Conteneurs pour le calcul

Martin Souchal (Centre de calcul Francois Arago)

En quoi les conteneurs peuvent nous aider dans le contexte scientifique ?

- Collaboration
  - Diffusion et partage simple de logiciels
  - Portabilité des logiciels
  - Technologie simple à mettre en œuvre
  - Rapidité de mise œuvre
- Répétabilité *
  - Assurance de répétabilité
  - Assurance de retrouver le même environnement
  - Assurance de retrouver les même résultats
- Autres
  - Solution identique du laptop au meso-centre
  - Interface graphique

* pas de reproductibilité avec les conteneurs mais de la répétabilité !

Cas d'usage pratique dans la science

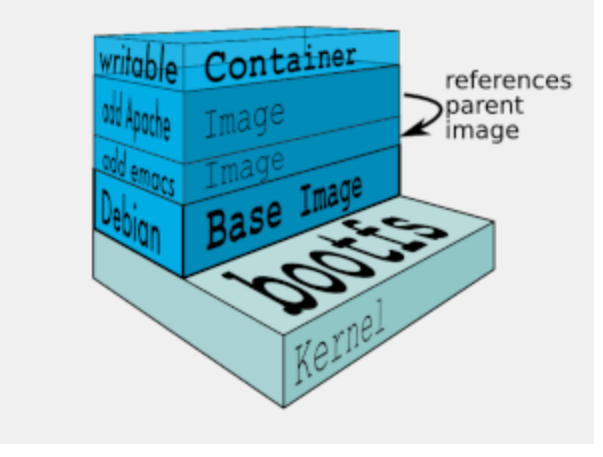
- Environnement de travail python prêt à l'emploi transportable sur n'importe quel cluster de calcul
- Les données restent dans les espaces de travail et n'ont pas besoin d'être intégrées au conteneur
- Tout ça de manière sécurisée sans avoir besoin d'être root

Répétabilité

- Le journal Rescience offre la possibilité de soumettre des articles scientifiques qui se doivent d'être reproductible.
- Figer l'environnement d'execution dans un conteneur et le joindre à une publication garantit la reproductibilité
- Voir le TP de Loic Gouarin : https://github.com/gouarin/container_precis2017

Docker et le calcul

- Docker est un micro service
- Virtualisation réseau (bridge, puis interfaces avec des IP privées)
- Image docker : superpositions de couches
- GPU, infiniband, openmpi pas intégrés
- Sécurité
  - daemon root
  - pas d'isolation : une appli root du conteneur qui s'échappe = un attaquant root sur la machine physique



Conteneurs orientés calcul

3 Conteneurs usermode compatibles openMPI

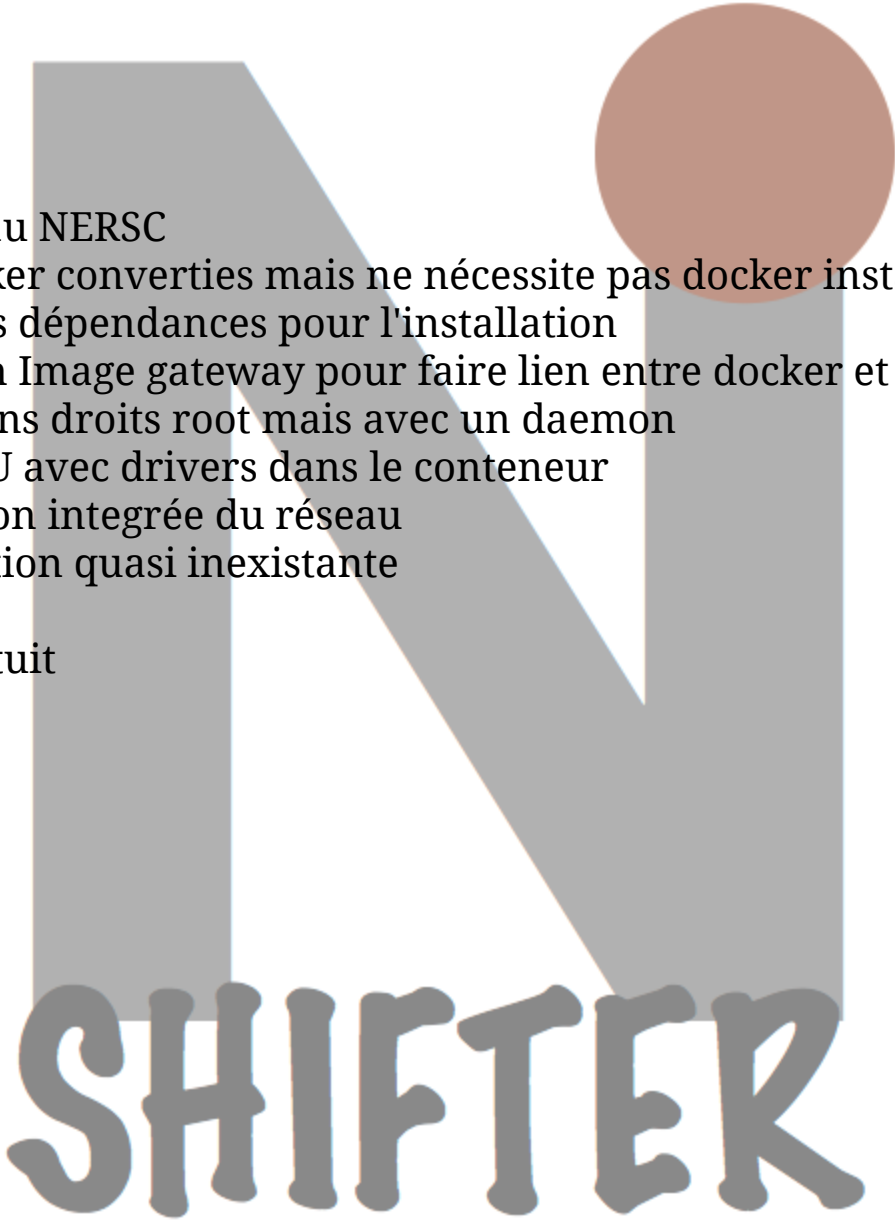
- Charlie-cloud
- Singularity
- Shifter

Charlie-cloud

- Développé à Los Alamos
- Univers docker
- nécessite de modifier la configuration kernel (user.max_user_namespaces, namespace.unpriv_enable=1)
- nécessite docker installé pour créer un conteneur
- conteneurs utilisables sans droits root
- un conteneur = un répertoire
- Support GPU avec drivers dans le conteneur
- Pas de gestion integrée du réseau
- Ecrit en C
- Documentation sommaire
- Libre et gratuit

Shifter

- Développé au NERSC
- Images docker converties mais ne nécessite pas docker installé
- Nombreuses dépendances pour l'installation
- Nécessite un Image gateway pour faire lien entre docker et shifter
- utilisable sans droits root mais avec un daemon
- Support GPU avec drivers dans le conteneur
- Pas de gestion integrée du réseau
- Documentation quasi inexistante
- Ecrit en C
- Libre et gratuit



Singularity

- Développé au Berkeley Lab // Sylabs par GREGORY KURTZER (CentOS, Warewulf)
- Aucune dépendance pour l'installation
- compatible toute distribution linux avec kernel récent
- droits root nécessaires pour créer un conteneur
- compatible avec les images docker
- compatible dockerhub
- Support GPU : possibilité de monter le GPU dans le conteneur avec les drivers de l'host (--nv)
- Pas de gestion integrée du réseau
- un conteneur = un fichier (meilleures perfs sur fs distribué) ou un répertoire
- Ecrit en C, réecriture en Go en cours
- Documentation pas toujours à jour
- Libre et gratuit / Version pro payante

Avantages de Singularity

- Facile à installer et à déployer sur un cluster de calcul
- Compatibilité avec Docker
- Utilisation de registres Public/Privé dédiés (http://www.singularity-hub.org/)
- Un conteneur est facile à transporter pour l'utilisateur (un fichier à copier)
- Singularity apps
- Compatible tous scheduler
- Intégration de tests et d'aide intégrée dans le conteneur
- Accès direct aux GPU de la machine

Création d'un conteneur Singularity

Avant de commencer

- Conteneur en mode "sandbox"
  - Pour le développement, tests
  - Possibilité de modifier le conteneur au fur et a mesure
- Conteneur en mode distribution (par défaut)
  - Il n'est plus modifiable une fois construit
  - Assurance que le destinataire du conteneur ne puisse pas altérer son contenu
  - Pour mettre à jour le conteneur, il faut l'incrémenter

Commandes de bases

- build: Create the container
- exec: Execute a command to your container
- inspect: See labels, run and test scripts, and environment variables
- pull: pull an image from Docker or Singularity Hub
- run: Run your image as an executable
- shell: Shell into your image
- apps : list apps
- help : help me !

```
singularity help example.img
Help me. I'm in the container.
```

Création d'un conteneur

- Création du conteneur depuis un fichier de "recette"

```
singularity build example.img example.def
```

- Import depuis *Docker*

```
singularity import example.img docker://sysmso/openmpi
```

- Execution d'un code dans un conteneur (compilation)

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

- Execution d'un programme *OpenMPI*

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

Bootstrap file

```
Bootstrap: docker
From: ubuntu:latest
%labels
    AUTHOR souchal@apc.in2p3.fr
    version 1.0
%post
    apt-get update
    apt-get -y install python
    exit 0
%help
    Help me. Im in the container.
%files
    pymultinest_demo_minimal.py /pymultinest_demo_minimal.py
%environment
    LD_LIBRARY_PATH=/usr/local/lib/
    export LD_LIBRARY_PATH
%runscript
    python /pymultinest_demo_minimal.py
```

Singularity apps

Nouveauté de la version 2.4

- un conteneur, plusieurs applications avec des environnements différents
- pratique pour les workflow partageant les mêmes dépendances

```
%apprun foo
    exec echo "RUNNING FOO"
%applabels foo
   BESTAPP=FOO
   export BESTAPP
%appinstall foo
   touch foo.exec
%appenv foo
    SOFTWARE=foo
    export SOFTWARE
%apphelp foo
    This is the help for foo.
%appfiles foo
   avocados.txt
```

Questions