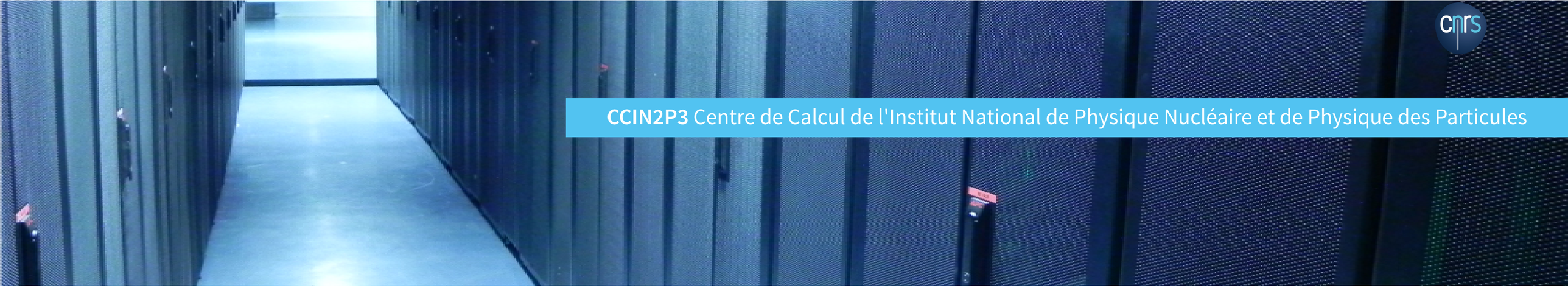


APPTAINER

DES CONTENEURS POUR LE CALCUL SCIENTIFIQUE

GUILLAUME COCHARD - CC-IN2P3

ANF UST4HPC - 21 JUIN 2023

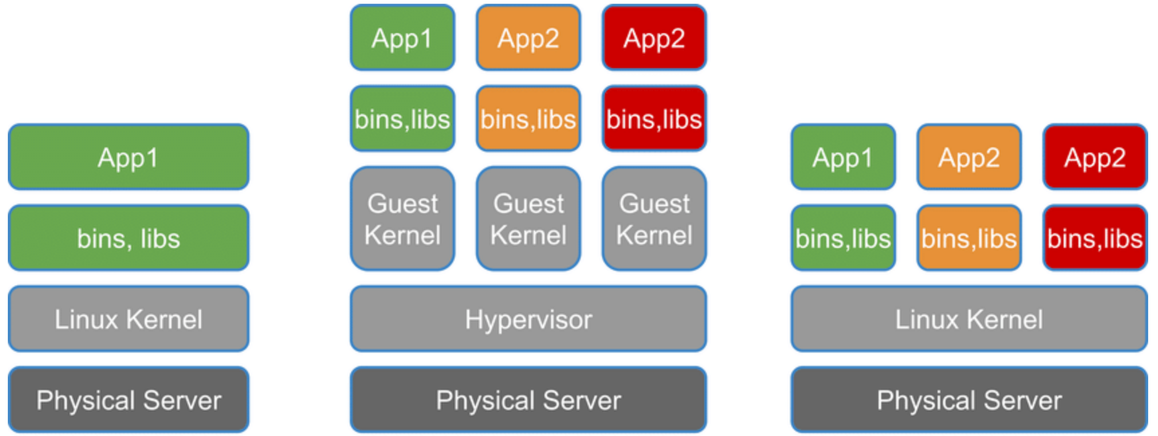
- Rôle du centre de calcul : mutualiser les ressources informatiques nécessaires aux expériences liées à l'IN2P3 : stockage, calcul, hébergement web.
- Pour la partie calcul, le CC possède deux clusters :
  - Grille de calcul (WLCG)
    - Une dizaine d’utilisateurs (“grosses” expériences type Atlas ou Alice)
    - Ordonnanceur : HTCondor
    - 765 workers pour 37 000 CPUs
    - Jusqu’à 50 000 jobs « standardisés » par jour
  - Ferme locale
    - Plusieurs centaines d’utilisateurs de l’IN2P3
    - Ordonnanceur : Slurm (anciennement UGE/Grid Engine)
    - 423 workers pour 22500 CPUs et 80 GPUs
    - Jusqu’à 120 000 jobs variés par jour

Plan

- Introduction : conteneurisation et calcul scientifique
- Apptainer
- Utiliser un conteneur Apptainer
  - Télécharger et utiliser un conteneur
  - Utilisateurs et droits
  - Montages
- Construire un conteneur
- Apptainer côté administrateur
  - Installation et configuration
  - Implémentation au CC

RAPPELS SUR LA CONTENEURISATION

VIRTUALISATION



(a) Bare Metal (b) Virtual Machine (c) Container

- **User Namespace** : table de conversion entre les users id du conteneur et ceux du système hôte. Permet une isolation entre les deux, ce qui autorise par exemple à avoir un utilisateur d'id 0 (root) dans le conteneur ayant en réalité un id non privilégié sur le système.
- **Runtime** : logiciel gérant la partie système. Soit bas niveau (appels système, cgroups, etc.), soit haut niveau (réseau). *Ex : runc, containerd*
- **Container Engine** : logiciel gérant les interfaces haut niveau (entrées utilisateurs, registries, etc.) et s'interfaçant avec le runtime. *Ex : Docker, Apptainer, etc.*
- **OCI** : *Open Container Initiative*, organisme maintenant des standards pour les runtime et engines (notamment pour les CLI) ainsi que pour les images.

- **Image** vs. **conteneur** : l'image décrit le conteneur, le conteneur est l'image une fois lancée
- **Manifeste** (ou recette) : fichier texte contenant les instructions pour construire un conteneur. *Ex : Dockerfile*
- **Registre** : service hébergeant des images. *Ex : Docker Hub*
- **Repository** : collection d’images liées différenciées par des tags. *Ex : Python repository*

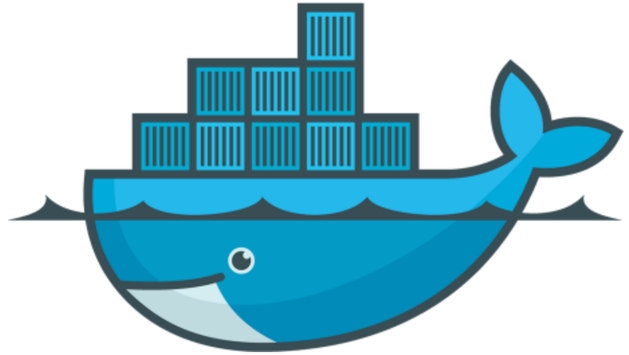
- **Personnalisation** : gestion côté utilisateur des dépendances (*BYOB: Bring Your Own Code*)
- **Portabilité** : le code est dissocié du hardware et du site
- **Isolation** : l'application tourne dans un système isolé
- **Reproductibilité** (?) : l'application n'est pas dépendante des évolutions du site

A way of packaging an application and all its dependancies ensuring cross-system portability

Et Docker alors ?

Malgré toutes ses qualités, Docker n'a pas été conçu pour le calcul :

- Démon root sur tous les workers
- Images construites en layers
- Isolation matérielle : pas de support MPI, support GPU non natif



APPTAINER

- Fork de Singularity (version 1.0.0 en mars 2022) maintenu par CIQ (Rocky Linux) et appartenant à la Linux Foundation
- Projet initialement pensé pour le calcul scientifique
- Développement actif
  - Version actuelle : 1.1.9 (version 1.2.0 en RC)
  - Environ une mise à jour par mois
- Très populaire dans le calcul scientifique
  - Côté expériences : WLCG (LHC), KM3NET…
  - Côté sites : CERN, la plupart des grands sites américains… (et le CC !)
  - Côté bonnes pratiques : conseillé pour l'Open Science

- Pas de démon, pas de *root** pour lancer une image
  - Pas de risque de *user escalation*
  - Installation et configuration très simple
  - Utilisation adaptée aux gestionnaires de batch
  - * : sauf si Apptainer est en mode setuid
- *“Integration over isolation”*
  - Support natif de MPI et GPU
  - Facile d'accéder à des données extérieures au conteneur
- Image sous la forme d'un unique fichier SIF
  - Facilité d'utilisation et de partage
  - Signature et chiffrement des images (chiffrement uniquement en mode setuid)
- Compatible images OCI (dont Docker)

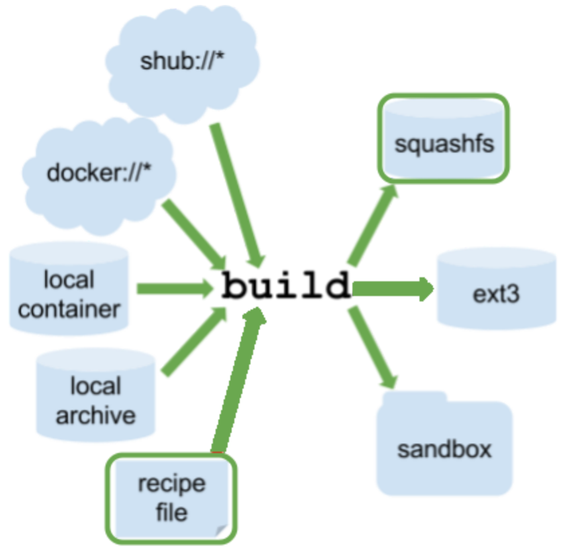
Bref historique d'Apptainer

- 2015 : Singularity, codé en Python et orienté HPC, est créé à Berkeley
- 2018 : création de l’entreprise Sylabs
- 2018 : version 3, réécriture complète en Go, nouvelles fonctionnalités
- 2021 : SingularityPro / SingularityCE maintenus par Sylabs, fork libre maintenu par CIQ (Rocky Linux)
- 2022 : la version libre rejoint la Linux Foundation et devient Apptainer

- Singularity existe toujours et est un produit de Sylabs
- Sylabs offre des services :
  - Builder online d'image
  - Registry (shub)
  - Support
- Pour le moment Apptainer et Singularity évoluent en parallèle
  - Dans un monde idéal, Apptainer serait la source upstream pour Singularity

Plus de détails sur les différences sur le site d'Apptainer

Cycle de vie


shub://*
docker://*
local container
local archive
recipe file
build
squashfs
ext3
sandbox

UTILISATION D'UN CONTENEUR

UTILISATION D'UN CONTENEUR

TÉLÉCHARGER ET UTILISER UN CONTENEUR

Apptainer: télécharger une image

- `pull` : télécharger une image depuis une registry

```
[chocolate@test-apptainer ~]$ apptainer pull docker://python:3.11.3-alpine3.17
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
Copying blob 796bfcef9dd7 done
[…]
Writing manifest to image destination
Storing signatures
2023/06/07 10:54:49  info unpack layer: sha256:f56be85fc22e46face30e2c3de3f7fe7c15f8fd7c4e5add29d7f64b87abdaa09
[…]
INFO:    Creating SIF file…

[chocolate@test-apptainer ~]$ ls
python_3.11.3-alpine3.17.sif
```

Apptainer: métadonnées d'une image

- `inspect` : afficher les métadonnées de l'image

```
[chocolate@test-apptainer ~]$ apptainer inspect python_3.11.3-alpine3.17.sif
org.label-schema.build-arch: amd64
org.label-schema.build-date: Wednesday_7_June_2023_10:54:50_CEST
org.label-schema.schema-version: 1.0
org.label-schema.usage.apptainer.version: 1.1.8-1.el7
org.label-schema.usage.singularity.deffile.bootstrap: docker
org.label-schema.usage.singularity.deffile.from: python:3.11.3-alpine3.17
```

- `inspect --deffile` : afficher le manifeste ayant servi à construire l'image

```
[chocolate@test-apptainer ~]$ apptainer inspect --deffile bakery.sif
Bootstrap: docker
From: python:3.11.4-alpine3.18
```

Apptainer: aide d'une image

- `run-help` : afficher l'aide de l'image

```
[chocolate@test-apptainer ~]$ apptainer run-help python_3.11.3-alpine3.17.sif
No help sections were defined for this image
```

Apptainer: lancer le conteneur

- `run` : exécuter le script par défaut (*runscript*)

```
[chocolate@test-apptainer ~]$ apptainer run python_3.11.3-alpine3.17.sif
Python 3.11.3 (main, May  3 2023, 08:52:44) [GCC 12.2.1 20220924] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
[chocolate@test-apptainer ~]$
```

Apptainer: accéder au conteneur

- `shell` : accéder au shell par défaut du conteneur

```
[chocolate@test-apptainer ~]$ echo $SHELL
/bin/bash

[chocolate@test-apptainer ~]$ apptainer shell python_3.11.3-alpine3.17.sif
Apptainer> echo $SHELL
/bin/sh
Apptainer> cat /etc/issue
Welcome to Alpine Linux 3.17
Kernel \r on an \m (\l)
Apptainer> exit  #ou ctrl-d
```

Apptainer: lancer une commande

- `exec` : lancer une commande dans le conteneur

```
[chocolate@test-apptainer ~]$ apptainer exec python_3.11.3-alpine3.17.sif cat /etc/issue
Welcome to Alpine Linux 3.17
Kernel \r on an \m (\l)

[chocolate@test-apptainer ~]$
```

Options d'exécution

- `--nv | --rocm` : pour utiliser les GPUs (nvidia | AMD)
- `--writable` : pour ouvrir le conteneur en RW (demande une image spéciale)

```
[chocolate@test-apptainer ~]$ apptainer shell --nv python_3.11.3-alpine3.17.sif
[chocolate@test-apptainer ~]$ apptainer shell --writable python_3.11.3-alpine3.17.sif
```

- L'étape du `pull` peut être omise

```
[chocolate@test-apptainer ~]$ apptainer shell --nv docker://python:3.11.3-alpine3.17
```

- Attention à l'ordre des arguments !

UTILISATION D'UN CONTENEUR

UTILISATEURS ET ENVIRONMENT

Gestion utilisateur

- L’utilisateur dans le conteneur est celui qui a lancé le conteneur

```
[chocolate@test-apptainer ~]$ id
uid=40309(chocolate) gid=40309(chocolate) groups=40309(chocolate)

[chocolate@test-apptainer ~]$ apptainer shell python_3.11.3-alpine3.17.sif
Apptainer> id
uid=40309(chocolate) gid=40309(chocolate) groups=40309(chocolate)
```

- `--fakeroot` : pour ouvrir le conteneur en simulant l'utilisateur root
  - L'utilisateur dans le conteneur sera root (id 0) et peut donc modifier tous les fichiers du conteneurs si le conteneur est ouvert en RW
  - Les modifications ne sont possibles que si l'utilisateur ayant lancé le conteneur à les droits de modification de ces fichiers sur le système hôte (droits RW sur l'image .SIF par exemple)

```
[chocolate@test-apptainer ~]$ apptainer shell --fakeroot python_3.11.3-alpine3.17.sif
Apptainer> id
uid=0(root) gid=0(root) groups=0(root)
```

Variables d'environnement

- Par défaut les variables d'environment sont propagées

```
[chocolate@test-apptainer ~]$ export DESSERT="brownie"
[chocolate@test-apptainer ~]$ echo $DESSERT
brownie
[chocolate@test-apptainer ~]$ apptainer shell python_3.11.3-alpine3.17.sif
Apptainer> echo $DESSERT
brownie
```

- `--cleanenv` : permet d'éviter la propagation des variables d'environment

```
[chocolate@test-apptainer ~]$ apptainer shell --cleanenv python_3.11.3-alpine3.17.sif
Apptainer> echo $DESSERT

Apptainer>
```

UTILISATION D'UN CONTENEUR

DONNÉES, LECTURE ET ÉCRITURE

- Par défaut le conteneur est en lecture seule (RO) et les données à l'extérieur de celui-ci sont inaccessible
- Les bindings (montages) permettent de monter un répertoire du système hôte dans le conteneur
- Il y a deux types de bindings :
  - Système : définis dans la configuration et sont effectués automatiquement sauf si spécifié autrement
  - Utilisateur : définis dans la commande utilisée par l'utilisateur
- Par défaut les montages sont en lecture-écriture (RW)
  - Utiles pour les données tant en entrée qu'en sortie
  - Permettent la polyvalence des images : pas besoin de build plusieurs images si seules les données changent
- Montages par défaut (tous en RW) :
  - `$HOME`
  - `$PWD`
  - `/tmp`, `/var/tmp`
  - `/sys`, `/proc`
  - `/etc/resolv.conf`
  - `/etc/passwd`

Exemple avec $HOME

- Montage automatique de $HOME :

```
[chocolate@test-apptainer ~]$ ls
alpine_latest.sif  file1.txt
[chocolate@test-apptainer ~]$ cat file1.txt
what is the best dessert?

[chocolate@test-apptainer ~]$ apptainer shell alpine_latest.sif
Apptainer> echo "chocolate is the best" >> file1.txt
Apptainer> cat file1.txt
what is the best dessert?
chocolate is the best
Apptainer> ls
alpine_latest.sif  file1.txt
Apptainer> exit

[chocolate@test-apptainer ~]$ cat file1.txt
what is the best dessert?
chocolate is the best
```

Ajouter des montages

- `--bind src[:dest[:opts]]` : monte un répertoire dans l'image
  - `src` est sur la machine hôte, `dest` dans le conteneur
  - `dest` doit être absolu et peut être omis si `src == dest`
  - Par défaut les montages sont en RW, sauf si `opts == ro`
  - `--bind` peut être remplacé par `-B`
- Par défaut `dest` sera créée si elle n'existe pas (overlay/underlay)

```
[chocolate@test-apptainer ~]$ ls piecemontee/
chantilly
[chocolate@test-apptainer ~]$ apptainer shell -B piecemontee:/mnt alpine_latest.sif
Apptainer> ls /mnt
chantilly
```

Retirer des montages

- `--no-mount <mount_name>` : désactive le montage automatique de `mount_name`
  - Donner le nom, pas le point de montage : `tmp`, pas `/tmp`
  - Plusieurs noms de montage peuvent être donnés (séparés par des virgules)

```
[chocolate@test-apptainer ~]$ apptainer shell alpine_latest.sif
Apptainer> ls /tmp
[some files]

[chocolate@test-apptainer ~]$ apptainer shell --no-mount tmp alpine_latest.sif
Apptainer> ls /tmp
Apptainer>
```

Autres fonctionnalités moins utiles en HPC

- instances : Apptainer en tant que service
- cgroups : limiter les ressources utilisées par un conteneur
- overlay persistant : rendre une image accessible en écriture

CONSTRUIRE UN CONTENEUR

CONSTRUIRE UN CONTENEUR

LE MANIFESTE

Le manifeste

- Manifeste = Recette (Recipe) = Definition File
- Permet de construire un conteneur personnalisé
- Construit from scratch ou depuis une image (locale ou non)
- Deux principales parties :
  - Le **header** (obligatoire): définissent l'OS de l'image
  - Les **sections** (optionnelles): configurent et personnalisent l'image
- Commentaires inline avec `#`

Build l'image

- `apptainer build [options] <image> <spec>` : builder une image
  - En root
  - `spec` : fichier(s) décrivant l'image. Manifeste, image sous forme de sandbox, etc.
  - `options` : options de build, par exemple `--sandbox`
- Exemple : construire l'image `ubuntu.sif` à partir du manifeste :

```
$ sudo apptainer build ubuntu.sif ubunturecipe
```

CONSTRUIRE UN CONTENEUR

LE MANIFESTE

LE HEADER

- Décrit l'OS
  - Source (image d'une registry ou locale, scratch, etc.)
  - Type et version de l'OS
  - Paquet(s) de base à inclure
  - Doit commencer par la ligne `Bootstrap` qui indique quel agent bootstrap est utilisé pour installer l'OS de base
- Les principales sources sont :
  - docker : depuis Docker Hub
  - oras : image issue d'un registre d'images OCI
  - localimage : image locale
  - yum : système basé sur yum (CentOS, Rocky)
  - debootstrap : système basé sur apt (Ubuntu, Debian)

Le header

- Exemple avec Docker

```
Bootstrap: docker
From: debian:7
```

- Exemple avec yum

```
Bootstrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum
```

CONSTRUIRE UN CONTENEUR

LE MANIFESTE

LES SECTIONS

Les sections

- Configurent le conteneur
- De nombreuses sections sont disponibles et sont toutes optionnelles :
  - Sections d'information
    - Help
    - Labels
  - Sections de configuration et personnalisation
    - Setup
    - Files
    - Environment
    - Post
  - Sections d'exécution
    - Test
    - Runscript
- Il peut y avoir plusieurs fois la même section

Section labels

- Section `labels` : complète les métadonnées de l'image : auteur, version, etc.
  - Couple `nom valeur`
  - Les noms champs sont libres

```
Bootstrap: docker
From: ubuntu

%labels
    Maintainer ChocolateLover
    TypeOfChocolate Dark
```

```
$ apptainer inspect ubuntu.sif
Maintainer: ChocolateLover
TypeOfChocolate: Dark
org.label-schema.build-arch: amd64
[…]
```

Section help

- Section `help` : définit l'aide associée au conteneur

```
%help
Why do you need help when you have chocolate?
```

```
$ apptainer run-help ubuntu.sif
Why do you need help when you have chocolate?
```

- Section `setup` : commandes exécutées par l'OS hôte après le build de base
  - Attention : ces commandes s'exécutent en root sur l'OS hôte. L'usage de cette section n'est conseillé que s'il est impossible de faire autrement
  - La variable `${SINGULARITY_ROOTFS}` donne le chemin vers le répertoire contenant l'image en cours de build

```
%setup
    touch ${SINGULARITY_ROOTFS}/test1
    touch test2
```

```
# apptainer build ubuntu.sif ubunturecipe
[…]
INFO:    Running setup scriptlet
+ touch /tmp/build-temp-3784265229/rootfs/test1
+ touch test2
[…]
```

```
$ ls
alpine_latest.sif  test2  ubunturecipe  ubuntu.sif

$ apptainer exec ubuntu.sif ls /
[…]  test1 […]
```

- Section `files` : copie des fichiers vers l'image
  - Une ligne par fichier
  - *Variable expansion* acceptée

```
$ touch test{1,2,3} file{1,2}

$ cat ubunturecipe
[…]
%files
    make_dessert.py /opt
    test* /opt
    file2 file2 /opt  # Ne marchera pas ! "INFO: Copying file1 to file2"
```

```
$ apptainer exec ubuntu.sif ls /opt
make_dessert.py test1 test2 test3
```

Section environment

- Section `environment` : définit les variables d'environnement disponibles dans le conteneur
  - Les valeurs ne peuvent pas être dynamiques

```
%environment
    export IS_VANILLA_GOOD=false
```

```
$ apptainer shell ubuntu.sif
Apptainer> echo $IS_VANILLA_GOOD
false

$ apptainer exec ubuntu.sif echo $IS_VANILLA_GOOD

$
```

Section post

- Section `post` : commandes exécutées dans le conteneur
  - Étape principale de personnalisation
  - Attention : ne pas utiliser de commande interactive

```
%post
    apt-get update
    apt-get install -y python3
    NOW=`date`
    echo "export NOW=\"${NOW}\"" >> $APPTAINER_ENVIRONMENT
```

```
$ apptainer exec ubuntu.sif python3
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Section `runscript` : commandes à exécuter au lancement du conteneur avec `run`
  - Simplifie l'utilisation du conteneur au dépend de sa polyvalence
  - `$@` : permet d'interprêter les arguments données au lancement du conteneur

```
%runscript
    python3 /opt/make_dessert.py "$@"
```

```
$ apptainer run ubuntu.sif flour sugar
```

- Section `test` : commandes à exécuter à la fin du build ou via la CLI
  - Vérifier que les étapes importantes se sont bien déroulées
  - Vérifier que l'image fonctionne sur un type spécifique de matériel

```
%test
    if [ "$BEST_ICE_CREAM" = "notvanilla" ]; then
        echo "env var: ok"
    else
        echo "env var: not ok"
    fi
```

```
$ apptainer build ubuntu.sif ubunturecipe
[…]
INFO:    Adding testscript
INFO:    Running testscript
env var: ok
INFO:    Creating SIF file...
INFO:    Build complete: ubuntu.sif
[root@test-apptainer ~]# apptainer test ubuntu.sif
env var: ok
```

CONSTRUIRE UN CONTENEUR

LE MANIFESTE

AUTRES FONCTIONNALITÉS

Multi-stage

- Créé un environnement spécifique pour une étape et un autre pour le build final
  - Limite la taille finale de l'image

```
Bootstrap: docker
From: golang:1.12.3-alpine3.9
Stage: devel

%post
    […]
    go build -o hello hello.go

Bootstrap: library
From: alpine:3.9
Stage: final

%files from devel
    /root/hello /bin/hello

%runscript
    /bin/hello
```

- Il est possible d'installer plusieurs app sur une même image
  - Chaque app aura ses propres point d'entrée, métadonnées, environnement, etc.

```
Bootstrap: docker
From: ubuntu

%environment
    GLOBAL=variables
    AVAILABLE="to all apps"

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

# bar
%apphelp bar
    This is the help for bar.
%applabels bar
    BESTAPP BAR
%appinstall bar
    touch bar.exec
%apprun bar
    exec echo "RUNNING BAR"
%appenv bar
    SOFTWARE=bar
    export SOFTWARE
```

CONSTRUIRE UN CONTENEUR

SANDBOX

Sandbox

- Créé un conteneur dans un répertoire accessible en RW
  - Permet de tester interactivement certaines étapes du build
  - `shell|exec --writable` permet de lancer des commandes dans la sandbox

```
# apptainer build  --sandbox ubuntu/ ubunturecipe
[…]
INFO:    Build complete: ubuntu/

# ls ubuntu/
bin  boot  dev  environment  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  singularity  srv  sys  te

# apptainer exec --writable ubuntu apt-get install lolcow
# apptainer shell --writable ubuntu
Apptainer> touch /test2 /
Apptainer>
exit
[root@test-apptainer ~]# ls ubuntu/
bin  boot  dev  environment  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  singularity  srv  sys  te
```

Sandbox ⇔ SIF

- La commande build permet de convertir une sandbox en SIF et vice-versa
  - Les modifications éventuelles sont persistantes, mais non enregistrées dans le manifeste

```
# apptainer build ubuntu.sif ubuntu/
INFO:    Creating SIF file...
INFO:    Build complete: ubuntu.sif

# apptainer exec ubuntu.sif ls /
bin  boot  dev  environment  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  singularity  srv  sys  te
```

```
# apptainer build --sandbox ubuntu/ ubuntu.sif
INFO:    Creating sandbox directory...
INFO:    Build complete: ubuntu/
```

CONSTRUIRE UN CONTENEUR

SIGNATURE ET VÉRIFICATION

Signature

- Apptainer permet d’utiliser des clés PGP pour signer et vérifier des images
  - Trousseau local de clé situé dans $HOME/.apptainer/keys
  - Possibilité de configurer un serveur de clés distant
  - La signature d'une image est inclue dans ses métadonnées
  - Une image peut avoir plusieurs signatures, qui peuvent par exemple représenter les différentes étapes d'un workflow

Créer ses clés

- Apptainer gère lui-même les clés PGP utilisées

```
# apptainer key newpair
Enter your name (e.g., John Doe) : Guillaume Cochard
Enter your email address (e.g., john.doe@example.com) : guillaume.cochard@cc.in2p3.fr
Enter optional comment (e.g., development keys) : example keys
Enter a passphrase :
Retype your passphrase :
Generating Entity and OpenPGP Key Pair... done
```

```
# apptainer key list
Public key listing (/root/.apptainer/keys/pgp-public):

0) U: Guillaume Cochard (example keys) guillaume.cochard@cc.in2p3.fr
    C: 2023-06-08 15:55:04 +0200 CEST
    F: F830DD6ED19472D05ADBA464D2BEF17734BDF7F9
    L: 4096
    --------
```

Signer et vérifier un conteneur

- Signature :

```
# apptainer sign ubuntu.sif
Signing image: ubuntu.sif
Enter key passphrase :
Signature created and applied to ubuntu.sif
```

- Vérification :

```
# apptainer verify ubuntu.sif
Verifying image: ubuntu.sif
[LOCAL]   Signing entity: Guillaume Cochard (example keys) guillaume.cochard@cc.in2p3.fr
[LOCAL]   Fingerprint: F830DD6ED19472D05ADBA464D2BEF17734BDF7F9
Objects verified:
ID  |GROUP   |LINK    |TYPE
------------------------------------------------
1   |1       |NONE    |Def.FILE
2   |1       |NONE    |JSON.Generic
3   |1       |NONE    |FS
Container verified: ubuntu.sif
```

CONSTRUIRE UN CONTENEUR

CHIFFREMENT ET DÉCHIFFREMENT

- Apptainer permet de chiffrer un conteneur
  - Le conteneur est chiffré au repos, mais aussi lors de l'exécution. Le déchiffrement ne se fait qu'au niveau du runtime en mémoire
  - Chiffrement via passphrase ou via paire de clé RSA au format PEM/PKCS1 (mieux)
  - Attention : le déchiffrement n'est possible qu'avec Apptainer en mode setuid

Chiffrement avec passphrase

```
# APPTAINER_ENCRYPTION_PASSPHRASE=toto apptainer build --encrypt ubuntu.sif ubunturecipe

# apptainer sif list ubuntu.sif
------------------------------------------------------------------------------
ID   |GROUP   |LINK    |SIF POSITION (start-end)  |TYPE
------------------------------------------------------------------------------
1    |1       |NONE    |32176-32595               |Def.FILE
2    |1       |NONE    |32595-37496               |JSON.Generic
3    |1       |NONE    |37496-37692               |JSON.Generic
4    |1       |NONE    |40960-75481088            |FS (Encrypted squashfs/*System/amd64)

# apptainer shell ubuntu.sif
FATAL:   Unable to use container encryption. Must supply encryption material through environment variables or flags.

# APPTAINER_ENCRYPTION_PASSPHRASE=toto apptainer run ubuntu.sif
Apptainer>
```

- `--passphrase` : permet de donner le mot de passe de manière interactive

CONSTRUIRE UN CONTENEUR

BONNES PRATIQUES

- Être le plus précis possible dans les versions (python3 != python3.11)
- Définir toutes les métadonnées utiles (version, auteur, etc.) et l'aide (documentation, encore et toujours)
- Ne copier que les données immuables, le reste sera géré via des montages
- Ne pas mettre de données dans des répertoires qui serviront de point de montage (/home, /tmp)
- S'assurer que le manifeste est complet (pas de modification via sandbox) et le partager le manifeste avec le conteneur
- Signer l'image
- Il est aussi possible de convertir un Dockerfile en Apptainer Definition File : cf. documentation pour la correspondance des sections

ADMINISTRATION

Setuid vs. user namespaces

- Apptainer-setuid : mode historique
  - `setuid` : permission permettant à un utilisateur de lancer un programme avec les droits du propriétaire de celui-ci
  - Rend Apptainer moins sécurisé
  - Quelques fonctionnalités en plus (chiffrement) et moins d'overhead
- User namespaces
  - Fonctionnalité kernel permettant la translation entre les users
  - Root n'est plus du tout impliqué et aucune escalade de privilége n'est possible
  - L'image est convertie en sandbox pendant le runtime : overhead
  - Mode conseillé par WLCG

- Paquets disponibles (RPM, .deb)
- Il est conseillé de l'installer sur tous les nœuds de la ferme (pas sur un espace partagé)

- Installation en mode *unprivileged user namespaces* :

```
yum install apptainer
echo 10000 > /proc/sys/user/max_user_namespaces
```

- Installation avec setud :

```
yum install apptainer-suid
```

Configuration

- `/etc/apptainer.conf` : fichier de configuration
- Fichier bien documenté et documentation en ligne

Overlay / underlay

- L'overlay est la méthode par défaut pour monter les répertoires sur des chemins qui n'existent pas dans le conteneur
  - Utilise FUSE : comportement différent selon les système, nécessite parfois d'être root
- L'underlay est une autre méthode pour créer des points de montage à la volée
  - Le montage est créé dans le répertoire scratch de l'image en cours d'exécution
- Dans la configuration :

```
enable overlay = no

enable underlay = yes
```

Quelques réglages utiles

- `mount home = no` : ne pas monter le home par défaut
- `bind path = /etc/hosts` : rendre accessible un fichier depuis le conteneur. Cette ligne peut être répétée pour plusieurs fichiers.
- `allow container <type> = yes` : autoriser ou non le type de conteneur (sif, dir, etc.)

APPTAINER AU CC-IN2P3

RESSOURCES

CVMFS	Banque d'images fournies par le CC-IN2P3
PBS	Images de l'utilisateur
Gitlab CI	Artifacts des jobs

Les images de CVMFS sont organisés par usage (HPC/HTC, CPU/GPU, …) et sont maintenus par le CC-IN2P3

- Apptainer "version WLCG" installé au niveau système sur les workers
- Autres versions et configurations (setuid/user namespaces) disponibles via `module`
- Pas de root, donc pas de build !
  - Soit builder par Gitlab-CI
  - Soit builder localement et transférer son image
  - Soit utiliser une image déjà prête (par exemple dans CVMFS)
- Apptainer accessible via les schedulers batch (Condor et Slurm)

POUR FINIR

Sources & liens utiles

- Apptainer documention version 1.1
- CernVM-FS with Apptainer/Singularity: A Great Combination, présentation de D. Dykstra (Fermilab)
- A Practical Introduction to Container Terminology, redhat.com
- Repo git contenant cette présentation et les exercices

QUESTIONS ?

- Prérequis : SSH et connaissances en Linux
- Chaque participant aura une VM hébergée au CC à disposition (connection en SSH)
- Deux exercices
- Les données pour les exercices sont déjà dans les VM
- Apptainer n'est pas installé (ni configuré), mais vous avez les droits root (via `sudo`)