Miyabiにおける Pythonとコンテナ

JCAHPC Open Hackathon
2025年2月3日（月）

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

Python環境

- Python公式版 (The Python Package Index(PyPI)) https://pypi.org
  - pip コマンド
  - インストール時にソースからビルドするものが多い
  - pyenv, venvを使って環境切り替えもできるが、以下のminiforgeがお勧め
- Miniforgeによるもの https://github.com/conda-forge/miniforge
  - conda コマンド
  - PyPIとは異なる独自ビルド、バイナリで提供
  - PyPIとの共存も可能
  - 大量のパッケージが連携するような大規模な物(機械学習など)は、こちらがおすすめ
    - Miyabi-CではIntelなどによる独自ビルドも使用可能
- https://www.python.jp/install/docs/install_plan.html

(Miyabi-C) Intelビルド版Pythonの使用

- 以前と使い方が変わっているので注意
  - 以前は conda -c intel で指定できたが、、
- チャンネル名として以下を指定する必要

```
https://software.repos.intel.com/python/conda/
```

- 例えばこんな感じ

```
$ conda create -n numpy_mkl1.26.4 ¥
  -c https://software.repos.intel.com/python/conda/ ¥
  numpy=1.26.4=py311h689b997_3
```

  - 依存関係は指定したチャンネルから、それ以外デフォルトでは conda-forgeから持ってくる

Miyabi-Gでのminiconda環境

- 以下のようにして有効化できるようにしています。(moduleのminicondaとは別)
  - チャンネルは conda-forge, nvidia を使用

```
$ source /work/share/setup-miniforge_aarch64.sh
Currently Loaded Modulefiles:
 1) gcc/11.4.1   2) cuda/12.6   3) nccl/2.23.4   4) cudnn/9.5.1.17   5) hpcx/2.18.1
…
# conda environments:
#
base                  * /work/share/miniforge3-aarch64
deepspeed               /work/share/miniforge3-aarch64/envs/deepspeed
PyTorch-2.5.1           /work/share/miniforge3-aarch64/envs/pytorch-2.5.1
$ conda activate deepspeed
```

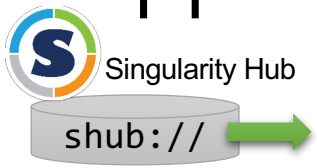
Miyabiで使えるコンテナ仮想化

- いずれもDockerコンテナとの互換性あり
- Apptainer
  - 開発元はLinux Foundation
  - 旧 singularity Community Edition, SyLab
  - Dockerとは少し使い勝手が違う
- Enroot
  - 開発元はNVIDIA
  - Dockerとかなり使い方が似ている

Apptainer

- スパコン向けのコンテナソフトウェアとして使われている
  - かつてはSingularityとして開発、Lawrence Berkeley国立研究所で開発開始
  - 商用版は Singularity Pro として現在も販売中
  - Singularity Community Edition ➔ Linux Foundationに移管され Apptainerに
    - Miyabiにはsingularityもインストールされているがapptainerを推奨
  - https://apptainer.org/docs/user/latest/
- Dockerコンテナとの高い互換性
  - 一部、管理者権限の問題で動かない場合もあるが、ほとんどの場合OK
  - アーキテクチャごとに異なるイメージが必要
    - Miyabi-G: aarch64, Miyabi-C: x86_64
    - DockerHubには Miyabi-G (aarch64)向けが用意されていない場合も多い

Apptainer: コンテナ仮想化



Singularity recipe

コンテナイメージ
ファイル

Production向け (Read only)

```
$ singularity build
container.img <source>
```

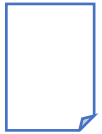
環境構築 (writable image)

```
$ sudo singularity build
--writable container.img
<source>
```

環境構築 (sandbox)

```
$ sudo singularity build
--sandbox container.img
<source>
```

コンテナイメージ
ファイル



container.img

コンテナ環境内で実行 (shell)

```
$ singularity shell container.img
> python mnist.py
```

コンテナ環境のコマンドで実行(exec)

```
$ singularity exec container.img ¥
python mnist.py
```

定義された通りに実行(run)

```
$ singularity run container.img
or
$ ./container.img
```

MiyabiにおけるApptainer：準備

- 共通

```
$ module load apptainer
$ module list
Currently Loaded Modulefiles:
 1) squashfuse/0.5.2  2) gocryptfs/2.4.0  3) apptainer/1.3.5
```

- コンテナイメージ置き場
  - /work/share/Container-{G,C}/apptainer
  - コンテナファイル(フルパス)を以下 $SIF で表すことにする

```
$ export SIF=/work/share/ContainerImages-G/apptainer/pytorch-ngc-25.01.sif
```

- ファイルサイズが大きいので、みなさんは実行はしないでください：

```
$ apptainer build $SIF docker://nvcr.io/nvidia/pytorch:25.01-py3
```

Miyabi-GにおけるApptainer: 使い方(1/2)

- ログインノードの場合

```
$ apptainer exec $SIF python -c "import torch;
print(torch.__version__)"
2.6.0a0+ecf3bae40a.nv25.01
```

- 計算ノードの場合： nvを忘れるとGPUを使ってくれないので注意!!

```
$ apptainer exec --nv $SIF python -c "import
torch;print(torch.__version__,¥
torch.version.cuda, torch.cuda.get_device_capability())"
2.6.0a0+ecf3bae40a.nv25.01 12.8 (9, 0)
```

- 要は、通常のPython実行にapptainer以降の下線部がprefixとして付くと思えば良い

Miyabi-GにおけるApptainer: 使い方(2/2)

- 大抵の場合、ファイルの書き換え、コンテナ外部のやり取りが必要➔どうする？
- -Bオプションでbind、コンテナの内外が繋がる
  - コンテナ内の出力ファイルをそのままcurrentに置きたければ
    -B `pwd`
  - コンテナ中 /workspace のファイルを細工したい場合

```
$ mkdir tmp
$ apptainer exec $SIF -B `pwd` cp -r /workspace/ ./tmp
$ ls tmp
/workspace
```

- Overlayの利用
  - Overlayファイルに記録される

```
$ apptainer overlay create --size 1024 --create-dir /workspace ./tmp-ext3_overlay.img
$ apptainer exec --nv –o ./tmp-ext3_overlay.img $SIF python exp.py
```

Enroot

- 従来のコンテナ/OSイメージを特権のないsandboxに変える、シンプルかつ強力なツール
  - どちらかというとsandbox化して使うことにフォーカス
- 以下、あくまでも例、ご自分のQuotaが消費されるのでみなさんは実行しないでください

```
$ enroot import ‘docker://$oauthtoken@nvcr.io#nvidia/cuda:10.0-base’  # コンテナのインポート

$ enroot create --name cuda nvidia+cuda+10.0-base.sqsh # コンテナ作成

$ enroot list
cuda

$ enroot start --root --rw cuda sh -c ‘apt update && apt install -y cuda-samples-10.0’ # コンテナ実行

$ enroot start --rw cuda sh -c 'cd /usr/local/cuda/samples/5_Simulations/nbody && make -j’

$ export ENROOT_MOUNT_HOME=y NVIDIA_DRIVER_CAPABILITIES=all

$ enroot start --env DISPLAY --env NVIDIA_DRIVER_CAPABILITIES --mount /tmp/.X11-unix:/tmp/.X11-unix cuda
¥ /usr/local/cuda/samples/5_Simulations/nbody/nbody

$ enroot remove cuda  #コンテナ削除
```

MiyabiにおけるEnroot：準備

- 共通
  ```
  $ module load enroot
  $ module list
  Currently Loaded Modulefiles:
  $ module list
  Currently Loaded Modulefiles:
   1) squashfuse/0.5.2   2) enroot/3.5.0
  ```
- コンテナイメージ置き場
  - /work/share/Container-{G,C}/enroot
  - コンテナファイル(フルパス)を以下 $SQSH で表すことにする
    ```
    $ export SQSH=/work/share/ContainerImages-G/enroot/pytorch-ngc-25.01.sqsh
    ```
- ファイルサイズが大きいので、みなさんは実行はしないでください：
  ```
  $ enroot import -o $SQSH docker://nvcr.io/nvidia/pytorch:25.01-py3
  ```
  - NGCコンテナのcredentialが必要で、然るべき設定をしないといけない…後述

Miyabi-GにおけるEnroot: 使い方

- Sandbox化する前提なら

```
$  enroot create --name pytorch2.6 $SQSH
    # unsquashするので ~/.local/enroot に全コピーが生成される
$ enroot list
pytorch2.6
```

  - （計算ノードでないとエラーになる）

```
$ enroot start pytorch2.6 sh -c 'python -c "import torch;print(torch.__version__,¥
torch.version.cuda, torch.cuda.get_device_capability())"'
2.6.0a0+ecf3bae40a.nv25.01 12.8 (9, 0)
```

- （Sandbox化しないなら）いきなりsqshファイルを渡すこともできる

```
$ enroot start $SQSH sh -c 'python -c "import torch;print(torch.__version__,¥
torch.version.cuda, torch.cuda.get_device_capability())"'
2.6.0a0+ecf3bae40a.nv25.01 12.8 (9, 0)
```