement limitée sur le cloud amazon (à cause du TMPDIR sur les instances t2.micro) :

```
sudo sed -i "s|overlay = try|overlay = no|"
/usr/local/etc/singularity/singularity.conf
```

```
sudo singularity build myRimage.simg min.def
```

Puis d'ouvrir un shell en tant que root dans ce conteneur minimaliste, il vous servira de bac à sable :

```
sudo singularity shell myRimage.simg
```

Ou plus direct :

```
sudo singularity shell docker://ubuntu:16.04
```

Notez bien toutes les commandes que vous allez utiliser, elles vous serviront à créer votre nouvelle recette sur la base de *min.def*.

dans le premier cas, vous serez en lecture seule, car l'image est déjà produite. Il faudrait convertir le format squashfs générée en ext3 pour qu'elle devienne writable (http://singularity.lbl.gov/docs-build-container#converting-containers-from-one-format-to-another)

Apps

Singularity propose une approche d'exécution modulaire avec le principe des Apps.

Construisez un conteneur avec `R` depuis `ubuntu:16.04` avec les packages littler et fortunes. Pensez à faire un lien symbolique dans le `PATH` pour que `littler` puisse être facilement exécuté (voir note ci-dessous). `littler` devra être vu comme une *app*. Vous pouvez aussi mettre `Rscript` comme une *app*.

Nous pouvons installer un package R dans notre recette ainsi (ex. avec devtools qui permet notamment d'installer des packages depuis github) :

```
echo install.packages\(\"devtools\"\, repos\=\'https://cloud.r-project.org\'\) | R
--slave
```

Dans R

```
install.packages("devtools", repos='https://cloud.r-project.org')
```

Sous ubuntu, le package R est r-base et les packages s'installent par défaut dans /usr/local/lib/R/site-library/

Fortunes peut être appelé ainsi dans R :

```
fortunes::fortune()
```

On peut écrire un simple fichier R, *fortunes.R* qui sera :

```
fortunes::fortune()
```

Qui s'exécutera ainsi :

```
r -p fortunes.R
```

en cas de problème, vous pouvez vous inspirer des recettes rocker (attention, c'est au format docker) ou celles présentes sur le singularity-hub (le bouton recherche pose parfois problème sous firefox). Au pire, vous pouvez regarder directement mes recettes sur github 1 ou celle-ci 2, en prenant soin d'enlever le superflu.

Exécution du conteneur R

Le but de l'exercice précédent est de pouvoir réussir à éxécuter le conteneur de la manière suivante :

```
ls -l /boot | awk 'BEGIN {print "size"} !/^total/ {print $5}' | singularity run --app
r myRimage.simg -de "print(summary(X[,1])); stem(X[,1])" 2>/dev/null
echo "fortunes::fortune()" | singularity run --app r myRimage.simg -p 2>/dev/null
singularity run --app r myRimage.simg -p fortunes.R
```

Au passage, nous remarquons que nous pouvons *piper* le résultat d'une commande vers le conteneur singularity.

Variables d'environnement

Il est possible de passer des variables d'environnement au runtime du conteneur, même s'il est préférable que ces dernières soient fixées dans les recettes singularity (voir `%environment`) :

```
SINGULARITYENV_TOTO=tata singularity exec myRimage.simg /bin/bash
echo $TOTO
```

D'autres variables d'environnement Singularity peuvent vous intéresser. Voir par exemple `SINGULARITY_ROOTFS` (utilisé lors du build dans la section `%setup`), `SINGULARITY_CONTAINER` ou bien encore `SINGULARITY_TMPDIR`. La doc.

Travailler sur muse

Connectez vous au cluster Muse du mésocentre meso@lr [4: Travail effectué avec le support de la Plateforme MESO@LR de l'Université de Montpellier .]:

```
ssh etu-f_singularity-xx@muse-login.hpc-lr.univ-montp2.fr
```

Puis il faut charger l'environnement 2.4 de singularity

```
module available
module load singularity/2.4 squashfs/4.3 libarchive/3.3.2
module list
singularity help
```

Vous avez maintenant utliser singularity dans votre environnement et vous pouvez lancer des images

```
singularity run shub://GodloveD/lolcow
```

Pour s'assurer que tout va bien, nous allons lancer une job test sur un conteneur de test :

```
singularity pull shub://GodloveD/lolcow
srun -p singularity --account=f_singularity ./GodloveD-lolcow-master.simg
```

Vous pouvez maintenant rapatrier sur Muse votre conteneur R et essayer de le relancer sur le cluster avec slurm.

Chapitre III

Création d’un autre conteneur, Overlay et bonnes pratiques

Au choix :

1. création d’une surcouche FS conteneur avec overlayfs : Overlay,
2. création d’un conteneur utile : Repartir avec un conteneur utile !,

Overlay

Nous allons continuer à travailler avec l'image de base *myRimage.simg*.

Nous allons complexifier un peu l'image précédente en rajoutant une couche `OverlayFS` au-dessus.

Nous allons travailler sur la VM amazon.

Il faut rebasculer l'overlay à `yes` ou `try` :

```
sed -i "s|overlay = no|overlay = try|" /usr/local/etc/singularity/singularity.conf
```

La documentation est disponible ici : http://singularity.lbl.gov/docs-overlay

Avant tout, nous vérifions la version de R dans myRimage.simg :

```
singularity run myRimage.simg --version
```

Je déconseille l'utilisation des Overlays car on sort alors de la recette et de sa répétabilité.

Rajout de la librarie raster et dismo dans une nouvelle version de R

Nous allons donc installer `R-3.4.4` (certains packages comme `rgdal` nécessite `R` > `R-3.3.0`) :

```
singularity image.create -s 1024 my-overlay.img
sudo singularity shell --overlay my-overlay.img myRimage.simg
export R_VERSION=3.4.4
echo "deb http://cran.r-project.org/bin/linux/ubuntu xenial/" >
/etc/apt/sources.list.d/r.list
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9
apt-get update
apt-get install -y r-base=${R_VERSION}* r-base-core=${R_VERSION}* r-base-
dev=${R_VERSION}* r-recommended=${R_VERSION}*
R
```

```
install.packages('raster')
install.packages('dismo')
```

Nous vérifions désormais que la nouvelle version de R est bien présente dans l'overlay :

```
singularity run -o my-overlay.img myRimage.simg --version
```

Nous allons maintenant essayer de refaire ce qui est fait sur cette page Le but n'étant pas de tout refaire/comprendre, mais simplement de relancer le code dans notre conteneur sur les données :

- le code en R

```
singularity run --app r --overlay my-overlay.img myRimage.simg mangouste_vs_cerf.R
# Le code ci-dessus ne fonctionne pas. Expliquez pourquoi ?
singularity run --overlay my-overlay.img myRimage.simg CMD BATCH --slave
mangouste_vs_cerf.R
```

Si la connexion est suffisante, transférez l'image `myRimage.simg` sans l'Overlay sur le cluster Muse et testez le.

Repartir avec un conteneur utile !

Vous pouvez désormais créer un conteneur dont vous aurez besoin. Deux choix s'offrent à vous :

- rechercher le logiciel souhaité sur le hub singularity : https://singularity-hub.org/
- le créer *de novo, via* une recette

Si vous n'avez pas d'idée, vous trouverez un code C de ping MPI ici (mpi-ring.c).

Si vous utilisez le hub singularity (shub), lorsque vous importez le conteneur, essayez les commandes d'introspection avec la commande "inspect" (http://singularity.lbl.gov/docs-inspect).

Affichez le fichier de recette du conteneur qui se trouve dans le conteneur à `/.singularity.d/Singularity`

Dans tous les cas, essayez de lancer votre conteneur sur le cluster meso@LR !

Charger l'environnement mpi (voir ici pour utiliser singularity avec mpi) sur meso@LR :

```
module load cv-standard
module load openmpi
```

Et maintenant....

Si vous êtes ici, vous avez terminé le TP.

Vous pouvez lire mes conseils en terme de bonnes pratiques : bonnes-pratiques.pdf

Vous pouvez aussi continuer à lire la documentation officielle, le fichier de configuration singularity ou expérimenter les projets de Vanessa Sochat, ici : les `sregistry` et le client `sregistry-cli` qui permet de convertir des recettes docker/singularity (python3 est nécessaire) !