Utilisation avancée de 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 d'openMPI avec Singularity (Singularity_02_openmpi/README.md)
- hauteur du plongeoir : 5m.
- maître(s) nageur(s) : Alexandre Dehne Garcia (alexandre.dehne-garcia@inra.fr), Remy Dernat (http://www.isem.univ-montp2.fr/recherche/les-plate-formes/bioinformatique-labex/personnel/dernat-remy/), 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 TP nous présumons que vous avez installé Singularity 2.6 dans une VM ubuntu. Nous allons construire des conteneurs singularity que nous lancerons sur un cluster de calcul.

Introduction

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 simple 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
- l'utilisateur est le même à l'intérieur et à l'extérieur du container
- une image peu contenir plusieurs applications
- le PID du conteneur appartient à l'utilisateur et est rattaché au processus courant

Interagir avec des images

Nous pouvons executer 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
```

NOTE: 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 (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 (https://singularityhub.github.io/singularity-cli/recipes).

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 "$*"
```

TIP: 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 (rappelez vous : il faut être root!):

```
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 (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"
```

Illustration des principes de Singularity

Dans cette section nous allons vérifier par la pratique quelques uns des grands principes de singularity vus en introduction :

- 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 simple 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,
- l'utilisateur est le même à l'intérieur et à l'extérieur du container,
- une image peut contenir plusieurs applications
- le PID du conteneur appartient à l'utilisateur et est rattaché au processus courant

même interfaces réseau entre le conteneur et l'hôte

Pour le vérifier, listez les interfaces de l'hôte:

```
sudo ifconfig
```

Puis dans un conteneur singularity.

```
sudo singularity shell docker://ubuntu
apt install net-tools
ifconfig
exit
```

Que pouvez vous conclure ?

Montages par défaut

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 en éditant la configuration de singularity (/usr/local/etc/singularity/singularity.conf).

Vérification :

```
ls /home/
touch $HOME/preuveHOME
touch /tmp/preuveTMP
touch /var/tmp/preuveVAR_TMP
cat /etc/resolv.conf
cat /etc/hosts
singularity shell docker://ubuntu
ls /home/
ls -l $HOME/preuveHOME  /tmp/preuveTMP /var/tmp/preuveVAR_TMP
cat /etc/resolv.conf
cat /etc/hosts
exit
```

Singularity utilise donc un mount sur des fichiers et repertoires. Par défaut, l'utilisateur n'a pas accès au home des autres.

Vous pouvez aussi essayer de lancer ces commandes dans un shell d'un conteneur :

```
mount
ls -l
```

userspace

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).

Vérification :

```
whoami
id
singularity shell docker://ubuntu
whoami
id
exit
```

Singularity utilise donc le user namespace.

PID rattaché au processus père

Vérification

```
ps
# rentenez le pid de ce bash dans l'hôte
singularity shell docker://ubuntu
ps
# on retrouve le PID du bash "père" de l'hôte
exit
```

Singularity utilise donc le PID namespace. Vous pouvez aussi vous amuser à comaprer via "pstree" les filiations des processus entre Docker et Singularity. C'est assez parlant !

sécurité ???

Singularity utilise le bit setuid actif sur certains de ces scripts :

- /usr/local/libexec/singularity/bin/action-suid
- /usr/local/libexec/singularity/bin/mount-suid
- /usr/local/libexec/singularity/bin/start-suid

Si vous êtes inquiet, reportez vous à la section sécurité de la documentation : http://singularity.lbl.gov/docs-security (http://singularity.lbl.gov/docs-security)

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

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 shell docker://ubuntu
echo $TOTO
exit
```

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`.

Conteneuriser une application scientifique

Immuabilité des conteneurs

Par défaut, lorsque vous créez un conteneur Singularity avec `build`, le conteneur crée est immuable ; c'est à dire que vous ne pouvez plus changer son contenu, ce qui vous assure tout autre personne qui va utiliser ce conteneur l'utilisera dans l'environnement que vous aurez configuré.

En mode développement il peut être utile de crée un conteneur que vous pourrez modifier à votre convenance pour tester vos recettes et les completer. Dans Singularity on appelle cela un conteneur 'bac à sable'.

Création d'un conteneur vide *bac à sable*

Vous pouvez créer un conteneur basique via une recette minimaliste *min.def* :

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

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

Puis d'ouvrir un shell en tant que root dans ce conteneur minimaliste :

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

Ou tout simplement :

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

ATTENTION: 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.

IMPORTANT: la base de votre *bac à sable* doit impérativement être la même que la distribution cible

Exemple d'une application R

L'éxécution d'un script R peut se faire de différentes mannières : par exemple avec le paquet `littler` ou le script `Rscript` (il existe d'autres manières)

Le but de cet exercice est de construire un conteneur avec `R` depuis `ubuntu:16.04` avec les packages littler (http://dirk.eddelbuettel.com/code/littler.html) et fortunes (https://cran.r-project.org/web/packages/fortunes/index.html). Pensez à faire un lien symbolique dans le `PATH` pour que `littler` puisse être facilement exécuté (voir note ci-dessous).

Singularity proposant une approche d'exécution modulaire avec le principe des Apps (http://singularity.lbl.gov/docs-recipes#apps), faites en sorte que `littler` et `Rscript` soient vus comme des **apps**.

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')
```

NOTE: 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
```

NOTE: en cas de problème, vous pouvez vous inspirer des recettes https://github.com/rocker-org/rocker[rocker] (https://github.com/rocker-org/rocker[rocker]) (attention, c'est au format docker) ou celles présentes sur le singularity-hub (http://www.singularity-hub.org/search) (le bouton recherche pose parfois problème sous firefox). Au pire, vous pouvez regarder directement les recettes de Rémy sur github 1 (https://github.com/remyd1/ubuntu-r-base) ou celle-ci 2 (http://singularity-hub.org/containers/1361), 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.

Exemple d'une application Python

Récupérer l'image sur le singularity Hub

```
singularity pull shub://sysmso/singularity-multinest
```

L'image récupérée s'appelle sysmso-singularity-multinest-master-latest.simg.

Exécuter un fichier de démo pour tester python :

```
singularity exec sysmso-singularity-multinest-master-latest.simg python pymultinest_demo_minimal.py
```

Création d'un autre conteneur, Overlay et bonnes pratiques

Utilisation du Singularity Hub

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

- rechercher le logiciel souhaité sur le hub singularity
- le créer *de novo, via* une recette

Le Singularity Hub est l'équivalent dans le monde Singularity du Docker Hub : c'est un catalogue de conteneur publique, sur lequel n'importe qui peut partager ses conteneurs. Le Singularity Hub est accessible à l'adresse suivante : http://www.singularity-hub.org/ (http://www.singularity-hub.org/)

Le Singularity Hub permet aussi de lier un projet Gitlab/GitHub pour automatiser le build de conteneurs dans le cloud. A chaque commit le conteneur est recrée dans le cloud.

Pour télécharger un conteneur qui vous interesse, vous pouvez utiliser le protocole 'shub' directement dans Singularity.

```
singularity pull shub://GodloveD/lolcow
singularity pull --name customname.img shub://GodloveD/lolcow
singularity pull --commit shub://GodloveD/lolcow
singularity pull --hash shub://GodloveD/lolcow
```

La syntaxe est la même qu'avec le Docker Hub, de la forme user/conteneur.

Vous pouvez aussi l'executer directement :

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

Il existe aussi un client en ligne de commande qui permet de faire des recherches :
https://singularityhub.github.io/sregistry-cli/client-hub (https://singularityhub.github.io/sregistry-cli/client-hub)

```
sregistry pull shub://GodloveD/lolcow
sregistry search shub://GodloveD/lolcow
sregistry record shub://GodloveD/lolcow
```

Lorsque vous importez le conteneur, essayez les commandes d'introspection avec la commande "inspect" (https://www.sylabs.io/guides/2.6/user-guide/environment_and_metadata.html?highlight=inspect#labels (https://www.sylabs.io/guides/2.6/user-guide/environment_and_metadata.html?highlight=inspect#labels)).

```
singularity inspect shub://GodloveD/lolcow
```

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

```
singularity exec shub://GodloveD/lolcow cat /.singularity.d/Singularity
```

Utilisation du registre IN2P3

Un registre singularity est un service qui permet d'envoyer des images et de les partager avec d'autres membres de la communauté IN2P3. Par exemple si vous avez une équipe d'astrophysiciens qui travaillent sur plusieurs cluster de calcul avec le même soft, il peut être utile de la stocker à un unique endroit accessible de n'importe ou.

Le registre IN2P3 a une interface web accessible à l'adresse suivante : https://sregistry.in2p3.fr/ (https://sregistry.in2p3.fr/)

Vous pouvez consulter la liste des conteneurs que vos collégues ont bien voulu partager en cliquant sur "All collections" : https://sregistry.in2p3.fr/collections (https://sregistry.in2p3.fr/collections)

Si vous voulez envoyer votre conteneur singularity, il faut utiliser le client sregistry et demander la création d'un compte sur le registre de l'in2p3.

Pour installer le client sregistry :

```
git clone https://github.com/singularityhub/sregistry-cli
cd sregistry-cli
python setup.py install
which sregistry
```

Pour envoyer une image il faut s'authentifier, pour cela connectez vous sur https://sregistry.in2p3.fr/ (https://sregistry.in2p3.fr/) en cliquant en haut à droite sur login. Une fois connecté, cliquez sur votre nom d'utilisateur et dans le menu déroulant sur "Token". Ensuite, créez un fichier texte dans votre home, appelé .sregistry et copiez collez le contenu du token à l'intérieur. Il faut ensuite faire un "push" sur votre fichier conteneur, avec des métadonnées qui renseigne sur son contenu (nom, tags) :

```
export SREGISTRY_CLIENT=registry
sregistry push conteneur.img --name monlabo/conteneur --tag astrophysique
```

Si vous n'avez pas de compte, vous pouvez simplement télécharger un conteneur depuis le registre. Ici on va récupérer le conteneur mpi-ping crée par le laboratoire APC (vous pouvez le visualiser sur l'interface web : https://sregistry.in2p3.fr/containers/4 (https://sregistry.in2p3.fr/containers/4)) :

```
singularity pull shub://sregistry.in2p3.fr/apc/mpi-ping:latest
```

Vous pouvez aussi l'executer directement :

```
singularity run shub://sregistry.in2p3.fr/apc/mpi-ping:latest
```

générateur de recettes

Cet exercice s'adresse à ceux qui aiment développer.

Le but est de générer à la demande des recettes singularity avec plusieurs apps au choix dans une liste.

- Créez un fichier de définition `fun.def` avec ces 3 apps : sl, lolcow et fortunecookie. Testez le conteneur en lançant les 3 apps séparément.
- Créez un répertoire nommé Apps
- Décomposez votre fichier définition :

- Dans le répertoire Apps créez 3 fichiers nommés sl, lolcow et fortunecookie. Mettez le code correspondant à chaque apps dans le fichier lui correspondant
- En dehors du répertoire Apps, créez un fichier `base.def` qui comporte tout le code de `fun.def` à l'exception des apps
- créer un script `appsGenerator` (dans le langage de votre choix) qui prenne en entrée un liste d'apps et qui génère un fichier (ou STDOUT) *.def* fonctionnel avec les apps demandées
- faites en sorte que si aucun argument n'est donné à `appsGenerator`, s'affiche alors une aide et la liste des apps possibles (ici sl, lolcow et fortunecookie).

Bien entendu, une autre app pourrait être rajoutée ultérieurement sans avoir à toucher à `appsGenerator`.

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.

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

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

ATTENTION: En science, l'utilisation des Overlays est déconseillées 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 [page](cette https://www.r-bloggers.com/simulating-animal-movements-and-habitat-use/ (https://www.r-bloggers.com/simulating-animal-movements-and-habitat-use/)) 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 (https://gitlab.in2p3.fr/alexandre.dehne-garcia/TP_singularity_EcoleConteneursProd/blob/master/data/mangouste_vs_cerf.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
```

Bonnes Pratiques

En dehors de la fabrication de recettes, il n'y a pas encore de bonnes pratiques documentées sur le site de Singularity. Ce qui suit est donc basé sur ma propre expérience.

L'utilisateur pourra soit concocter ses propres formules soit demander au support HPC de l'aider pour que les performances soient au rendez-vous. On touche ici à un point critique. Certaines plateformes ne voudront pas que des images non optimisées tournent sur leur cluster. C'est le cas des centres de calcul évalués selon leur efficacité.

Le pipeline qui va en découler sera donc différent selon les cas. Dans le cas où l'administrateur de cluster laisse faire ses utilisateurs, on peut imaginer communiquer proprement et ouvertement sur la présence et l'utilisation de singularity. Dans le second cas, on ne communiquera pas trop sur les conteneurs et on limitera l'accès des utilisateurs à la commande (voir `LIMIT CONTAINER OWNERS` dans la configuration) et éventuellement aux dépôts d'images.

En combinant l'utilisation de singularity et des modulefiles, on pourra chaîner l'appel au module singularity puis à l'image elle-même (ça permet de voir l'image comme un exécutable classique (ce qu'elle est, mais l'appel à singularity est ainsi masqué, et le PATH/et autres variables d'environnement seront aussi correctement configurés); on peut même faire un lien symbolique dans un PATH qui portera le même nom que l'exécutable conteneurisé).

Pour les utilisateurs

La fabrication des recettes fait déjà l'objet de bonnes pratiques : http://singularity.lbl.gov/docs-recipes#best-practices-for-build-recipes (http://singularity.lbl.gov/docs-recipes#best-practices-for-build-recipes)

On peut rajouter à cette liste de bonnes pratiques qu'une section `%test` est pratique pour s'assurer de certaines choses (reproductibitité... ?); de même il faut user et abuser des `%labels`.

En effet, certaines informations sont très pratiques et permettent de retrouver facilement des recettes dans les dépôts. Nous pouvons donc rajouter ce type de `%labels` :

- la version de Singularity utilisée pour le `build`,
- le créateur, la/les personne(s) qui les a modifié (avec éventuellement un log de la modification),
- la date de création et de mises à jour,
- l'application principale et les éventuelles apps (elles ont elles-mêmes leur %applabels),
- une URL vers la recette,
- toute information que vous trouverez pertinente.

Dans le cas où un Dockerfile existe déjà, on se retrouve alors confronté au choix suivant :

- convertir l'image Docker en image Singularity,
- convertir la recette Dockerfile en recette Singularity (soit https://singularityhub.github.io/singularity-cli/recipes[de (https://singularityhub.github.io/singularity-cli/recipes[de) manière automatique] ou manuellement / en s'en inspirant)

Il est préférable de choisir la seconde option car ça permet de reprendre proprement ce qui est fait, sauf dans le cas où on ne souhaite ne maintenir qu'un Dockerfile.

Il faut également faire attention au champ "From" dans une recette; en effet, nous considèrerons que si on fait un "From", on fait confiance à la recette parente (sur le Docker-hub, on prendra soin de choisir des images étiquettées "officielles"), et si possible légères (busybox, alpine...).

Bien entendu, on pense à versionner les recettes. Si on utilise un dépôt github, on pourra le coupler aisément au singularity-hub (https://github.com/singularityhub/singularityhub.github.io/wiki/Automated-Build), pour faire des *builds* de conteneurs à chaque *commit*.

Pour les administrateurs

Vous aurez sûrement à reprendre les images des utilisateurs pour rajouter les spécificités de votre plateforme.

`singularity inspect ...` permet déjà d'obtenir des informations sur l'image. Si vous avez fait un pull depuis le http://www.singularity-hub.org/[singularity-hub] (http://www.singularity-hub.org/[singularity-hub]), vous pouvez aussi lire la recette dans le fichier `/.singularity.d/Singularity` du conteneur.

TIP: Il semblerait que l'outil https://github.com/singularityhub/sregistry-cli["sregistry-cli"] (https://github.com/singularityhub/sregistry-cli["sregistry-cli"]) soit capable de récupérer le contenu de la recette lorsqu'il fait un `inspect` après avoir fait le `pull`. Par ailleurs, le singularity-hub permet d'explorer le contenu des images.

Dans le fichier de configuration de Singularity `singularity.conf`, vous allez sûrement rajouter un espace scratch local à chaque noeud à bind monter. Dans ce cas, pour que ça soit exploitable dans chaque image, il faut qu'un dossier parent existe correspondant au futur point de montage.

Ainsi une simple modification de recette pourrait s'exprimer ainsi :

```
BootStrap: shub
From: nickjer/singularity-r:3.4.4

%post
mkdir /scratch
```

Bien entendu, vous pouvez imaginer des modifications bien plus complexes à rajouter en post-traitement dans une enième recette, puisqu'elles peuvent être chainées indéfiniment (à éviter).

Nous pouvons donc imaginer un premier type d'organisation avec le singularity-hub officiel pour des recettes publiques publiées sur github, puis un autre dépôt local, qui contient les recettes modifiées de ces images. Il faudra ensuite penser à une arborescence correcte si les images sont distrisbuées sur un dossier (ex: /dossier/partagé/version majeure de singularity/version mineure/nom de l'application/version majeure/version mineure/image). Cette méthode sera préférée si on souhaite limiter l'usage de Singularity sur la plateforme.

Dans le cas où on souhaite étendre au maximum l'utilisation de singularity, on préfèrera ne maintenir qu'un seul dépôt, public ou privé selon le besoin.

Les `sregistry` de Singularity répondent à ce besoin : https://github.com/singularityhub/sregistry (https://github.com/singularityhub/sregistry) Ils permettent une collaboration accrue entre les utilisateurs du dépôt. Il semblerait qu'il soit désormais possible de les relier entre eux pour former un réseau de confiance (voir https://singularityhub.github.io/interface/[ici] (https://singularityhub.github.io/interface/[ici])). Cependant, il n'y a pas encore d'outil intégré aux sregistry pour lancer un build depuis un dépôt git (contrairement au singularity-hub officiel connecté à github).

Monitoring

Pour l'instant, il n'y a pas vraiment d'équivalent à un "docker ps", sauf dans le cadre des "instances" singularity (== tâches de fond / services). Cependant, il faut voir les conteneurs Singularity comme des applications classiques, ils peuvent donc être monitorés comme n'importe quel autre job.

Pour superviser l'utilisation des namespaces, vous pouvez utiliser `lsns` et `nsenter` du package `util-linux`. Normalement, les 2 utilitaires sont disponibles à partir de la version 2.28 du package.

C'est un peu plus complexe qu'avec un `docker attach ...` :

```
lsns |grep bash
# recuperation du PID correspondant
sudo nsenter -a -t <PID>
# une fois dans le conteneur
chroot /usr/local/var/singularity/mnt/container
# /usr/local à remplacer par le prefix d'installation du ./configure ou localstatedir si utilisé
ls -l /
```

Cache et nettoyage

Il pourra s'avérer utile de vérifier le `SINGULARITY_TMPDIR`, voir éventuellement faire un *epilog* / *prolog* à votre Job Scheduler pour pouvoir fixer ce dernier à un sous-dossier de votre TMPDIR contenant le JOB ID (ex : `SINGULARITY_TMPDIR="/tmp/singularity-slurm-${SLURM_JOB_ID}"`).

Scans d'image

Avec l'aide de l'outil `clair` de `CoreOS`, il est possible de scanner des images pour chercher des vulnérabilités dans celles-ci :

```
singularity pull shub://marcc-hpc/tensorflow:1.8.0-gpu
singularity check --tag security marcc-hpc-tensorflow-1.8.0-gpu-1.8.0-gpu.simg
```

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 :
