Containers et HPC

10èmes journées mésocentres

Cédric Clerget et **Laurent Philippe**

laurent.philippe@univ-fcomte.fr

26 septembre 2017

mésocentre de calcul de franche-comté

UFC
UNIVERSITÉ DE FRANCHE-COMTÉ

Gestion de l'éco-système logiciel sur un cluster

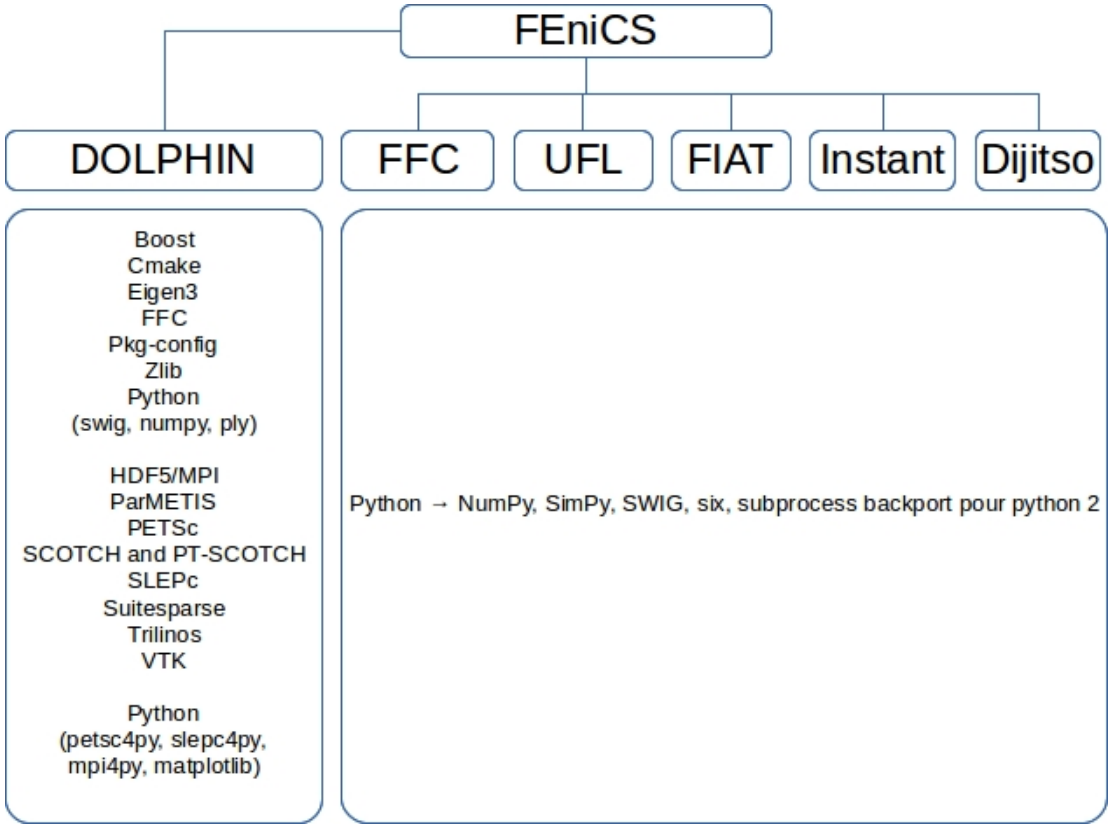
Côté administrateur

- Compilation et installation de nouveaux logiciels
- Mise à jour des logiciels
- Faciliter leur utilisation :
  - Différentes versions d'un même logiciel
  - Logiciels utilisant différentes versions de bibliothèques

Côté user

- Mettre des développements, simulations à disposition
- Replicated Sciences 

Un exemple : FEniCS et ses dépendances



A refaire à chaque mise à jour...

Problématiques rencontrées

Côté administrateur

- Compilations complexes avec beaucoup de **dépendances**
- Compilations **non reproductibles**
- Vieilles distributions **VS** nouveaux logiciels
- Compatibilté avec les environnements de calcul type MPI et GPU

Côté user

- Portabilité
- Facilité de déploiement

Solutions

- Système de modules :
  - Très utilisé sur les clusters de calcul, facilite l'utilisation, non portable
- Gestionnaire de package (Nix/Guix)
  - Gestionnaires de package multi-user et multi-version, complétement indépendant du système hôte, possibilité de créer des conteneurs, apprentissage
- Environnement virtuel (Vagrant)
  - Fait appel à la virtualisation ou aux conteneurs (LXC/Docker)
- Reproduction d'exécution (CARE, ReproZip)
  - Trace l'exécution et génère une archive (mini-conteneur) pour la reproduction. Idéal lorsqu'il n'y a pas de dépendance matérielle
- Containers
  - **Isolation "légère" de l'environnement logiciel**

Plan

Éco-système logiciel

Containers

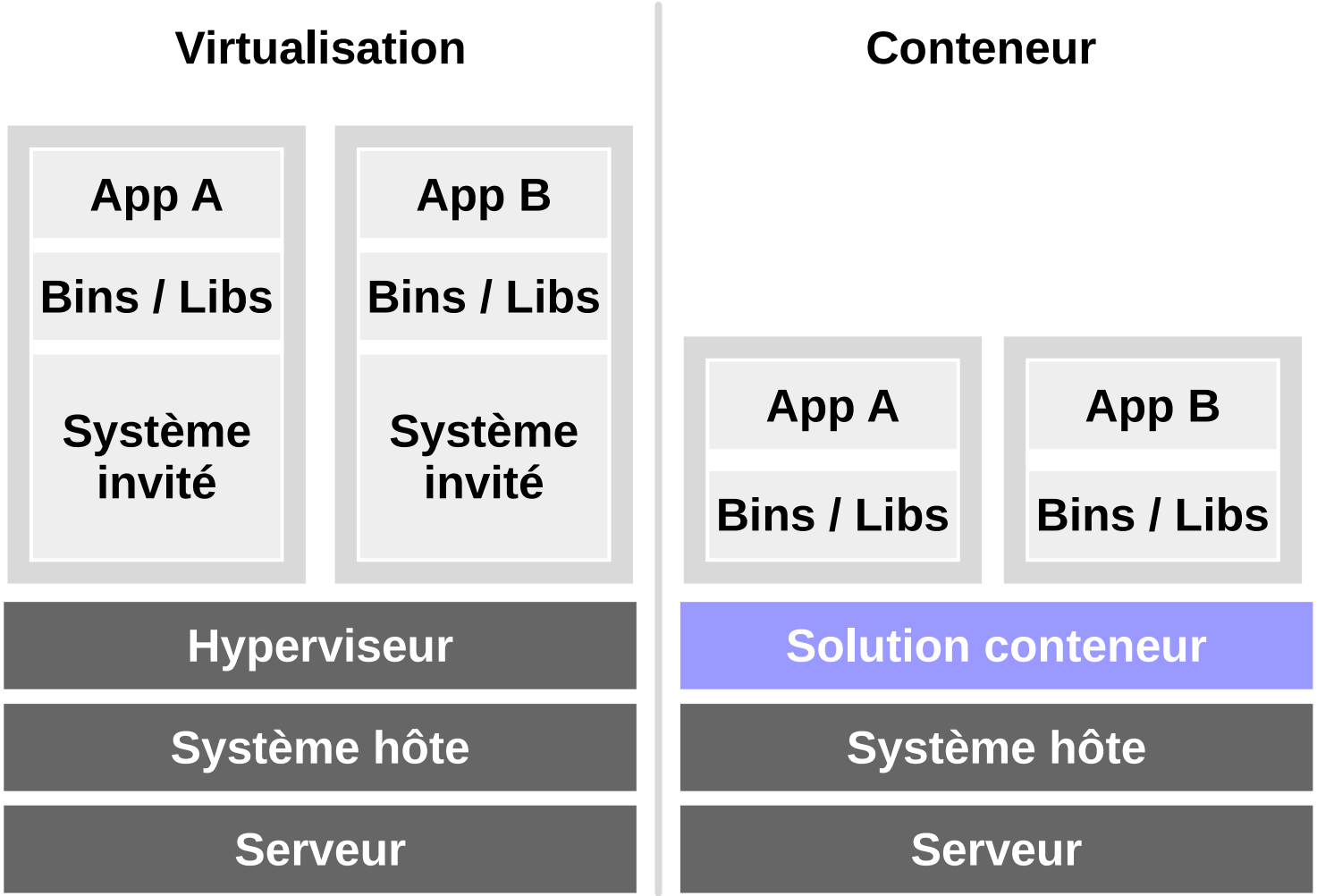
Singularity

Reproduction et diffusion avec Singularity

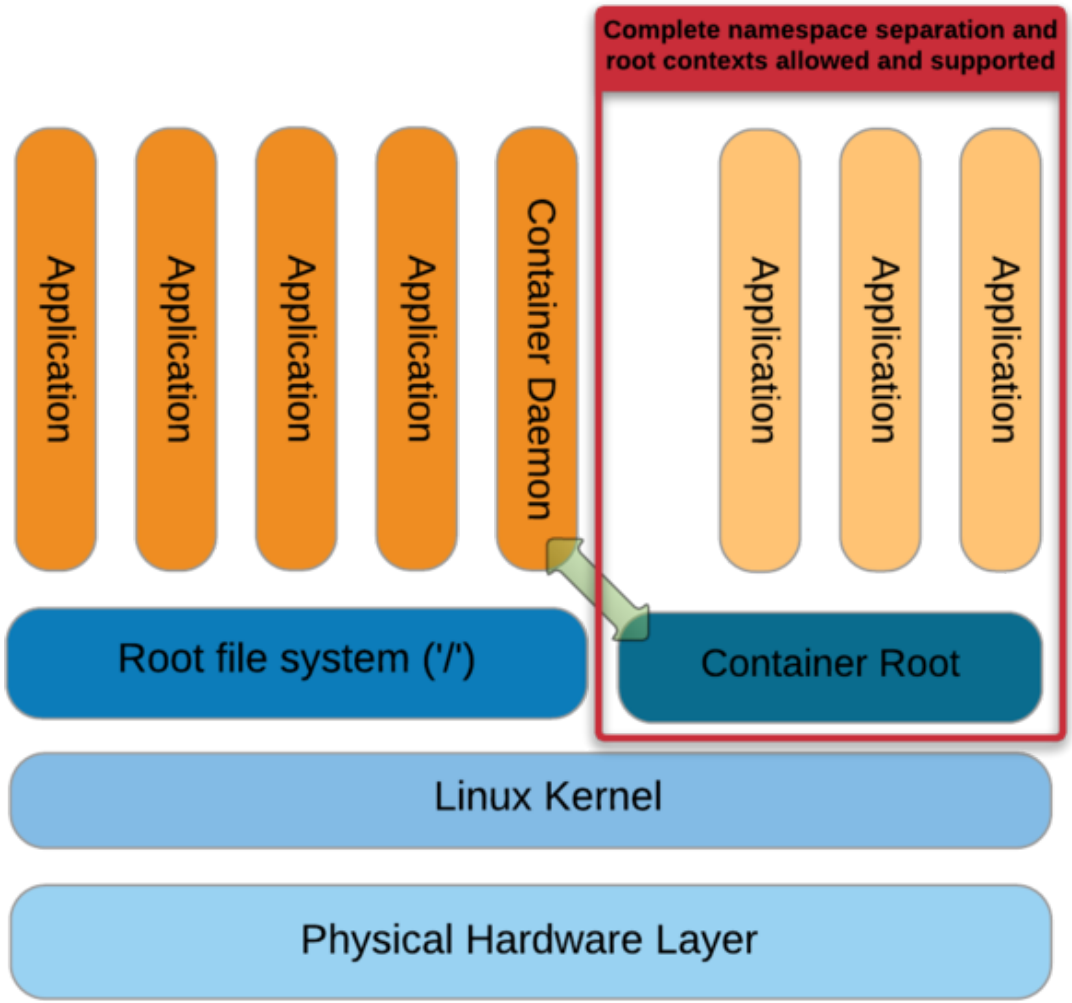
Qu'est-ce qu'un container ?

- Environnement virtuel léger :
  - espace indépendant (chroot)
  - pas de système invité
  - au sein d'une image
- Limité au logiciel
- Partage du système hôte
- Accès aux ressources systèmes et matérielles (contrôlé)

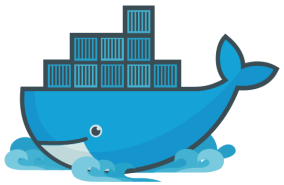
Container vs. Virtual Machine



Architecture du container



Les containers d'applications

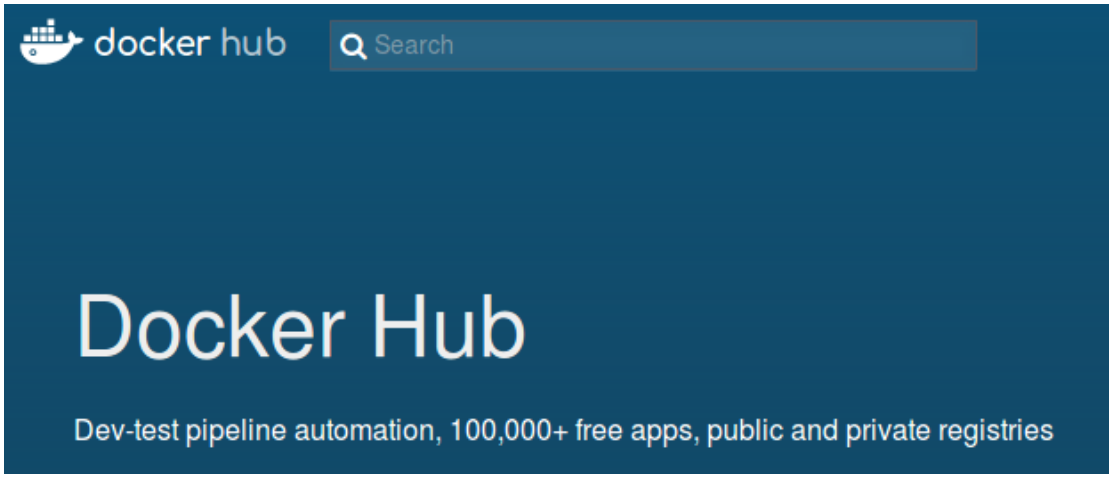


Docker	https://www.docker.com
Singularity	http://singularity.lbl.gov
Shifter	https://github.com/NERSC/shifter

Docker

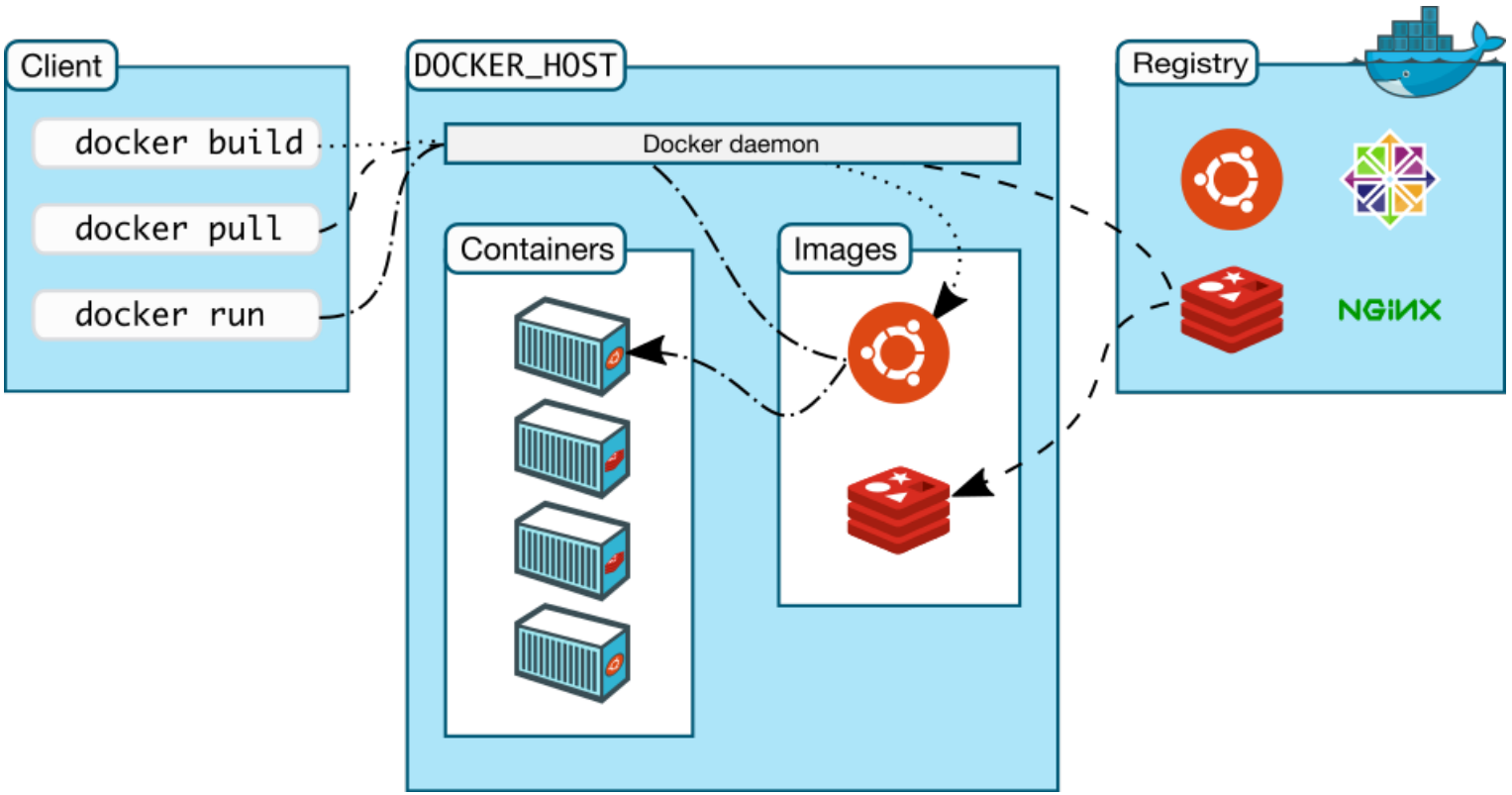
Sans doute le système de container le plus populaire :

- Jeune (2013) mais mature
- Engouement important de la communauté
- Dépôt central de plus de 100 000 images
- Très utilisé en entreprise (DevOps, Microservices)



Docker : comment ça marche ?

- Démon pour la gestion des images et des containers
- Dépôts d'images : local et distant (Hub Docker)



Docker

Avantages

- Large communauté
- Dépôt central de plus de 100 000 images

Inconvénients

- Accès root dans le conteneur
- Forte isolation, pas d'accès à l'Infiniband et aux partages réseaux
- Pas de support GPU natif

Inadapté pour le HPC

Shifter

Avantages

- Accès au dépôt central de docker via un serveur d'image local
- Accès Infiniband, partages réseaux
- Pas d'accès root
- Provisioning et lancement d'image depuis le Resources Manager

Inconvénients

- Nécessite une intégration avec le Resources Manager (epilog, prolog)
- Pour le moment pas de support GPU (shifter-gpu)

Dédié HPC, sans support GPU

Singularity

Avantages

- Accès au dépôt central de docker pour la création d'image
- Accès Infiniband, partages réseaux et GPU
- Pas d'accès root
- Utilisable avec les modules et le Resources Manager comme un logiciel classique

Inconvénients

- Quelques bugs pour certaines images Docker, facilement corrigeables
- Intégration MPI pas totalement mature, utilisation de OpenMPI 2

Testé et approuvé

Plan

Singularity

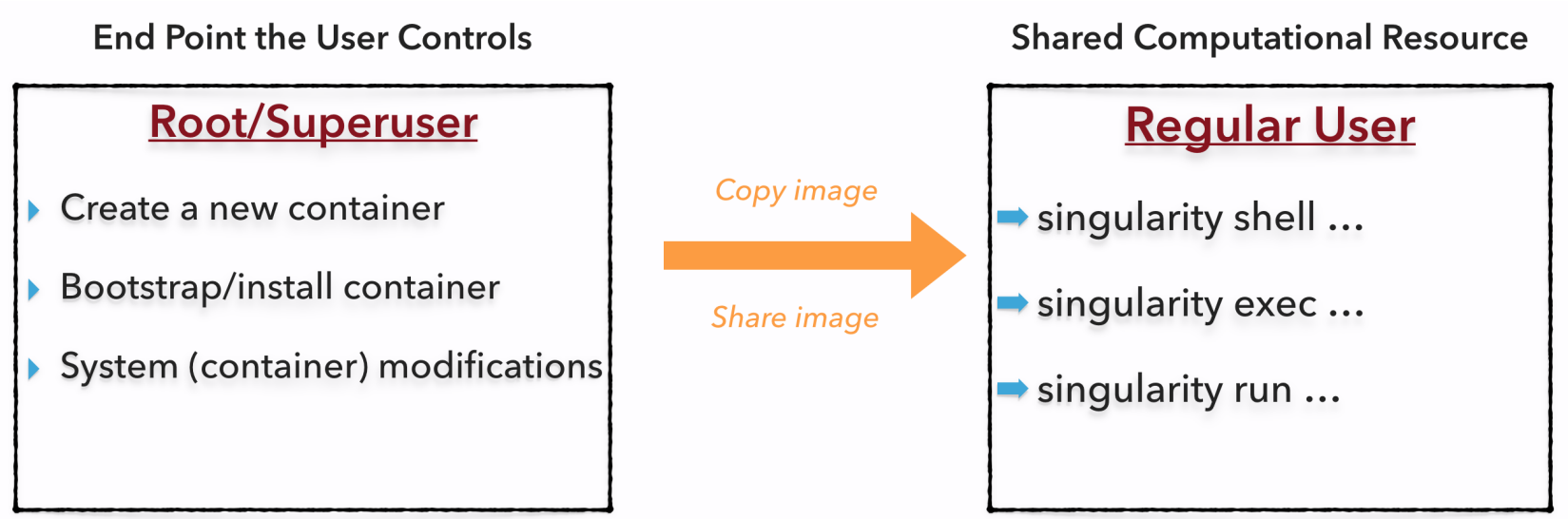
Projet développé au laboratoire LBNL pour remplir 3 critères :

- **Portabilité** entre environnements Linux
- **Mobilité** entre clusters
- **Reproductibilité** des résultats

Fonctionalités :

- Encapsulation de l'environnement utilisateur
- Droits utilisateurs identiques dans et hors container
- Containers à base d'image :
  - dépôt singularity
  - utilisation des images Docker

Aperçu



- Pas d'élévation de droits dans le container, seul root peut modifier le container
- Elévation des droits temporairement accordées à l'utilisateur (binaire setuid) pour :
  - Monter l'image du container (loop mount)
  - Modifier la racine du système (chroot)
  - Monter les systèmes de fichier virtuels (/proc, /sys, /dev, /tmp)

Utilisation (I)

Mise à disposition d'images par les administrateurs

- Création d'une image
- Création d'un environnement
- Mise en place d'un dépôt

Création d'un conteneur avec Singularity

- Création de l'image :

```
user@local:~$ sudo singularity create -s 1024 ubuntu.img
Creating a new image with a maximum size of 1024MiB...
Executing image create helper
Formatting image with ext3 file system
Done.$
```

- Import de l'environnement Ubuntu 16.04 depuis Docker :

```
user@local:~$ sudo singularity import ubuntu.img docker://ubuntu:16.04
library/ubuntu:16.04
Downloading layer: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c229
     55b4
...
Bootstrap initialization
No bootstrap definition passed, updating container
Executing Prebootstrap module
Executing Postbootstrap module
Done.$
```

- Exécution du conteneur pour modification :

```
user@local:~$ sudo singularity shell -w -s /bin/bash ubuntu.img
Singularity: Invoking an interactive shell within container...

root@local:~# apt-get update
root@local:~# apt-get install -y python
```

Création d'un environnement avec Singularity bootstrap

BootStrap : fichier de définition de la configuration

- yum (système hôte type RedHat)
- debootstrap (système hôte type Debian)
- docker (tout système)

Sections pour l'exécution contextuelle de commandes shell :

- **%setup**, hors container durant le bootstrap
- **%post**, dans le container durant le bootstrap
- **%runscript**, à chaque "run" du container

Création d'un environnement avec Singularity bootstrap

Exemple de fichier de bootstrap

```
# bootstrap.def
BootStrap: docker
From: ubuntu:16.04

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

% runscript
echo 'This is what happens when you run the container...'
```

Boostrap de l'image :

```
user@local:~$ sudo singularity create python.img
user@local:~$ sudo singularity bootstrap python.img bootstrap.def
```

Exécution dans l'environnement

- Lancer un shell dans le container :

```
user@local:~$ singularity shell python.img
Singularity: Invoking an interactive shell within container...

Singularity.ubuntu.img$>
```

- Exécuter une commande dans le container :

```
user@local:~$ singularity exec python.img python
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
>>>
```

- Lancer les commandes de la section runscript du fichier boostrap :

```
user@local:~$ singularity run python.img
This is what happens when you run the container...
user@local:~$
```

- Lancer l'image comme un exécutable (runscript) :

```
user@local:~$ ./python.img
This is what happens when you run the container...
user@local:~$
```

Utilisation (II)

Définition d'un environnement par l'utilisateur :

- Création d'une image sur machine personnelle
- Adapter image : création points de montage (Scratch :BeeGFS et Home)
- Copie sur le mésocentre
- Exécution dans l'image

Utilisation (II)

- Exécution dans l'image : intégration Resource Manager
- Lancement transparent dans le RM
- Exemple SGE : programme python

```
#!/bin/bash
#$ -q all.q
module load tools/singularity
singularity exec $SING_SPOOL/ubuntu-61 /usr/bin/python hello.py
```

- Exemple OpenMP

```
#!/bin/bash
#$ -q all.q
#$ -pe openmp N
module load tools/singularity
singularity exec $SING_SPOOL/ubuntu-61 OMP_NUM_THREADS=$NSLOTS myappli
```

- Supporte aussi MPI

Plan

Reproduction et diffusion avec Singularity

Etapes nécessaires pour la diffusion :

- Créer un dépôt Github public (hébergement Dockerfile)
- Créer un dépôt Singularity Hub (hébergement image)
- Lier les deux dépôts pour la construction de l'image

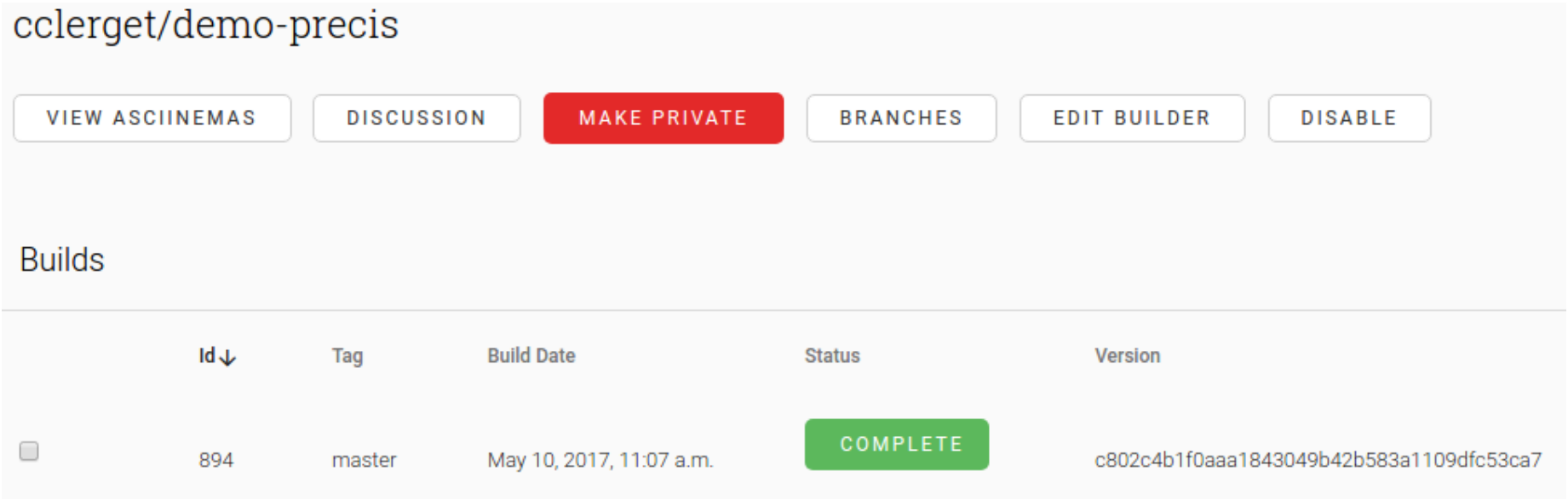
Le dépôt git contiendra un fichier Singularity :

```
BootStrap: docker
From: ubuntu:16.04

% post
apt-get update
apt-get install -y \
python-numpy \
cython \
python-matplotlib \
git

% runscript
cd /tmp
echo 'Get code and run simulations'
git clone \
https://github.com/ReScience-Archives/ReScience-Entry-Topalidou-Rougier-2015.git
cd ReScience-Entry-Topalidou-Rougier-2015/code
python setup.py build_ext --inplace
python single_trial.py
python 250-simulations.py
```

Reproduction et diffusion avec Singularity



Exécution depuis le Hub Singularity :

```
user@local:~$ singularity run shub://cclerget/demo-precis
cclerget/demo-precis:latest
...
Get code and run simulations
```

Comparaison avec Docker

	Singularity	**Docker**
Orienté HPC	Oui	Non
Accès root dans le container	Non	Oui
Configuration réseau additionnelle	Non	Oui
Accès aux systèmes de fichiers hôte	Oui	Oui (sécurité)
Exécution GPU	Oui *	Oui *
Exécution MPI	Oui *	Oui *
Support des gestionnaires de ressources	Oui	Incomplet
Installation facile (zéro conf)	Oui	Oui
Container modifiable lors de l'utilisation	Non	Oui
Contrôle administrateurs	Oui	Non
Noyaux Linux compatibles	>= 2.6	>= 3.8

*Configuration requise selon les ressources matérielles du système hôte

Quelques limites

Les dépendances avec le système hôte impactent la portabilité :

- Des problèmes peuvent survenir avec un ancien noyau Linux et une Libc trop récente
- Les pilotes noyaux peuvent requérir des versions de bibliothèques précises dans le container (CUDA)
- MPI :
  - Configuration et installation de la bibliothèque MPI avec les bibliothèques de la Fabric Infiniband correspondante au système hôte

Merci