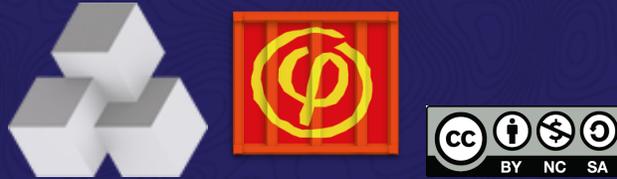


Conteneurs pour le calcul



Martin Souchal 2021 (APC - FACe - ComputeOps)

ComputeOps

- Le projet **ComputeOps** a pour objectif d'étudier les avantages des conteneurs pour les applications de type HPC. Lancé en 2018, le projet est financé par le master project DecaLog de l'IN2P3 dans le cadre du programme de R&D transverse.
- Participants IN2P3 : APC, LAL, LPNHE, LLR, IPHC
- Partenaires extérieurs :
 - Groupe de travail Aristote sur la virtualisation légère.
 - Ecole Centrale de Nantes
 - IAS
 - INRAE
 - Intel
 - Sylabs



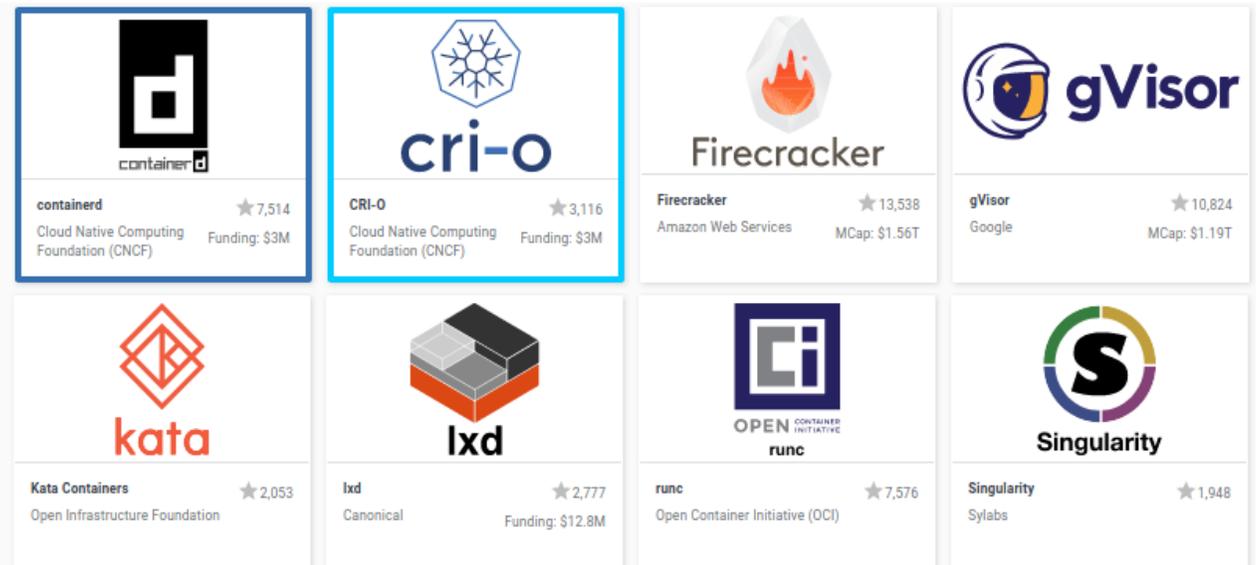
Plan

- Les conteneurs : principes généraux
- Les conteneurs et le HPC
- Focus sur Singularity
- Orchestration et Job Scheduling
- Conclusion

Les conteneurs

Un peu d'histoire...

- Technologie apparue dans Solaris via les jails
- Basé sur les kernel namespaces (kernel 2.4.19) et les cgroups (kernel 2.6.24)
- LXC/LXD en 2008, Docker en 2013
- Kubernetes en 2014 chez Google
- 2015 : Docker top 15 GitHub, création de l'**Open Container Initiative** (OCI)



Les conteneurs

Principe de base

- A l'arrêt, un conteneur est un fichier (ou un ensemble de fichiers) qui est enregistré sur un disque.
- Au démarrage du conteneur, le moteur décompresse les fichiers et les méta-données nécessaires, puis les transmet au noyau Linux, comme un processus normal.
- Un appel API au noyau déclenche une isolation supplémentaire (si nécessaire) et monte une copie des fichiers de l'image du conteneur.
- Une fois lancés, les conteneurs sont un processus Linux comme un autre.
- Un conteneur est immuable (modifier un conteneur = créer un nouveau)

Open Container initiative (OCI)

Le processus de démarrage des conteneurs, ainsi que le format de l'image sur le disque, sont définis et régis par des normes.

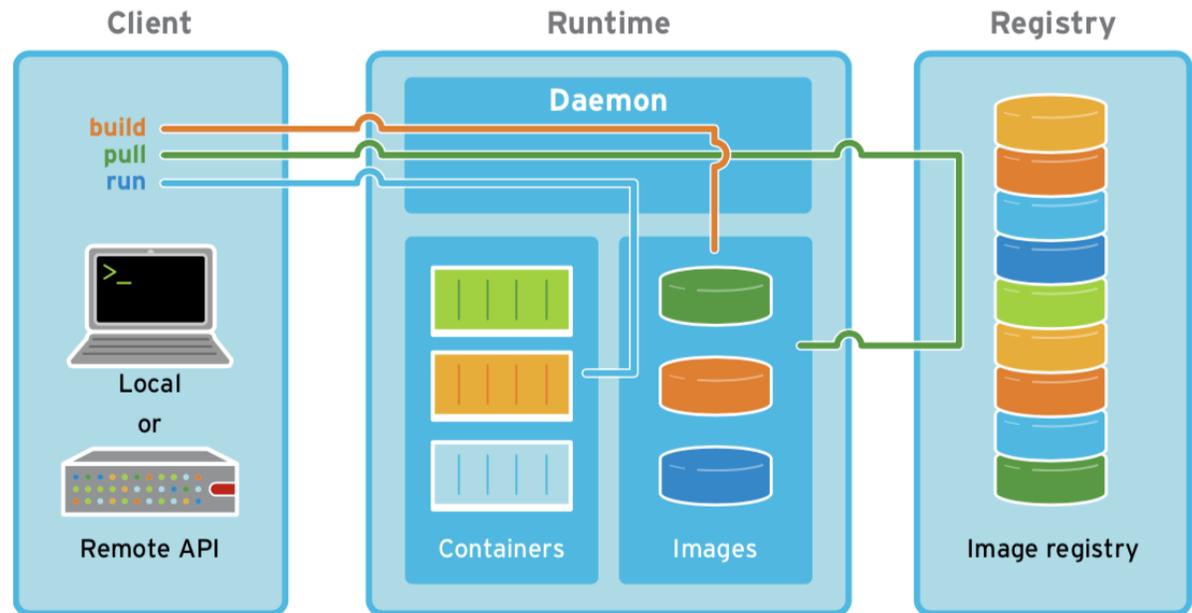
- Gouvernance ouverte chapeautée par la Linux Foundation
- Standard pour les runtimes et les images de conteneurs (pas les manifestes !)
- Ligne de commande unifiée

```
docker run example.com/org/app:v1.0.0
alias docker=podman
alias docker=singularity
...
```



Vocabulaire

- Runtime : moteur chargé d'exécuter le conteneur
- Image : fichier (ou un ensemble de fichiers) enregistré sur un disque
- Manifeste : Fichier texte, "recette" du conteneur
- Registre : un serveur abritant des images de conteneurs
- Repository : ensemble de registres



Manifeste

- Plusieurs formats de manifestes : Dockerfile, Singularity...
- Un manifeste est transformé en image via un mécanisme de build (intégré ou non)
- Un conteneur peut être construit "from scratch" ou à partir d'une autre image
- La base d'une image est un OS Linux (Alpine Linux, Busybox, Ubuntu, CentOS...)
- Pas de standards pour les manifestes

```
FROM python:3
RUN mkdir /code
RUN apt-get update -q && apt-get install graphviz libgraphviz-dev libldap2-dev libsasl2-dev -yqq
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
```

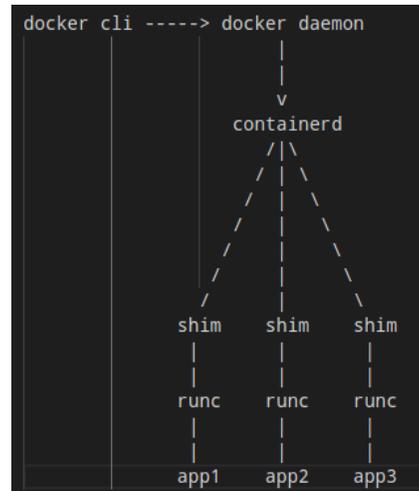
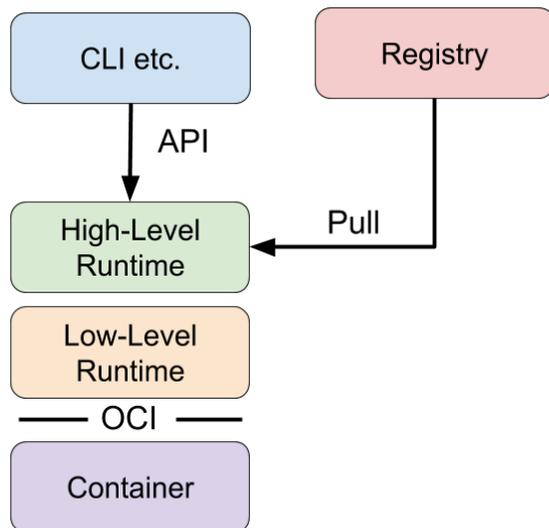
```
docker build -f Dockerfile -t Container .
```

```
singularity build Container.sif Singularity.def
```

```
buildah bud -f Dockerfile -t Container .
```

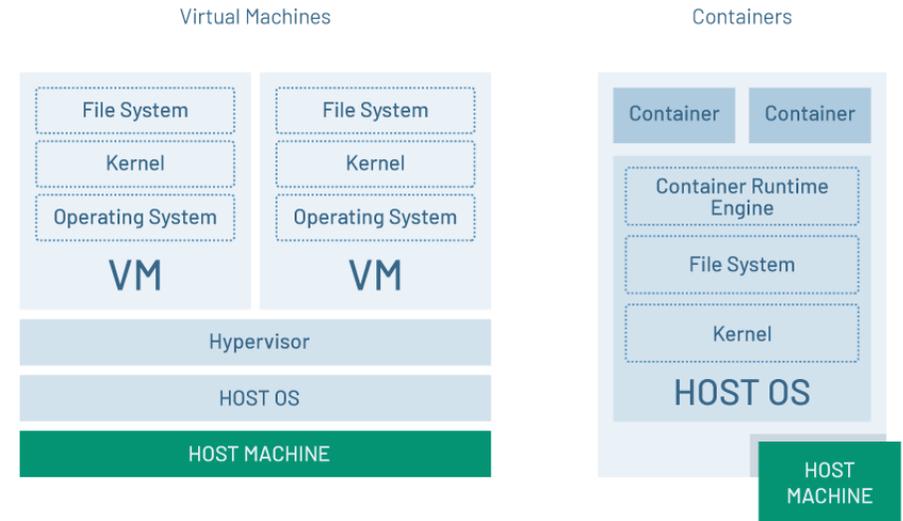
Runtimes

- Runtimes bas niveau : runc (docker), LXC/LXD, crun, conmon (RedHat), nvidia-docker (Nvidia)
- Runtimes haut niveau : Docker, Singularity, Podman (RedHat), containerd (Docker), CR-IO (Kubernetes), enroot (Nvidia), SmartOs (Samsung)
- MicroVms : KataContainer (INTEL), Firecracker (AWS), gVisor (google)
- Image builder : buildah, img, orca-build...



Isolation et performances

- runtimes de bas niveau en user mode (namespaces) :
 - Temps d'instanciation minimal
 - Isolation faible (proche du matériel)
 - Performances optimales
- runtimes sécurisées (Kata, gVisor) :
 - Isolation forte (équivalent VM)
 - Temps d'instanciation modéré (plus rapide que VMs)
 - Performances dégradées, dépendantes hardware
- Performances liées aux formats d'images
 - Format en couches optimisés pour téléchargement
 - Format fichier optimisé pour le stockage



Isolation et Sécurité

focus sur seccomp

- Seccomp est utilisé par Docker et Singularity
- Permet de filtrer les syscalls depuis les runtimes (liste noire/blanche)

```
~ $ docker info
...
Security Options:
  apparmor
  seccomp
  Profile: default # default seccomp profile applied by default
...
```

- En désactivant complètement seccomp, tous les syscalls sont autorisés :

```
# Lancer un conteneur sans seccomp
~ $ docker run --rm -it --security-opt seccomp=unconfined alpine sh
/ # reboot # oops
# Lancer docker avec le profil seccomp par défaut
~ $ docker container run --rm -it alpine sh
/ # reboot # ouf
```

Isolation et Sécurité

focus sur seccomp

- Lancement d'un conteneur avec un profil seccomp personnalisé

```
# Run avec un profil interdisant le chmod
~ $ docker container run --rm -it --security-opt seccomp=no-chmod.json alpine sh
/ # whoami
root
/ # chmod 777 -R /etc
chmod: /etc: Operation not permitted
```

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "syscalls": [
    {
      "names": [
        "accept",
        "chown",
        "kill",
        "mmap",
        ...
      ],
      "action": "SCMP_ACT_ALLOW",
      "args": [],
      "comment": "",
      "includes": {},
      "excludes": {}
    }
  ]
}
```

Sécurité

- Docker : daemon root, possibilité d'être root dans un conteneur
- Possibilité d'utiliser AppArmor ou seccomp coté admin pour "verrouiller" l'infra
- Conteneurs dans le namespace "user" :
 - même utilisateur hors du conteneur et dans le conteneur
 - les problématiques d'élévation de privilèges sont reportées sur le noyau linux
 - Attention au SUID...
- Isolation plus ou moins forte (de userNS a VM)
- Vecteur de failles de sécurité logicielle important (cf Alpine Linux)
- Vecteur d'attaque pour rootkit (très facile de cacher du code malicieux dans un conteneur)



SetUID : un programme lancé avec le droit "suid" sera exécuté avec les droits du propriétaire du programme et non les droits de l'utilisateur qui l'a lancé

Registres publics

- Images vérifiées
 - [NVIDIA NGC](#) - images orientées GPU
 - [Docker Hub](#) - images officielles
- Libres
 - [Docker Hub](#)
 - [Singularity Hub](#)
 - [Sylabs cloud](#) - images signées
 - [Quay.io](#) (scan de sécurité)



Registres privés

- Gitlab (Registre Docker)
- Harbor (Registre Docker)
- Singularity Hub (Registre Singularity)
- Azure, AWS, etc... (OCI)
- Pour stocker des images OCI dans des registres docker : **ORAS**

The logo for ORAS (Open Container Registry and Artifact Registry) is displayed in a colorful, bubbly font. The letters are 'O' (red), 'R' (teal), 'A' (yellow), and 'S' (light green), all with a blue outline and a slight 3D effect.

Bonnes pratiques

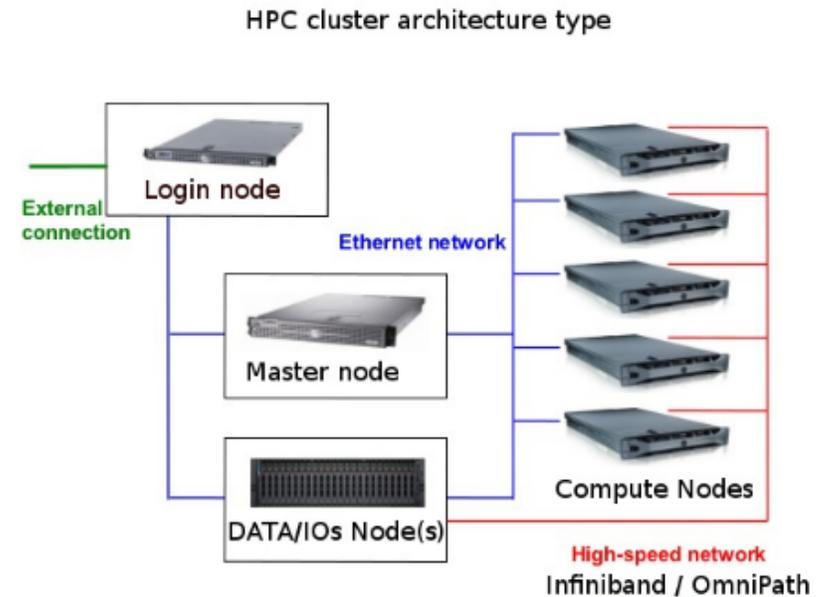
- Partager le manifeste du conteneur avec le conteneur
- Utiliser l'intégration continue pour la génération de conteneurs
- Spécifier toutes les versions exactes des OS et des logiciels dans les manifestes (conda env export, pip freeze, etc....)
- Taguer les images des conteneurs
- Signer les conteneurs diffusés à l'aide de clés
- Préférer les catalogues d'images vérifiées, ou au moins ceux qui scannent les failles de sécurité

Les conteneurs et le HPC

Architecture HPC

High Performance Computing

- Cluster/grappe de calcul
- Matériel spécifique :
 - Utilisation d'accélérateurs (GPUs, TPUs...)
 - Réseau Infiniband/Omnipath
- Calcul parallèle avec mémoire partagée (MPI)
- Spark, hadoop...
- Orchestration avec un Job Scheduler (Univa, SGE, Slurm, etc...)



Problématiques software HPC

- Besoin de performances et d'optimisations au plus proche du matériel.
- Des jobs qui tournent des semaines, voir des mois ; la maintenance doit être planifiée longtemps en avance,
- Applications scientifiques parfois complexes à compiler, avec de nombreuses dépendances,
- Des noyaux souvent plus vieux que d'ordinaire (nombreux soucis avec la glibc)...,
- Environnement cluster = multi-utilisateurs.
- Support : "J'ai besoin du logiciel X..."

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 *
- Simplicité
 - Solution identique du laptop au meso-centre
 - Interface graphique

* C'est faux !

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

Workflow HPC

- Build de l'image du conteneur sur le poste de travail, ou en CI
- Test en local
- Mise a disposition dans un catalogue
- Execution dans un environnement HPC

Répétabilité

- Le journal **Rescience** offre la possibilité de soumettre des articles scientifiques qui se doivent d'être reproductible.
- Figurer l'environnement d'exécution dans un conteneur et le joindre à une publication garantit la répétabilité *, parfois la reproductibilité
- La reproductibilité en informatique nécessite de partager un environnement logiciel *exactement identique*, mais aussi un hardware *exactement identique*...
- Exécuter un environnement logiciel exactement identique n'est pas garanti par l'utilisation de conteneurs.
 - bonnes pratiques à respecter
 - transparence totale
 - rigueur



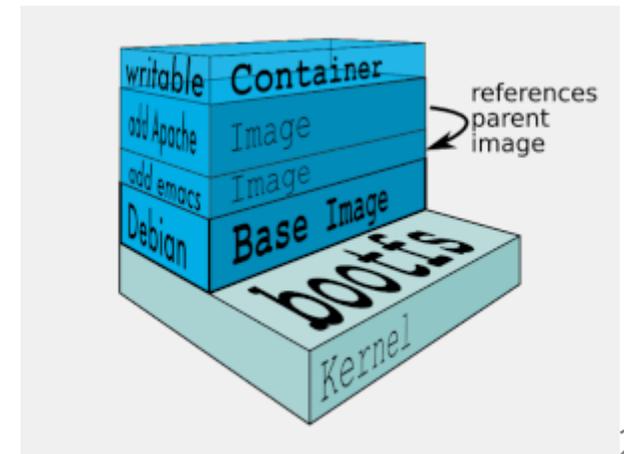
* C'est toujours faux !

Docker et le calcul

Un écosystème pas très adapté...

- Docker est adapté pour les micro services (un conteneur, un service), pas très facile de gérer des chaînes de calcul
- Network namespace et compatibilité matériel réseau pour cluster (Intel OmniPath, Infiniband...)
- Aucun support MPI
- Image docker : superpositions de couches, pas très portable dans un cluster
- Sécurité :
 - daemon root à installer sur tous les noeuds de calculs...
 - des utilisateurs root dans votre cluster...

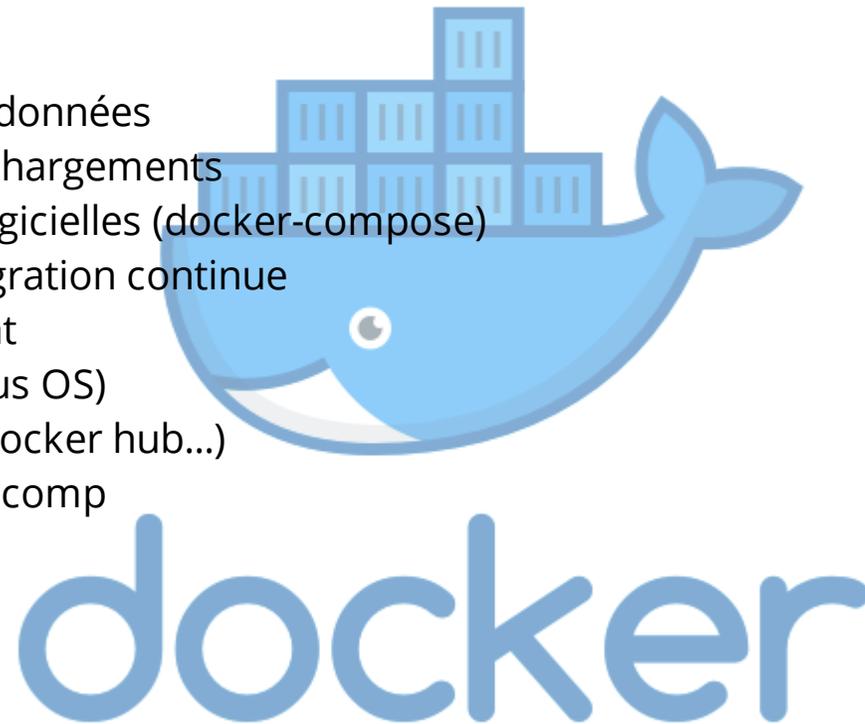
docker



Docker et le calcul

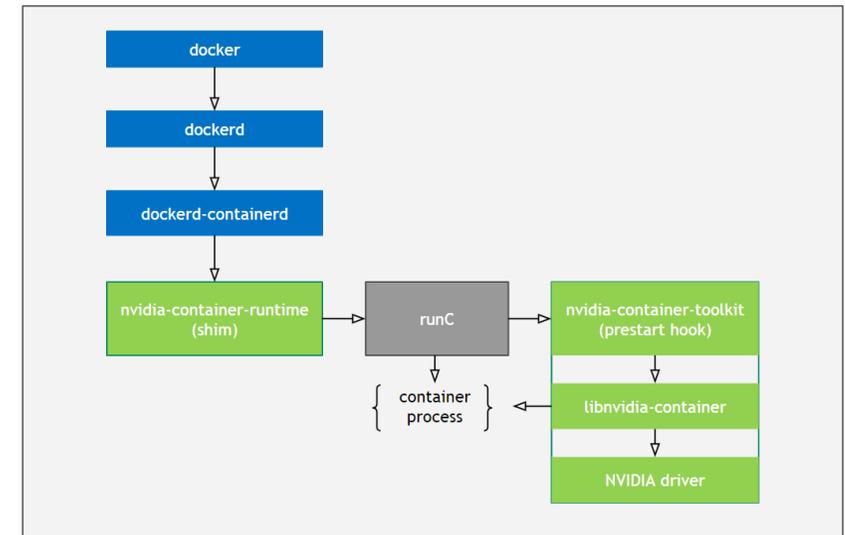
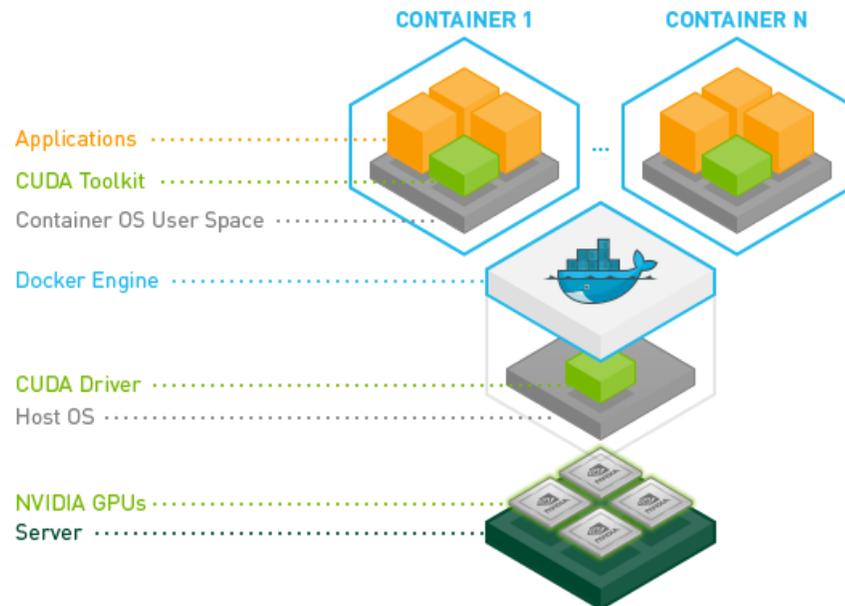
... mais néanmoins intéressant

- pour la gestion de bases de données
- pour l'optimisation des téléchargements
- pour la création de stacks logicielles (docker-compose)
- utilisé massivement en Intégration continue
- pour faire du développement
- très répandu et portable (tous OS)
- pour le catalogue existant (docker hub...)
- sécurisation possible via seccomp
- pour nvidia-docker...



Nvidia docker

- Permet d'utiliser des GPUs Nvidia avec Docker
- Utilisable comme runtime et comme toolkit
- Prérequis : kernel version > 3.10, Docker >= 19.03, NVIDIA GPU > Fermi, NVIDIA drivers ~ = 361.93



Conteneurs orientés calcul

4 Conteneurs compatibles HPC

| Nom | Dernière Release | Support HW | OCI | ARM | OS | Compatible Scheduler | Contrôle Admin | Privilèges |
|---------------|------------------|----------------|-----|-----|---------------------|----------------------|----------------|-------------|
| Docker | 04/01/2021 | GPU Nvidia | Oui | Oui | MacOs - W\$ - Linux | Non | Oui | Root daemon |
| Charlie-cloud | 18/12/2020 | MPI - GPU - IB | Non | Non | Linux | Oui | Non | UserNS |
| Singularity | 12/01/2021 | MPI - GPU - IB | Oui | Oui | Linux - MacOS | Oui | Oui | SUID/UserNS |
| Shifter | 01/04/2018 | MPI - GPU - IB | Non | Non | Linux | Slurm | Oui | SUID |
| Nvidia enroot | 02/12/2020 | GPU - IB | Oui | Non | Linux | Oui | Non | UserNS |

Charlie-cloud

- Développé à Los Alamos National Laboratory (LANL)
- Univers docker
- nécessite de modifier la configuration kernel (`user.max_user_namespaces`, `namespace.unpriv_enable=1`)
- Pas de package, make install uniquement
- Runtime uniquement (pas de construction d'image)
- conteneurs utilisables sans droits root
- un conteneur = un répertoire
- Support GPU avec drivers dans le conteneur
- Pas de gestion intégrée du réseau
- Écrit en C, peu de ligne de code (autour de 1000)
- Libre et gratuit



Shifter

- Développé au NERSC
- Pas de package
- 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 intégrée du réseau
- Peu de documentation
- Écrit en C
- Libre et gratuit



SHIFTER

Nvidia enroot

Enroot peut être considéré comme un chroot non privilégié amélioré. Il utilise les mêmes technologies sous-jacentes que les conteneurs mais supprime une grande partie de l'isolement qu'ils procurent tout en préservant la séparation du système de fichiers.

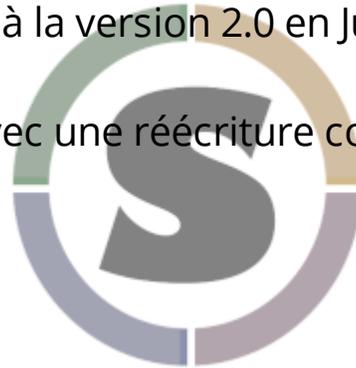
- Pratique le principe KISS
- Standalone (pas de daemon)
- Tourne en namespace user (pas de binaire setuid, cgroups, configuration par utilisateur...)
- Facile à utiliser (simple image format, scriptable, root remapping...)
- Pas à peu d'isolation
- Configuration administrateur et utilisateur
- Import rapide d'images Docker (vitesse de 3x à 5x sur des grosses images)
- Images SquashFS
- Support GPU Nvidia avec libnvidia-container
- Packages DEB et RPM, make install...

```
# Import and start an Ubuntu image from DockerHub
$ enroot import docker://ubuntu
$ enroot create ubuntu.sqsh
$ enroot start ubuntu
```

Singularity

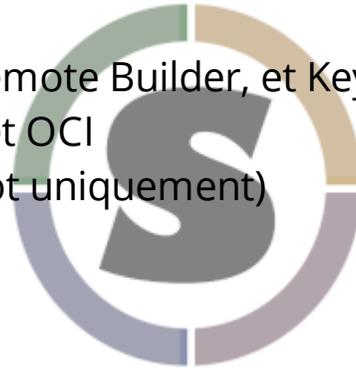
Historique

- Inventé par Greg Kurtzer (Rocky Linux, CentOS, Warewulf) au Berkeley Lab pour répondre aux problèmes spécifiques au HPC
- Développement démarré en Octobre 2015
- Première release en Avril 2016
- Les retours de la communauté ont menés à la version 2.0 en Juin 2016
- Sylabs.io en Janvier 2018
- Sortie de la version 3.0 en Octobre 2018 avec une réécriture complète, de nouvelles fonctionnalités et une ambition hors HPC
- Support officiel NVIDIA via GPU Cloud



Fonctionnalités

- Nécessite Golang > 1.13 pour la compilation
- Package RPM uniquement
- Droits root nécessaires pour créer un conteneur localement
- Fichier image SIF : unique, exécutable, chiffré et signé
- Support natif de MPI, Xorg, GPU
- Infrastructure Cloud (Container Library, Remote Builder, et Keystore)
- Possibilité d'exécuter des images Docker et OCI
- Support/isolation réseau (plugins CNI / root uniquement)
- Libre et gratuit / Version pro payante



Sécurité

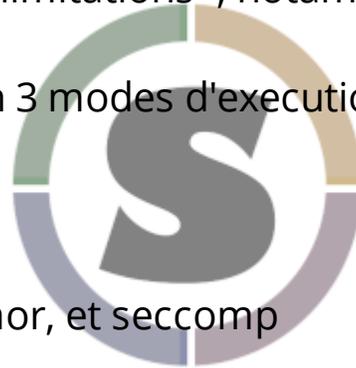
- Support des Cgroups pour la gestion des ressources
- L'utilisateur root peut définir un set de capabilities à l'exécution
- L'administrateur peut gérer les capabilities des utilisateurs (avec précaution)
- Les utilisateurs peuvent passer un contexte d'exécution SELinux, ou un profil AppArmor
- Les utilisateurs peuvent passer leurs propres filtres d'appel système via seccomp
- Support d'exécution avec des UID/GID différents pour l'utilisateur root (par défaut c'est le même)
- Conteneurs chiffrés, signés (SIF)



Singularity coté admin

singularity.conf

- Définir les ressources, les restrictions de sécurité, les montages, les options réseau...
- Activer ou non le mode SUID entraîne des limitations*, notamment liées aux images SIF et a fakeroot (user NS)
- Gérer les capabilities des utilisateurs selon 3 modes d'exécution :
 - full
 - file
 - no
- Gestion de la sécurité via SELinux, AppArmor, et seccomp



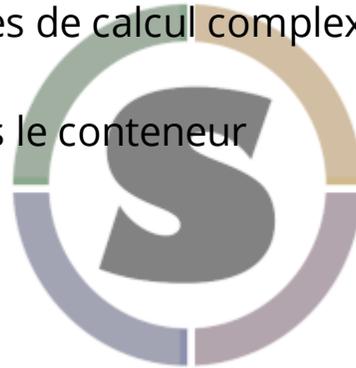
```
$ singularity exec --drop-caps CAP_NET_RAW library://sylabs/tests/ubuntu_ping:v1.0 ping -c 1 8.8.8.8
ping: socket: Operation not permitted
```

```
$ sudo whoami
root
$ sudo singularity exec --security uid:1000 my_container.sif whoami
david
```

*Plus d'infos dans la documentation [officielle](#)

Avantages de Singularity pour le HPC

- Facile à installer, configurer et à déployer sur un cluster de calcul
- Compatibilité avec le catalogue d'image Docker
- Utilisation de registres Public/Privé dédiés (<http://www.singularity-hub.org/>), compatibilité gitlab
- Un conteneur est facile à transporter pour l'utilisateur (un fichier SIF à copier)
- Singularity apps pour la gestion des chaines de calcul complexes
- Compatible tous job scheduler
- Intégration de tests et d'aide intégrée dans le conteneur
- Accès direct aux GPU de la machine
- Intégration dans Kubernetes
- Conteneurs chiffrés et signés
- Conteneurs en lecture seule
- Sécurité a la carte

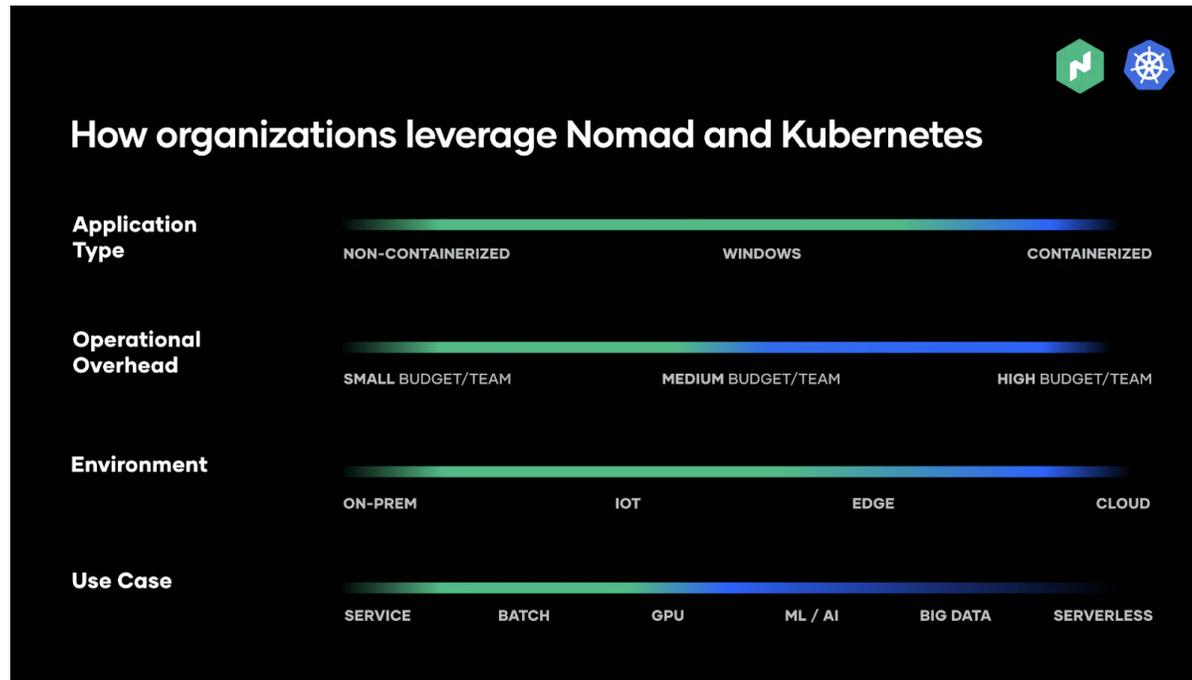


Orchestration et scheduling

Orchestrateur de conteneur

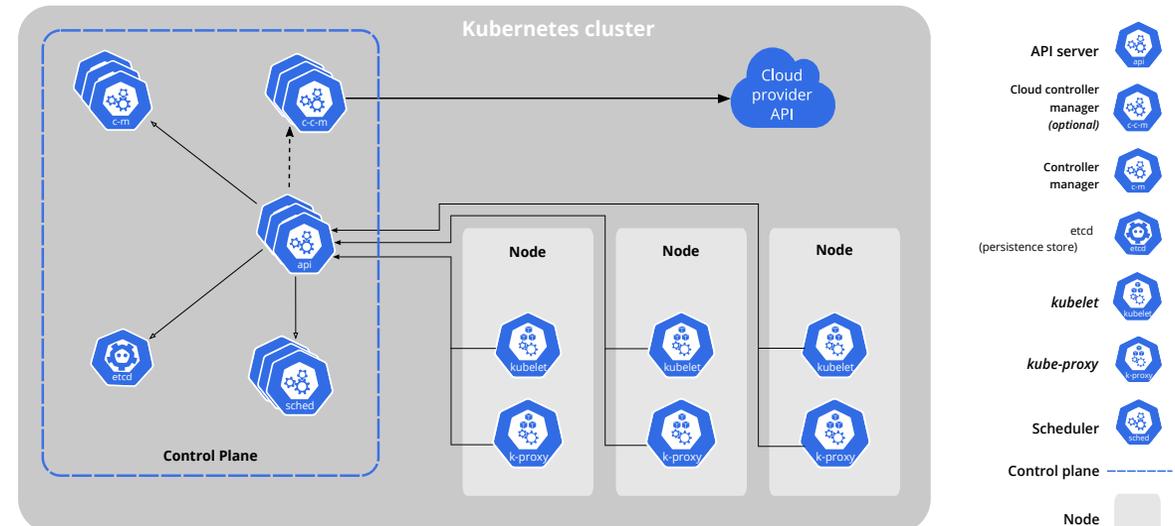
Un Orchestrateur est un système permettant d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées

- Approche microservice : les conteneurs qui composent une application sont regroupés dans des unités logiques (pods) pour en faciliter la gestion et la découverte
- Deux acteurs principaux : **Kubernetes** (The Linux Foundation) et **Nomad** (Hashicorp)



Kubernetes

- Kubernetes est orienté microservice et web
- Gestion d'accélérateurs avec framework device plugins
- Basé sur Docker, ouvert aux runtime OCI (containerd, CRI-O, Singularity)
- Pas de gestion de batch dans Kubernetes, ce qui s'en rapproche le plus est la notion de Job
 - Un ensemble de pods qui exécutent une tâche et s'arrêtent.
 - Le job lance un nouveau pod si un pod échoue ou est supprimé.
 - Possibilité de lancer plusieurs pods en parallèle (mais pas avec MPI).
- Singularity-CRI permet le support des GPU NVIDIA en **device plugin**



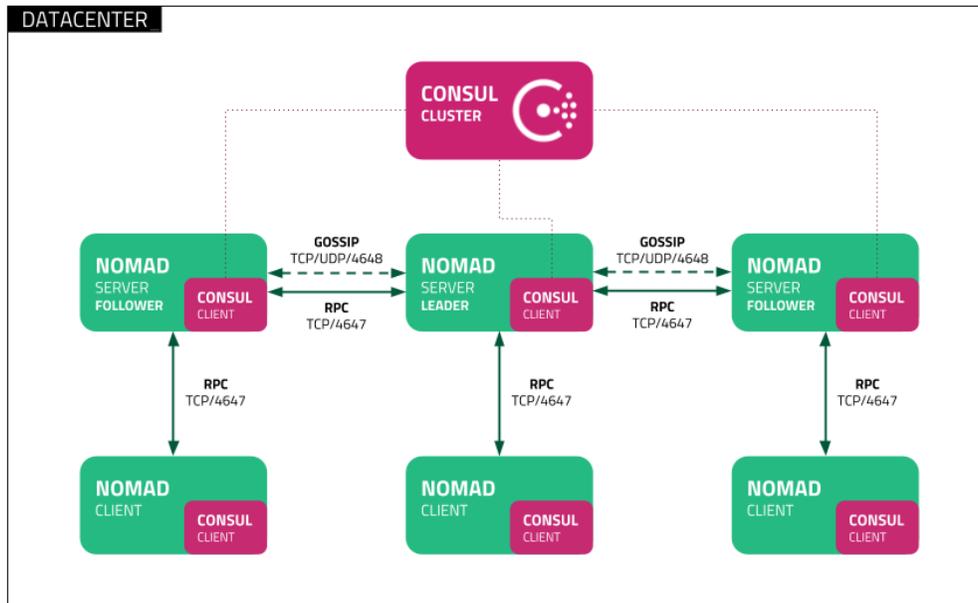
Nomad

- Infrastructure as cloud déclaratif
- Version 1.0.2
- Kubernetes vise à fournir toutes les fonctionnalités nécessaires à l'exécution d'applications basées sur Docker, notamment la gestion des clusters, le scheduling, la découverte de services, la surveillance, la gestion des secrets, etc...
- Nomad ne fait que de la gestion de cluster et du scheduling



HashiCorp

Nomad



Nomad

- S'intègre avec Consul (découverte et maillage de services, réseau), Terraform (déploiement) et Vault (gestion de secrets)
- Nomad supporte la fédération native multi-region et multi-datacenter
- Nomad intègre la gestion des GPU/TPU/FPGA, support Spark
- Nomad permet d'orchestrer des VMs ou des applications batchs en plus des conteneurs
- Scalabilité : Kubernetes est limité a 5000 nœuds et a 100 pods par nœuds. Instances de Nomad de plus de 10 000 nœuds en production.
- Nomad est entièrement et uniquement développé par des employés d'HashiCorp, en open source mais sans support communautaire
- Nomad est compatible Linux, MacOS et Windows, Kubernetes n'est supporté que sur Linux
- Nomad distribué sous la forme d'un binaire de 75Mb (pour client et serveur), Kubernetes demande le déploiement de nombreux conteneurs



HashiCorp

Nomad

Désavantages de Nomad

- Définition des ressources manuelle
- Auto scaling pas encore disponible
- Développement très rapide, avec une certaine tendance à la dépréciation sauvage et sans préavis
- Un certains nombre de fonctionnalités payantes



HashiCorp

Nomad

Job scheduler

- Notion de job avec heure de début - fin, ressources à utiliser
- Mécanismes de :
 - files d'attente/de mises en attente ou pause (checkpointing)
 - priorités / Fair sharing
 - droits/accès et partages des ressources physiques
- Intégration dans l'environnement cluster existant
- Compatible tous runtimes en userNS



Conclusion

- Les conteneurs ne garantissent pas la reproductibilité
- ... ni la répétabilité *a priori*
- ils ont d'autres qualités : la portabilité, l'intégration continue, l'autonomie de l'utilisateur, le partage de logiciels...
- Peu d'isolation entraîne une grande dépendance à l'environnement hôte...
- ...et posent des problèmes de sécurité !

Questions

Retrouvez nous sur

- Rocket Chat : [#computeops](#)
- [Citadel CNRS](#)

