

Utilisation Singularity sur Jean Zay

Premiers préparatifs liés à Jean-Zay

Doc Jean-Zay sur Singularity: <http://www.idris.fr/jean-zay/cpu/jean-zay-utilisation-singularity.html>

De préférence sur Jean Zay, un utilisateur va amener sur l'infrastructure son image Singularity / image .sif déjà prête à l'emploi (en faisant une copie du fichier .sif dans l'un des répertoires de Jean Zay, par exemple dans le \$WORK ou dans le \$SCRATCH). [WIP: Il y a la possibilité de créer une image .sif directement sur Jean Zay à partir d'une image docker se trouvant sur le docker hub mais cela sera abordé dans une autre partie]

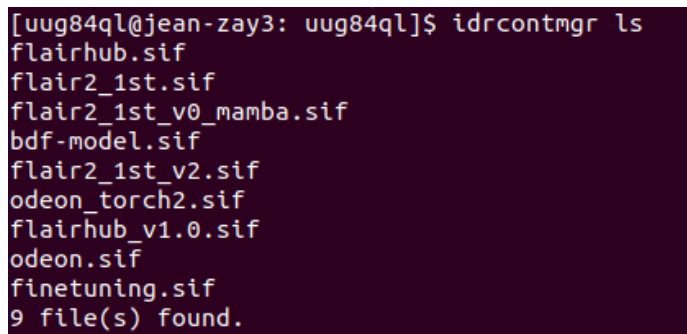
Une fois l'image .sif copiée sur Jean-Zay, il faut maintenant suivre les consignes d'utilisation mise en œuvre sur le cluster. Les images .sif doivent être enregistrer dans le répertoire "cache" \$SINGULARITY_ALLOWED_DIR de Jean-Zay avec la commande:

"> idrcontmgr cp /chemin/vers/mon_image.sif" (cela prends un peu de temps)

Ce répertoire cache n'est accessible en écriture que via la commande idrcontmgr.

Une fois copiée le cache, on peut vérifier qu'elle est bien utilisable avec la commande:

"> idrcontmgr ls"



```
[uug84ql@jean-zay3: uug84ql]$ idrcontmgr ls
flairhub.sif
flair2_1st.sif
flair2_1st_v0_mamba.sif
bdf-model.sif
flair2_1st_v2.sif
odeon_torch2.sif
flairhub_v1.0.sif
odeon.sif
finetuning.sif
9 file(s) found.
```

Si on veut effacer une image sif existante, il faudra utiliser la commande:

"> idrcontmgr rm mon_image.sif"

Pour utiliser une image .sif, on peut soit exécuter via un fichier .slurm et la commande sbatch pour lancer un job soit utiliser l'image en mode interactif. Singularity ne s'exécute que sur les nœuds de calcul et via le chargement du module singularity ("> module load singularity"). [Sur Jean Zay il y a Singularity mais pas Apptainer]

Lancement de job avec Singularity via un fichier .slurm et sbatch

Fichier singularity_gpu.slurm:

```
#!/bin/bash
#SBATCH --job-name=singularity_flair      # nom du job
##SBATCH --partition=gpu_p2              # de-commenter pour la partition gpu_p2
#SBATCH --nodes=2                        # on demande 2 noeuds
#SBATCH --ntasks-per-node=4              # avec 2 tache par noeud (= nombre de GPU ici)
#SBATCH --gres=gpu:4                     # nombre de GPU (1/4 des GPU)
#SBATCH --cpus-per-task=10               # nombre de coeurs CPU par tache (1/4 du noeud 4-GPU)
#SBATCH --hint=nomultithread              # hyperthreading desactive
#SBATCH --time=00:30:00                  # temps maximum d'execution demande (HH:MM:SS)
#SBATCH --qos=qos_gpu-dev
#SBATCH --output=/lustre/fswork/projects/rech/tel/uug84ql/logs/SingularityGPU_debug_2_nodes_4_gpus%j.out
#SBATCH --error=/lustre/fswork/projects/rech/tel/uug84ql/logs/SingularityGPU_debug_2_nodes_4_gpus%j.out
#SBATCH --account=tel@v100

# on se place dans le repertoire de soumission
cd ${SLURM_SUBMIT_DIR}

# nettoyage des modules charges en interactif et herites par default
module purge

# chargement des modules
module load singularity

# echo des commandes lancees
set -x

srun singularity exec --nv \
  --no-mount $HOME \
  --bind /lustre/fsnl/projects/rech/tel/commun/dataset_ocs/FLAIR-INC/./data \
  --bind /lustre/fsnl/worksf/projects/rech/tel/uug84ql/FLAIR_INC_BASELINES/./output:rw \
  --bind /lustre/fsnl/worksf/projects/rech/tel/uug84ql/flair_debug:/app \
  --bind /lustre/fswork/projects/rech/tel/uug84ql/debug_csvs:/csvs \
  --bind /lustre/fsnl/worksf/projects/rech/tel/uug84ql/.cache/torch:/torch_home \
  --bind /lustre/fsnl/projects/rech/tel/uug84ql/torch_tmp:/tmpdir \
  --env TORCH_HOME="/torch_home" \
  --env TMPDIR="/tmpdir" \
  $SINGULARITY_ALLOWED_DIR/flairhub_v1.0.sif \
  mamba run -n flairhub python /app/src/flair_inc/main.py --conf_folder /app/configs/debug_2_nodes_4_gpus
```

Pour lancer un job via sbatch et un fichier .slurm il faut utiliser un fichier slurm comme ci-dessus.

La partie avec les arguments #SBATCH ne change pas des fichiers .slurm habituels.

Les premières commandes `cd ${SLURM_SUBMIT_DIR}` et `module load singularity` sont obligatoires pour le fonctionnement mais n'ont pas particulièrement d'intérêt. Une fois cela fait, on utilise la commande `srun` pour lancer la commande que l'on veut exécuter sur le(s) noeud(s) de calcul.

Ensuite utiliser la commande: "singularity exec " permet d'exécuter une application.

Les arguments sont ensuite utilisés:

- nv permet au conteneur d'utiliser les gpus nvidias
- no-mount \$HOME permet de ne pas monter le répertoire \$HOME de l'utilisateur afin d'éviter des conflits entre le conda/mamba se trouvant dans l'image .sif et celui se trouvant sur le nœud de calcul.
- bind permet de monter des répertoires de la machine hôte dans le conteneur, par exemple cela donnera "--bind repertoire/en/local: /repertoire/dans/le/conteneur
- env définit les variables d'environnement que l'on souhaite définir à l'intérieur du conteneur.

L'argument suivant est le chemin vers l'image .sif ici:

`$SINGULARITY_ALLOWED_DIR/mon_image.sif` avec `$SINGULARITY_ALLOWED_DIR` le chemin vers le répertoire où la commande `idrcontmgr` stock les images.

Enfin les derniers arguments après le nom de l'image sont les arguments composants la commande que l'on souhaite exécuter au sein du conteneur.

Les répertoires montés ici dans le cas de flairhub:

- /data : répertoire où les données sont entreposés
- /output: répertoire où les sorties seront générés (l'argument rw permet de forcer l'écriture et la lecture)
- /app: répertoire où se situe le code à exécuter
- /csvs: répertoire où se trouve les csvs (pourrait ne pas être nécessaire en fonction du code)
- /torch_tmp: répertoire où les poids des modèles sont stocké
- /tmpdir: répertoire où les fichiers temporaires propres à torch sont stockés

Prendre le slurm existant et changer les chemins des répertoires montés, chaque répertoire correspondant à un répertoire cible que le code va attendre. Changer aussi les fichiers de config correspondant par exemple: [penser à adapter les csvs si besoins]

Exemple de fichier de config modifié:

```
paths :
  out_folder: '/output'
  out_model_name: 'test_flairhub_10k_singularity_1_node_4_gpus'
  path_dataset: /data
  train_csv: '/csvs/TRAIN_FLAIR-INC.csv'
  val_csv: '/csvs/VALID_FLAIR-INC.csv'
  test_csv: '/csvs/TEST_FLAIR-INC.csv'
  global_mtd_folder: '/data/GLOBAL_ALL_MTD/'
```

Une fois le fichier .slurm constitué et les fichiers de configs adaptés il suffit de lancer la commande:

```
$ sbatch singularity_gpu.slurm
```

WARNING: par défaut avec Singularity le conteneur voit tout le FileSystem de la machine hôte, cela veut dire que les montages avec l'argument --bind (et modifs chemins dans les configs) ne sont pas forcément nécessaires mais cela n'est vraiment pas une bonne pratique et peut engendrer de potentiellement erreurs.

Utilisation d'un conteneur en mode interactif

Pour utiliser un conteneur en mode interactif, il faut tout d'abord se placer sur un nœud de calcul en mode interactif grâce à la commande **srun**. Dans le cas de flairhub sur le projet tel cela donne:

```
">srun --pty --nodes=1 --ntasks-per-node=1 --cpus-per-task=12 --gres=gpu:1 --hint=nomultithread --account=tel@v100 bash" [Les args sont mêmes que ceux dans la partie #SBATCH du fichier slurm]
```

Une fois le noeud alloué, l'utilisateur arrive sur un terminal du type: "> bash-5.1\$"

Il faut maintenant charger le module singularity puis se placer soit dans \$WORK soit dans \$SCRATCH et lancer la commande singularity souhaité, ici bash pour permettre de simplement lancer un terminal à l'intérieur du conteneur:

```
"> cd $WORK
```

```
> module load singularity
```

```
> singularity exec --nv --no-mount $HOME $SINGULARITY_ALLOWED_DIR/image.sif bash
```

(une fois à l'intérieur du conteneur en mode interactif il faut encore activer l'environnement virtuel)

```
> conda activate mon_env
```

```
> faire ce que l'on souhaite ...
```

[ici dans la cmd singularity on aurait pu mettre aussi des variables d'environnement et monter des répertoires comme dans l'exemple avec le fichier .slurm]

Informations complémentaires

- Le répertoire qui va contenir de façon temporaire le filesystem du container lors de son exécution sera créé sous l'emplacement valorisé par `$SINGULARITY_CACHEDIR`. Une fois le module chargé, l'emplacement du cache personnel Singularity associé à la variable d'environnement `$SINGULARITY_CACHEDIR` est positionné automatiquement dans l'espace disque `$SCRATCH`. Il est donc fortement recommandé de ne pas modifier la variable d'environnement `$SINGULARITY_CACHEDIR` afin de ne pas dépasser les quotas des autres espaces disques et d'y en simplifier le nettoyage automatique.
- Dans la documentation il est dit que le code est exécutable en parallèle en passant les argument: `--mp=pmix_v2` et `--mp=pmix_v3` mais cela ne fonctionne pas ..

Écrire une partie sur l'utilisation du code `flair_docker` et mettre dedans des exemples de fichiers `slurms` et idem un exemple de fichiers de configs

Création d'une image Singularity

Création d'image:

Installation préalable:

Il faut installer

Passage d'une image docker locale en image singularity .sif:

> `singularity build image_docker.sif docker-daemon://image_docker:tag`

par exemple: `singularity build flairhub.sif docker-daemon://flairhub:v0.0.1`

La conversion d'image docker en image sif ce fait depuis une machine locale, sinon sur Jean-Zay on peut créer une image Singularity depuis le docker hub ou depuis le SingularityHub

Différences entre Singularity/Apptainer et Docker

1. Public cible et cas d'usage

Critère	Docker	Singularity/Apptainer
Environnement	Cloud, DevOps, déploiement	HPC (clusters), calcul scientifique
Utilisateurs	Développeurs, ingénieurs DevOps	Chercheurs, bioinformaticiens
Permissions	Nécessite des droits root (sudo)	Fonctionne sans droits root

2. Architecture technique

Critère	Docker	Singularity/Apptainer
Démon	Nécessite un démon (dockerd)	Aucun démon (exécution directe)

Isolation	Forte isolation (namespaces)	Moins isolé, intégré au système hôte
Images	Format .dockerfile → .tar	Format .sif (Singularity Image File)
Portabilité	Optimisé pour le cloud	Optimisé pour les clusters HPC

3. Gestion des images

Critère	Docker	Singularity/Apptainer
Registry	Docker Hub, Google Container Registry	Peut importer depuis Docker Hub (docker://)
Taille des images	Souvent volumineuses (couches superposées)	Images plus compactes (squashFS)
Modification	Possible via docker commit	Images immuables (sauf en mode --writable)

4. Sécurité et permissions

Critère	Docker	Singularity/Apptainer
Root par défaut	Conteneurs tournent souvent en root	Exécution avec les permissions de l'utilisateur hôte
GPU/MPI	Nécessite --gpus all + NVIDIA Container Toolkit	Support natif via --nv ou --rocm
Réseau	Isolation réseau avancée (bridge, host)	Partage le réseau de l'hôte

5. Intégration avec les systèmes HPC

Critère	Docker	Singularity/Apptainer
Schedulers	Peu compatible (Slurm, PBS)	Intégration native (Slurm, PBS, etc.)
Stockage parallèle	Difficulté avec Lustre/GPFS	Support natif des systèmes de stockage HPC
Performance	Surcharge due à l'isolation	Proche des performances natives

TODO

Refaire tourner le pipeline entier docker build / docker run / singularity build / run

Ecrire un script permettant de faire une image sif depuis un fichier .yaml
Mettre à jour le repo flair_docker

Regarder les pistes suivantes:

faire une installation direct dans le container et non via conda

faire une install poetry ?

Sur jzay regarder comment avoir la queue de pre processing afin de pouvoir faire un build complet

Retenter de faire marcher le code depuis une image sif buildé depuis une image docker existante et non pas provenant du hub

Retenter sur jean zay de faire un build d'une image provenant du docker hub

- Je n'arrive pas à faire un build d'image sif depuis une image docker sur jzay
 - j'obtiens une erreur (FATAL: While performing build: while creating squashfs: create command failed: signal: killed:)
 - d'après la doc il vaut mieux aller sur la queue de preprocessing, est-ce qu'il a réussi à y aller?

Mettre dans le README.md de flair_docker la doc d'utilisation

regarder un tutoriel sur le principe des runscript dans singularity

Pousser mon image mon sur docker hub puis sur le hub de singularity

Faire des images .sif intermédiaires avec ubuntu / conda puis installer juste les libs nécessaires

Mettre en place un client github automatisant l'enregistrement ids/mdp github avec des clés ssh dans l'interface github

Prendre le temps de faire un branchement entre VScode et git

Voir si avec les utilitaires apptainer présent sur jean zay je peux essayer de builder quelque chose dessus

Réfléchir davantage à la gestion des paramètres passer à la ligne de cmd python

Faire une meilleur gestion des entry point / passage de script au container si possible (permettant d'avoir un entry point pour le script et l'autre avec fish)