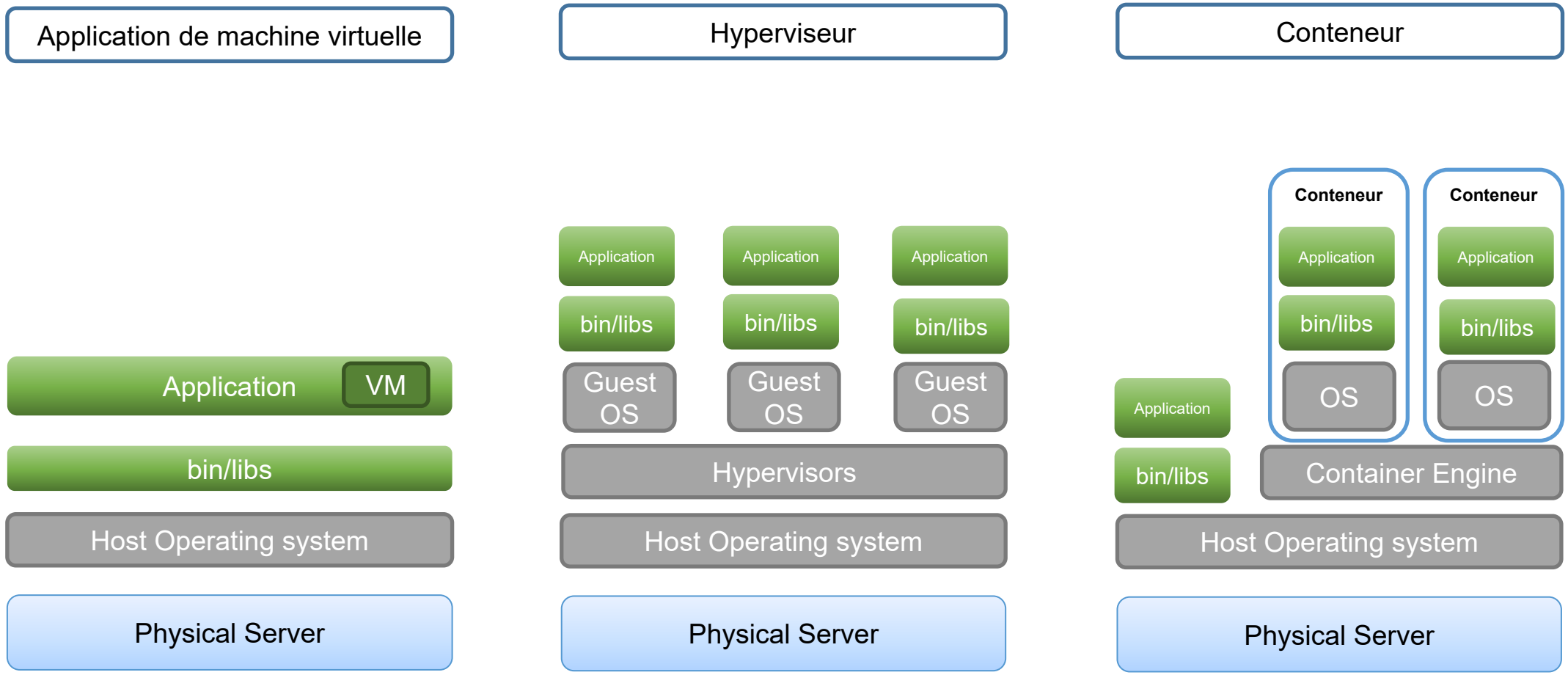
Singularity

Virtualisation



Intérêt des Conteneurs

- Les conteneurs permettent d'assembler et d'isoler des applications avec leur environnement d'exécution complet contenant tous les fichiers nécessaires à leur exécution.

- Les applications conteneurisées sont plus faciles à déplacer d'un environnement à un autre (développement, test, production, etc.), tout en conservant l'intégralité de leurs fonctions.

Conteneurs les plus connues : Dockers & Singularity

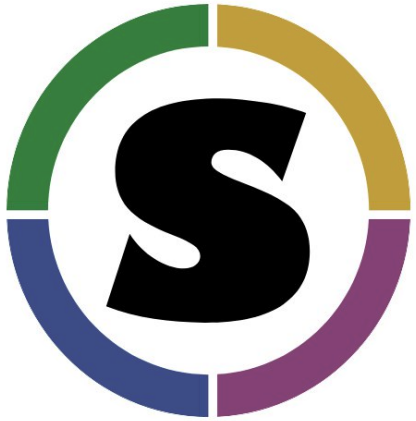
Dockers

- Démon root
- Couche d'abstraction
- Images Docker

SINGULARITY

Orienté calcul scientifique

- Pas de démon
- Calcul sur GPU
- Images Docker &Singularity



Singularity

- Initialement développé en python par Gregory Kurtzer au Lawrence Berkeley National Laboratory pour leur HPC

- Entièrement réécrit en GO fin 2018 par Sylabs (mais toujours avec G. Kurtzer) qui le maintient aujourd'hui avec une version pro.

- Mais engagement à conserver la version open-Source.

Documentation : https://www.sylabs.io/

Principe de fonctionnement général de singularity

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).

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
- 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.

Principe de fonctionnement général de singularity

L'utilisateur qui lance le conteneur est le même utilisateur dans le conteneur.

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

- $HOME
- /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).

1/ Installation des dépendances

```
$ sudo apt-get update && sudo apt-get install -y \
build-essential \
libssl-dev \
uuid-dev \
libgpgme11-dev \
squashfs-tools
```

Installation de Singularity : Installation de GO

Documentation : https://golang.org/dl/

le wget ne fonctionne pas, il faut télécharger au préalable sur son ordi perso (https://dl.google.com/go/go1.11.5.linux-amd64.tar.gz).

```
$ scp /home/mdarocha/Téléchargements/go1.11.5.linux-amd64.tar.gz mdarocha@147.99.84.183:/home/tarball/.
mdarocha@147.99.84.183's password:
go1.11.5.linux-amd64.tar.gz

$ cd /home/tarball

##Télécharger l'archive dans /usr/local

$ sudo tar -C /usr/local -xzf go1.11.5.linux-amd64.tar.gz
```

Version de go va dépendre de la version de singularity que l'on installe

Modification /etc/profil:

```
$ sudo vim /etc/profile
##Add
export GOPATH=/home/tools/go
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin

##
$ source /etc/profile
```

Création d'un programme en GO / compilation / lancement

```
$ cd /home/tools
$ mkdir -p go/src/hello
$ cd go/src/hello
$ geany hello.go
```

Ajout du texte suivant :

```
package main

import "fmt"

func main() {
        fmt.Printf("hello, world\n")
}
```

```
$ go build

$ ./hello
hello, world
```

Installation de Singularity

Téléchargement de Singularity :

Téléchargement des dépendances de GO necessaire pour Singularity :

```
$ go get -u -v github.com/golang/dep/cmd/dep
```

Compilation de Singularity :

Installation de Singularity : Test de fonctionnement

Vérification que le help fonctionne:

```
$ singularity help

###
Linux container platform optimized for High Performance Computing (HPC) and
Enterprise Performance Computing (EPC)
....
For additional help or support, please visit https://www.sylabs.io/docs/

Usage:
  singularity [global options...]
###
```

En recupérant une image sur un repository:

```
$ singularity shell library://sylabsed/examples/lolcow

###Vous ouvrez un shell dans une

Singularity lolcow_latest.sif:/home/tools/go/src/github.com/sylabs/singularity>
Singularity lolcow_latest.sif:/home/tools/go/src/github.com/sylabs/singularity> cowsay moo
 _____
< moo >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Si la vache dit moo, l'installation est réussit ;-)

Récupération d'un conteneur existant dans un shub (singularity hub)

On peut récuperer des images existantes sur des hubs existants.

```
# Récupérer une image
$ singularity pull shub://vsoch/hello-world
$ singularity run hello-world_latest.sif
```

On peut récuperer des fichiers de définitions d'images existantes et les recompiler

```
# Récupèrer un fichier de définition de l'image - recompilation avec son propre système
$ singularity build hello-world.sif shub://vsoch/hello-world
$ singularity run hello-world.sif
```

Pour utiliser un conteneur, il suffit d'avoir singularity installé!

Utilisation d'un conteneur : Shell interactive

- Le shell interactive correspond à l'ouverture d'une console dans le conteneur : permet par exemple de tester les commandes installées dans le conteneur.

```
$ singularity shell hello-world.sif
Singularity hello-world.sif:/home/biotools/singularity_img> cat /etc/issue
Ubuntu 14.04.5 LTS \n \l

Singularity hello-world.sif:/home/biotools/singularity_img> exit
exit
```

Exécution d'une commande dans le conteneur : exec

La commande **exec** fait exécuter une commande dans le conteneur

Exemple 1 : lire le fichier /etc/issues pour connaitre la version

```
$ cat /etc/issue
Ubuntu 16.04.4 LTS \n \l          #sur la machine

$ singularity exec hello-world.sif cat /etc/issue          # dans le conteneur
Ubuntu 14.04.5 LTS \n \l
```

Exemple 2 : Lister les dossiers et les fichiers à la racine

```
$ ls /                    #sur la machine
bighub  boot   core  data1  dev  home        initrd.img.old  lib64       media  opt   root  sbin  srv      sys  usr
bin     cdrom  data  data2  etc  initrd.img  lib

$ singularity exec hello-world.sif ls /          # dans le conteneur
bin  boot  dev  environment  etc  home  lib  lib64  media  mnt  opt  proc  rawr.sh  root  run  sbin  singularity  srv
```

Métadonnées d'un conteneur : inspect

Le créateur du conteneur peut associés un ensemble de métadonnée décrivant le conteneur dans la partie label du fichier de définition. La commande **inspect** permet de les afficher.

```
$ singularity inspect hello-world.sif

{
        "MAINTAINER": "vanessasaur",
        "WHATAMI": "dinosaur",
        "org.label-schema.build-date": "Tuesday_14_May_2019_9:19:29_CEST",
        "org.label-schema.build-size": "333MB",
        "org.label-schema.schema-version": "1.0",
        "org.label-schema.usage.singularity.deffile": "Singularity",
        "org.label-schema.usage.singularity.deffile.bootstrap": "shub",
        "org.label-schema.usage.singularity.deffile.from": "vsoch/hello-world",
        "org.label-schema.usage.singularity.version": "3.1.1-663.gceb00f6"
}
```

Droit des utilisateurs dans un conteneur singularity

Dans un conteneur singularity , l'utilisateur est la personne qui l'utilise. Il conserve les mêmes droits que celle qui a sur la machine .

Dans le conteneur :

```
$ singularity shell hello-world.sif
Singularity hello-world.sif:/home/biotools/singularity_img> whoami
mdarocha
Singularity hello-world.sif:/home/biotools/singularity_img> id
uid=1172(mdarocha) gid=1100(bioinfo) groups=1100(bioinfo)
Singularity hello-world.sif:/home/biotools/singularity_img> exit
exit
```

Sur la machine :

```
$ whoami
mdarocha
$ id
uid=1172(mdarocha) gid=1100(bioinfo) groupes=1100(bioinfo)
```

Pas besoin d'être root pour utiliser un conteneur singularity

Accès aux fichiers présents sur la machine physique : Montages par défaut

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

- $HOME
- /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).

L'image (conteneur) est par défaut en read-only.
Seuls les montages seront en read-write et leur accès dépend des droits de l'utilisateur.

Accès aux fichiers présents sur la machine physique : Montages d'autres dossiers --bind

L'Utilisation de l'option **--bind** ou -B permet de monter plusieurs dossiers dans le conteneur.

```
$ singularity shell hello-world.sif        #sans montage
Singularity hello-world.sif:/home/biotools/singularity_img> ls -l /bighub/hub
ls: cannot access /bighub/hub: No such file or directory
```

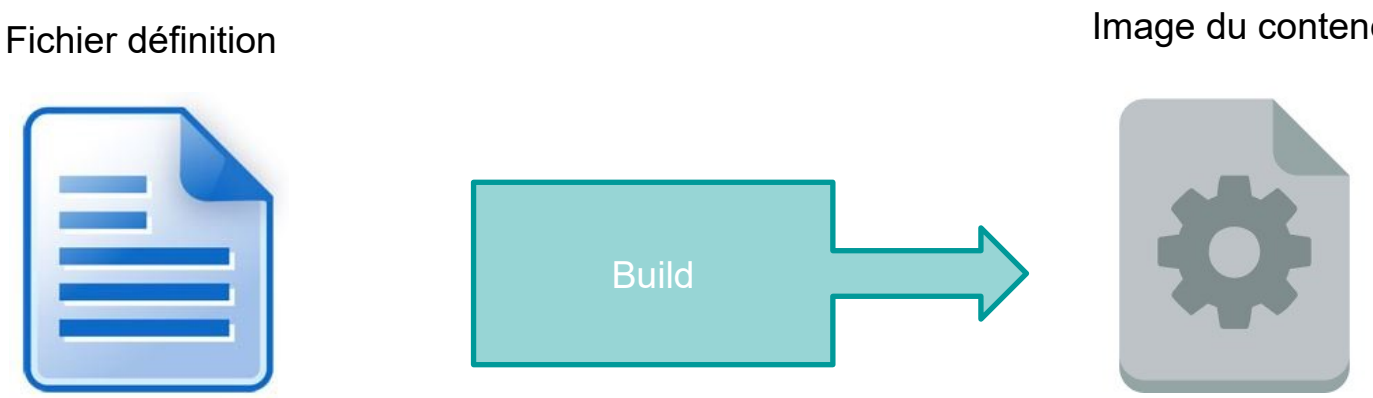
```
$ singularity shell --bind /bighub/hub hello-world.sif  #avec montage
Singularity hello-world.sif:/home/biotools/singularity_img> ls /mnt
Singularity hello-world.sif:/home/biotools/singularity_img> ls -l /bighub/hub
total 60
drwx------  9     1124 bioinfo 4096 Jul 20  2017 BACKUP_COMPUTER
drwxrwxr-x  9     1124 bioinfo 4096 Jan 23 10:55 BIG
```

- Montage de plusieurs directory : -B path_dir1, path_dir2
- Définir si le montage est en lecture seule. (par défaut, on a les droits d'écriture)
- Définir le nom du point de montage

```
$ singularity shell --bind /bighub/hub /mnt:ro hello-world.sif
$ singularity shell --bind /bighub/hub:/mnt:rw hello-world.sif
```

RO => lecture
RW => ecriture

Création d'un conteneur



Création d'un fichier de définition qui va décrire :

- l'OS
- les commandes d'installation des packages
- les commandes d'installation des programmes contenus dans le conteneur

Création d'un conteneur : build

La commande **build** crée à partir d'un fichier de défintion une "image" singularity corrrespondant.

Attention : Le build se fait en tant que ROOT!!!

```
#singularity build  ubuntu_docker.sif ubuntu_docker.def
INFO:    Starting build...
Getting image source signatures
Copying blob sha256:f476d66f540886e2bb4d9c8cc8c0f8915bca7d387e536957796ea6c2f8e7dfff
 35.48 MiB / 35.48 MiB [=====================================================] 1s
Copying blob sha256:8882c27f669ef315fc231f272965cd5ee8507c0f376855d6f9c012aae0224797
 851 B / 851 B [=============================================================] 0s
Copying blob sha256:d9af21273955749bb8250c7a883fcce21647b54f5a685d237bc6b920a2ebad1a
 547 B / 547 B [=============================================================] 0s
```

Création d'un conteneur : Description d’un fichier de définition

```
BootStrap: docker
From: ubuntu:16.04

%help
My first train container
Run the sl train

%setup
mkdir ${SINGULARITY_ROOTFS}/data
touch ${SINGULARITY_ROOTFS}/data/soulTrain.txt
cp /home/biotools/singularity_img/test.txt ${SINGULARITY_ROOTFS}/data/soulTrain.txt
%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 "$*"
```

Création d'un conteneur : Header

Décrit l'OS:

- Source (hub,scratch)
- Version
- Paquet(s) à inclure

```
Bootstrap: docker
From: ubuntu:latest
```

```
BootStrap: library
From: ubuntu:18.04
```

Supporte les sources suivantes:

- shub (images hosted on Singularity Hub)
- docker(images hosted on Docker Hub)
- localimage (images saved on your machine)
- yum (yum based systems like CentOS or Scientific Linux)
- debootstrap (apt based systems like Debian orUbuntu)
- arch (Arch Linux)
- busybox
- Zypper (zypper based systems like Suse or OpenSuse)

Description fichier de définition : Section Help

%help : Le créateur du fichier de définition de l'image devrait normalement indiquer comment utiliser le conteneur dans cette section.

Syntaxe dans le fichier de définition

```
%help
Help description
version 2019-01-24
```

Commande pour afficher le help : **singularity run-help image.sif**

```
$ singularity run-help train.sif
My first train container
Run the sl train

Help me. The train is stuck in this container.
```

Description fichier de définition : Section label

%labels : Le créateur du fichier de définition de l'image va pouvoir noter les métadonnées associées au conteneur. Exemple : créateur , version

Syntaxe dans le fichier de définition

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

Commande pour afficher les labels : singularity inspect image.sif

Description fichier de définition : Section setup

%setup : Commandes exécutées par l'OS hôte lors du build de l'image de base.

Attention : Donc sur votre machine alors que vous êtes root!

Syntaxe dans le fichier de définition

```
%setup
mkdir ${SINGULARITY_ROOTFS}/data
touch ${SINGULARITY_ROOTFS}/data/soulTrain.txt
cp /home/biotools/singularity_img/test.txt ${SINGULARITY_ROOTFS}/data/soulTrain.txt
touch test2
```

${SINGULARITY_ROOTFS} : indique la racine du conteneur

Exemple d'utilisation : Création de fichier ou de dossiers

Description fichier de définition : Section post

%post : Commandes exécutées seulement dans le conteneur lors du build après l'installation de l'OS.
Permet l'installation des applications ou des package spécifiques au conteneur.

Attention : pas d'interactivités possibles !

Syntaxe dans le fichier de définition

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

Description fichier de définition : Section environment

%environment : Définis dans le conteneur les variables d'environnement
Exemples : Chemin d'accés à un programme
Langue à prendre en compte.

Syntaxe dans le fichier de définition

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

Description fichier de définition : Section files

%files : Importer des fichiers vers le conteneur.

Syntaxe dans le fichier de définition

```
%files
  <source> <destination>
```

Description fichier de définition : Section runscript

%runscript : Définir les commandes à lancer lorsqu'on exécute le conteneur avec **run.**
Simplifier l'utilisation du conteneur au détriment de sa polyvalence

Syntaxe dans le fichier de définition

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

Création d'un conteneur modifiable : option --sandbox / --writable

Objectif : pouvoir faire des tests d'installation
Par ex : déterminer les dépendances nécessaires à l'installation d'un outil.

Commande pour builder un conteneur modifiable : singularity build --sandbox

```
$sudo singularity build --sandbox ubuntu_writable/ library://ubuntu
```

Commande pour utiliser un conteneur modifiable : singularity shell --writable

```
$sudo singularity exec --writable ubuntu_writable touch /foo
$singularity exec ubuntu/ ls /foo
$singularity exec ubuntu_writable/ ls /foo
```

Utilisation de BioConda dans un conteneur

Conda gestionnaire de paquets open-source :

- conçu pour gérer des paquets et dépendances des programmes
- distribution des outils précompilés,
- gestionnaire d'environnements virtuels, sur le même principe que les environnements virtuels de Python.

L’installation de paquet conda est rapide, robuste et facile

Conda offre la possibilité d'ajouter d'autres sources de paquets, aussi appelées channels.

Spécialisé dans les outils bioinformatiques, le channel Bioconda contient des paquets pour déployer des outils bioinformatiques. (https://bioconda.github.io/)

Utilisation de BioConda dans un conteneur : exemple

```
Bootstrap: library
From: ubuntu:18.04

%help
PfamScan
Run functional annotation

%setup
mkdir ${SINGULARITY_ROOTFS}/home/tools
%files
/home/singularity/tarball/Miniconda3-latest-Linux-x86_64.sh /home/tools/.
%post
apt-get -y update
apt-get -y install wget build-essential
apt-get -y install g++
apt-get -y install make
apt-get -y install bash

cd /home/tools/
yes y | bash ./Miniconda3-latest-Linux-x86_64.sh -b -p /home/tools/miniconda

cd /home/tools/miniconda/bin/

yes y | ./conda config --add channels defaults
yes y | ./conda config --add channels bioconda
yes y | ./conda config --add channels conda-forge

yes y | ./conda install -c bioconda pfam_scan
yes y | ./conda install -c bioconda/label/cf201901 pfam_scan

%labels
Maintainer Corinne Rancurel & Martine Da Rocha & Arthur Pere
Updater  Corinne Rancurel<corinne.rancurel@inrae.fr>
ContainerVersion v1
Software

%environment
    export LC_ALL=en_US.utf8
    export PATH=/home/tools/miniconda/bin:$PATH
%runscript
```

Utilisation du repository galaxyproject

Le **GalaxyProject** est un projet open-source , qui à pour but à travers une interface web de permettre de faire des analyse bioinformatique de façon accessible, reproductible et transparente.
https://galaxyproject.org/

Pour facilité la reproductivité des analyses le **GalaxyProject** mets à disposition de la communauté des images singularity . Repository : https://depot.galaxyproject.org/singularity/

Récupération de conteneur Singularity repository galaxyproject : copier l'adresse du lien / utiliser wget

```
wget https://depot.galaxyproject.org/singularity/fasta3%3A36.3.8--0
```

Fonctionnement du conteneur : Certain de ces conteneurs ne montent pas automatiquement le /home/ de l'utilisateur.
Il est nécessaire de **montée tous les dossiers** auxquels il doit avoir accès en utilisation la commande **--bind**.

```
singularity exec \
--bind /home/share/bioinfo/miARN_synthése_prédiction/bin:/home/share/bioinfo/miARN_synthése_prédiction/bin:rw,/l
ssearch36 \
/home/share/bioinfo/miARN_synthése_prédiction/bin/prediction_miARNfa \
/lerins/hub/DB/MIRBASE/matures_miRNA_v22.fa
```

SLURM : Gestionnaire de ressources

SLURM est un « JOB SCHEDULER » ou un gestionnaire de ressources

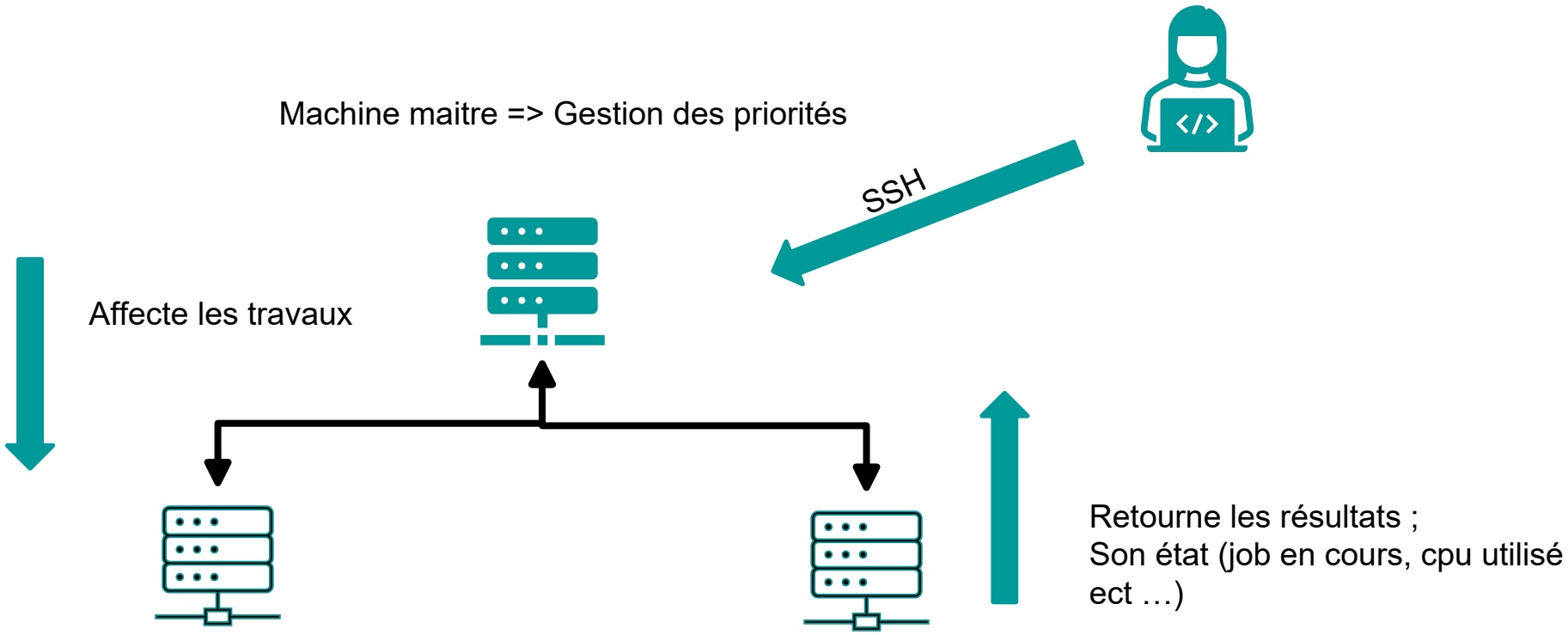
Sert à gérer les ressources d'une machine multi-utilisateurs multi-taches en permettant:

- de choisir les ressources physiques à utiliser
- de planifier le lancement des calculs (heure de départ, durée limite …)
- de réserver les ressources nécessaire au bon déroulement d'un calcul
- d'ordonnancer les « JOB » lancés par les utilisateurs

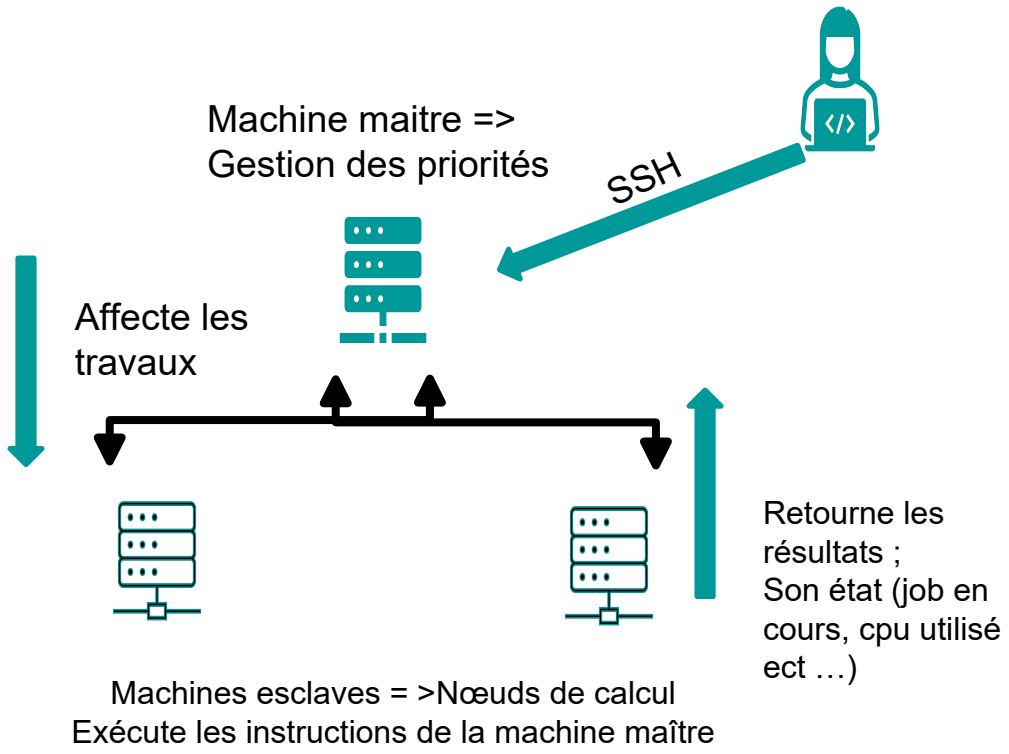
En tant que gestionnaire de ressources de informatiques, Slurm a trois fonctions principales :

- il alloue l'accès aux ressources (les nœuds de calcul) pour les utilisateurs pendant une durée de temps définie afin qu'ils puissent effectuer des tâches ;
- il fournit un cadre pour le démarrage, l'exécution et le suivi des tâches (normalement ce sont des tâches qui sont parallèles) sur l'ensemble des nœuds affectés ;
- il arbitre l'accès aux ressources par la gestion d'une file d'attente des tâches en cours.

SLURM est un « JOB SCHEDULER » ou un gestionnaire de ressources



SLURM : Structure gestionnaire de ressource de BIG



Connexion à la machine maitre : ssh –X login@147.99.85.52
Nœuds de calcul : - ipn-geno (62 cpu; 500G RAM)
- bigbang (70 cpu ; 750G RAM)

Une partition dans SLURM correspond à un groupement logique de nœuds de calcul. Chaque partition est associée à des contraintes en termes de ressources (en particulier le temps de calcul maximum d'un job).

On à trois partitions différentes :

- **treed** : par défaut, limitée à 3 jours de calculs, exclusivement sur le nœud ipn-geno.
- **infinity** : illimité en temps de calcul, exclusivement sur le noeud bigbang,
- **all** : illimité en temps de calcul, utilise les deux nœud (ipn-geno et bigbang), moins prioritaire que pour les autres.

Les logiciels ne sont pas installé sur les serveurs de calcul, ils doivent être conteneurisé dans des **images singularity**.
Pour permettre l'utilisation de plusieurs version de singularity nous utilisons des **modules**. Les modules sont des fichiers de configuration qui contiennent des instructions pour modifier votre environnement logiciel.

Principe :
A partir d'une connexion au frontal du cluster, lancer votre calcul en précisant vos besoins en termes de ressources (coeurs, noeuds …), on peut aussi préciser la durée approximative du Job.

Comment le faire ?
Deux modes principaux de fonctionnement sont possibles :

• **srun** : en interactif réserve les ressources demandées et ouvre une session sur un noeud du cluster. L'utilisateur a ensuite la main pour lancer son application ou tout autre commande. Dès lors que l'utilisateur quitte sa « session » il libère automatiquement les ressources.

•**sbatch** : les ressources requises, la durée du calcul, le(s) job(s) à lancer sont décrits dans un script, lancé en tâche de fond. Les ressources seront libérés à la fin du job, à la fin du temps fixé, ou par la suppression du job (scancel). => **solution à privilégier**

Gérer ses travaux lancés sur le cluster

Obtenir la liste des jobs tournant ou en attentant sur le cluster de calcul :

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES  CPUS MIN_MEMORY NODELIST(REASON)
   71663_[111-120]       all    hmmer cbelliar PD       0:00      1     4        32G (Priority)
    71631_[32-110]       all    hmmer cbelliar PD       0:00      1     4        32G (Resources)
          71631_1       all    hmmer cbelliar  R    1:11:10      1     4        32G ipn-geno
          71631_2       all    hmmer cbelliar  R    1:11:10      1     4        32G ipn-geno
```

Obtenir la liste des jobs spécifique à un utilisateur :

```
$ squeue -u $USER
```

Recueillir des informations sur un job lorsqu'il est en cours:

```
$ scontrol show job jobid
```

Recueillir des informations sur un job lorsqu'il est finis:

```
sacct -j job_id
```

Supprimé un job:

```
$ scancel jobid
```

Séquence à suivre lors de la soumission d'un travail en mode BATCH sur le cluster :

- écrire le script de soumission monscript.sh
- soumettre le batch: sbatch monscript.sh
- Suivre son déroulement (squeue, ….)
- vérifier le fichier d'erreur .err
- vérifier les résultats

Création d'un script pour le mode BATCH : Générateur d'entête de script bash

http://big-project.sophia.inrae.fr/BYUHPC/

Parameters	
Limit this job to one node:	☑
Number of processor cores **across all nodes**:	1
Number of CPU per task	1
Memory per CPU or Nodes ?	☐
Memory per Nodes:	1 GB
Walltime:	0 days 1 hours 00 mins 00 secs
I am in a file sharing group and my group members need to read/modify my output files:	☑
Group name (case sensitive):	bioinfo
Job name:	
Receive email for job events:	☐ begin ☐ end ☐ abort ☐ all
Email address:	myemail@inra.fr
Partitions:	☐ **treed** nb CPUs: 62 Memory: 512 Go ☐ **infinity** nb CPUs: 70 Memory: 765 Go ☐ **all** nb CPUs: 132 Memory: 512 Go

Job Script

Script format: Slurm

```
#!/bin/bash

#Submit this script with: sbatch thefilename

#SBATCH --time=1:00:00   # walltime
#SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
#SBATCH --cpus-per-task=1     # number of CPU per task
#SBATCH --nodes=1   # number of nodes
#SBATCH --mem=1G   # memory per Nodes
#SBATCH --gid=bioinfo


# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
```

Création d'un script pour le mode BATCH : exemple

Les conteneurs crées sont **/home/singularity/image**. Ce dossier est accessible dans le cluster de calcul.

Exemple de script

```
#!/bin/bash

# submit : sbatch file.sh

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=64
#SBATCH --nodes=1
#SBATCH -p infinity
#SBATCH --mem=500G
#SBATCH -J "Trinity"
#SBATCH --mail-user=login@inrae.fr
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END

module purge
module load singularity/3.5.3

SING_IMG=/home/singularity/images/Trinity_2.4.sif

singularity exec $SING_IMG Trinity --seqType fq --left 1A_1P --right 1A_2P --SS_lib_type FR --full_cleanup --nor
put trinity_1A_FR --max_memory 500G --CPU 64 > trinity.log
```

Création d'un script pour le mode BATCH : exemple

Les conteneurs téléchargés sont à déposé dans le dossier **/lerins/hub/DB/SINGULARITY_GALAXY/**. Ce dossier est accessible dans le cluster de calcul.

Exemple de script

```
#!/bin/bash

# submit : sbatch file.sh

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=64
#SBATCH --nodes=1
#SBATCH -p infinity
#SBATCH --mem=500G
#SBATCH -J "Trinity"
#SBATCH --mail-user=login@inrae.fr
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END

module purge
module load singularity/3.5.3

SING_IMG=/lerins/hub/DB/SINGULARITY_GALAXY/trinity:2.12.0--ha140323_1

singularity exec $SING_IMG Trinity --seqType fq --left 1A_1P --right 1A_2P --SS_lib_type FR --full_cleanup --nor
put trinity_1A_FR --max_memory 500G --CPU 64 > trinity.log
```

Nextflow

Outil permettant de définir des jobs et d'automatiser l'exécution :

- Faciliter la reproductibilité des analyses
- Faciliter la publication des workflows
- Faciliter l'utilisation des workflows sur différentes plateformes
- PC, Serveur, Cluster HPC, Cloud
- Industrialiser vos workflows

https://www.nextflow.io/

Chaque workflow correspond à un dossier comprenant 3 fichiers et un dossier :

- un fichier contenant les commandes des différentes étapes du worflow nextflow**.nf**. Il ne doit pas être modifié sinon création d'un nouveau workflow.
- un fichier contenant la configuration nextflow**.config**. Ce fichier doit être modifié pour adapter le worflowaux données à analyser.
- un script d'exemple pour le lancement du workflow sur notre cluster de calcul script**.sh**. Modification possible notamment email/nom du job.
- un dossier module contenant les différentes étapes du workflow et leurs fichiers de définition.

Exemple nextflow contrôle de la qualité

```
$ cd /lerins/hub/DB/NEXTFLOW/DSL2/CQ/
$ ls -1
CQ.nf
module
nextflow_CQ.config
script_CQ.sh
```