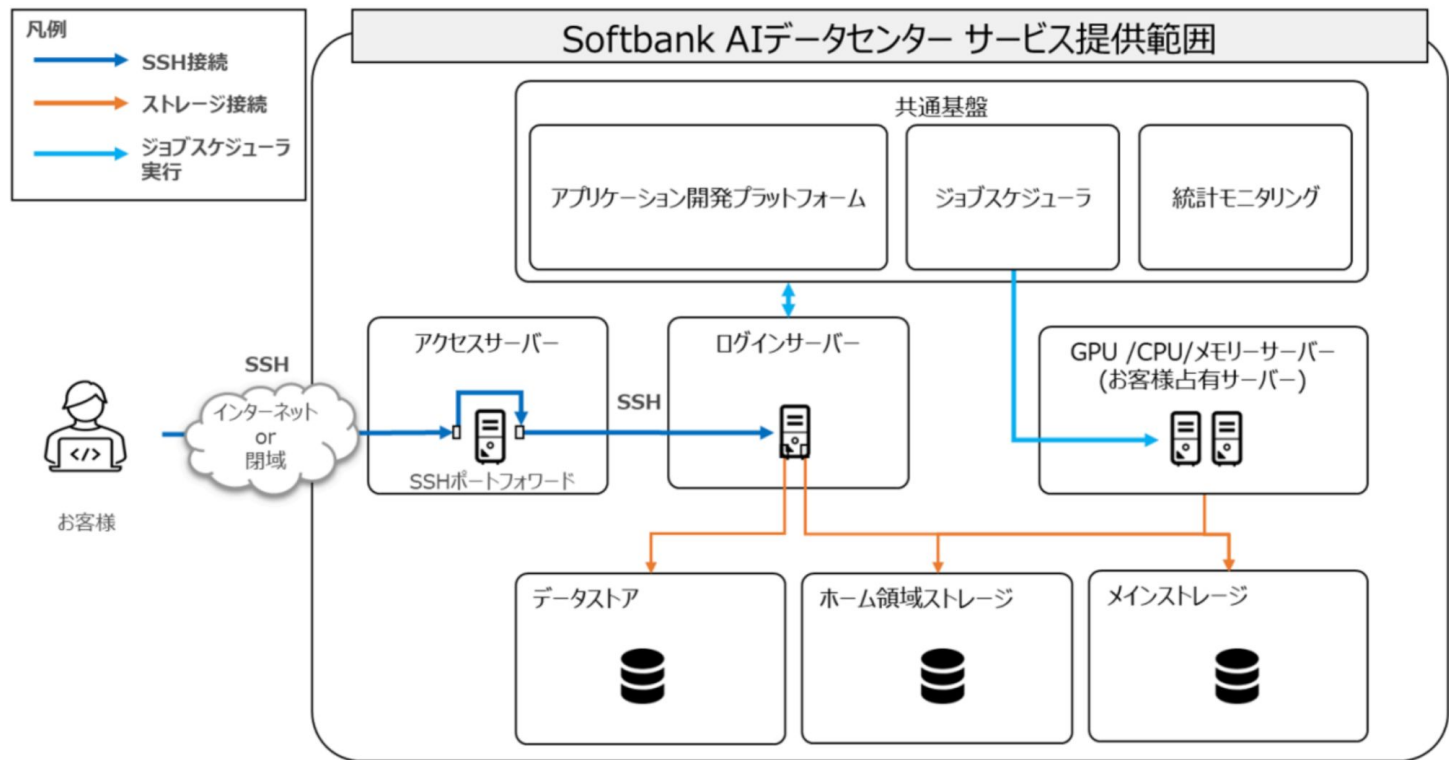


第17回LLMコンソーシアム勉強会 SuperPod活用事例の紹介

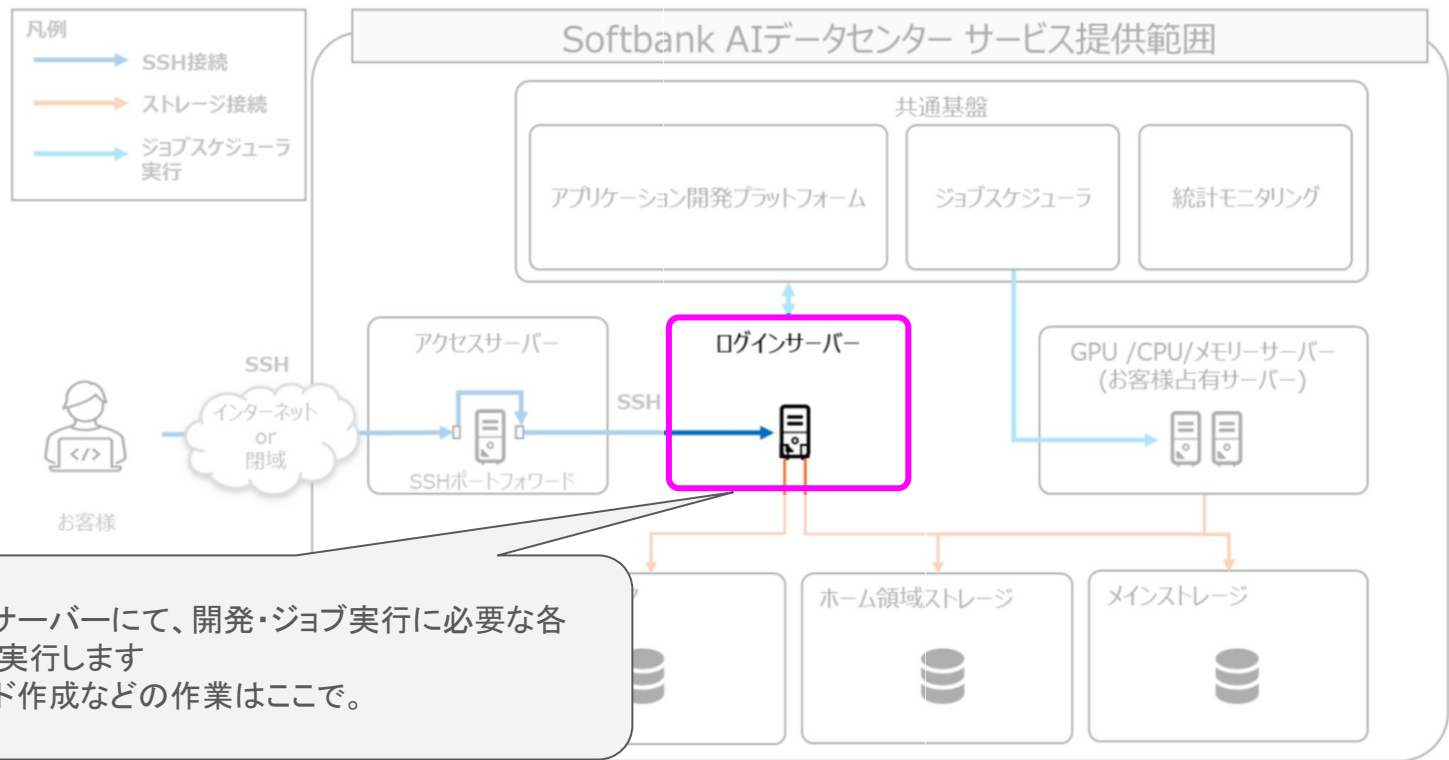
BeyondAI推進課 竹田康亮



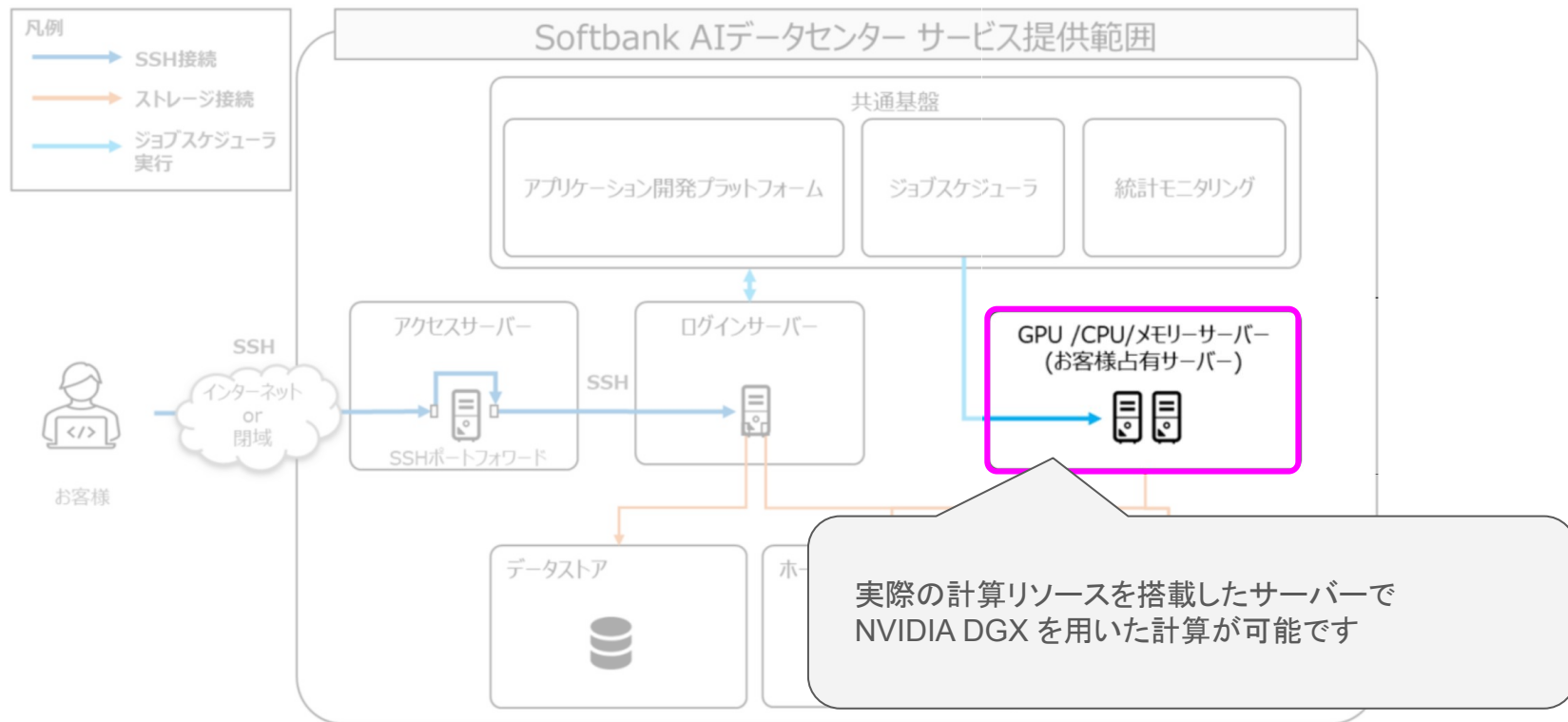
- SuperPOD 上での計算方法の概要について
 - アクセス方法
 - Docker コンテナの利用方法
 - Slurm の利用方法
 - 計算実行の概要



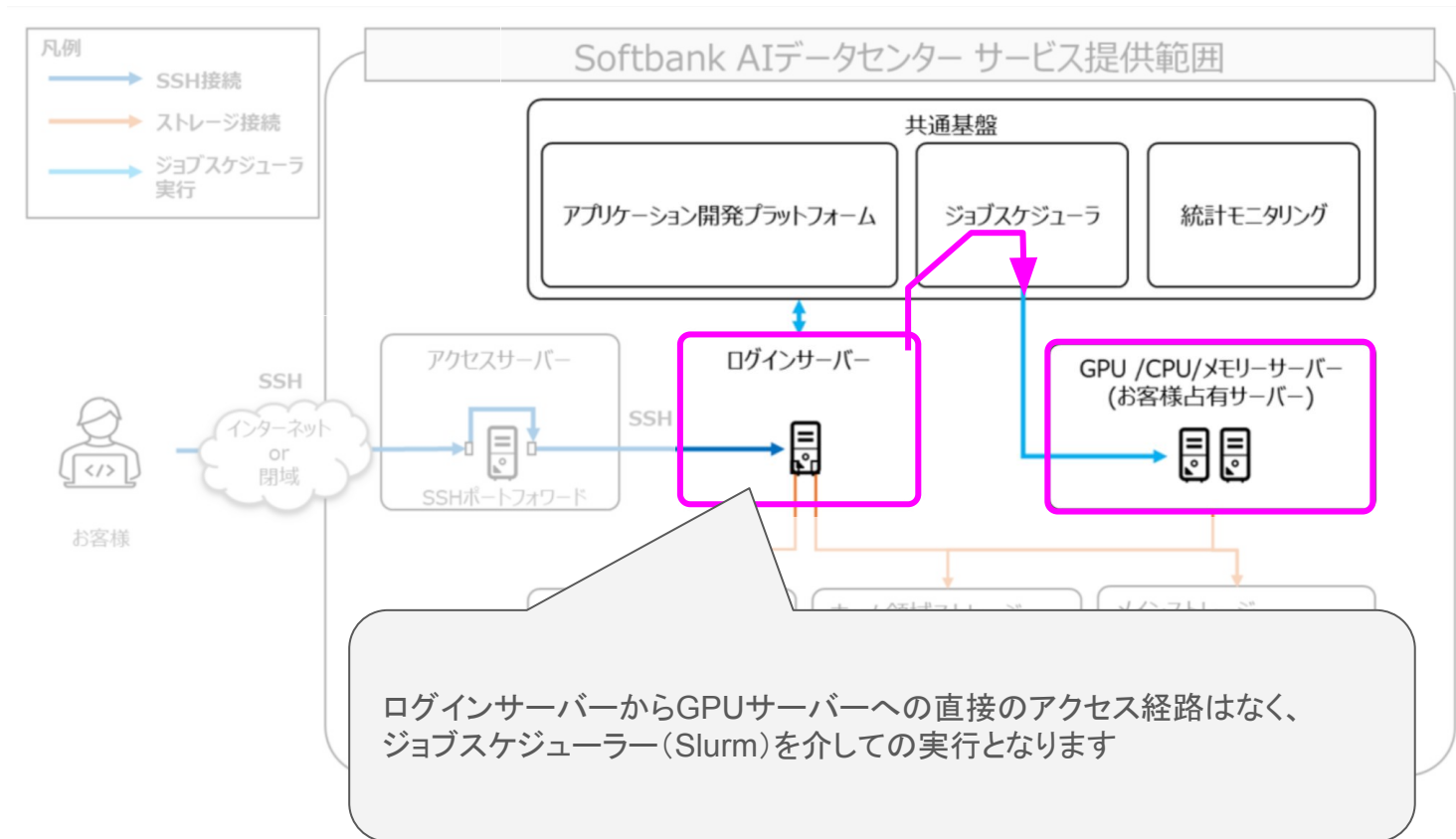
SoftBank AI データセンターご利用ガイド v1.2 より



SoftBank AI データセンターご利用ガイド v1.2 より



SoftBank AI データセンターご利用ガイド v1.2 より

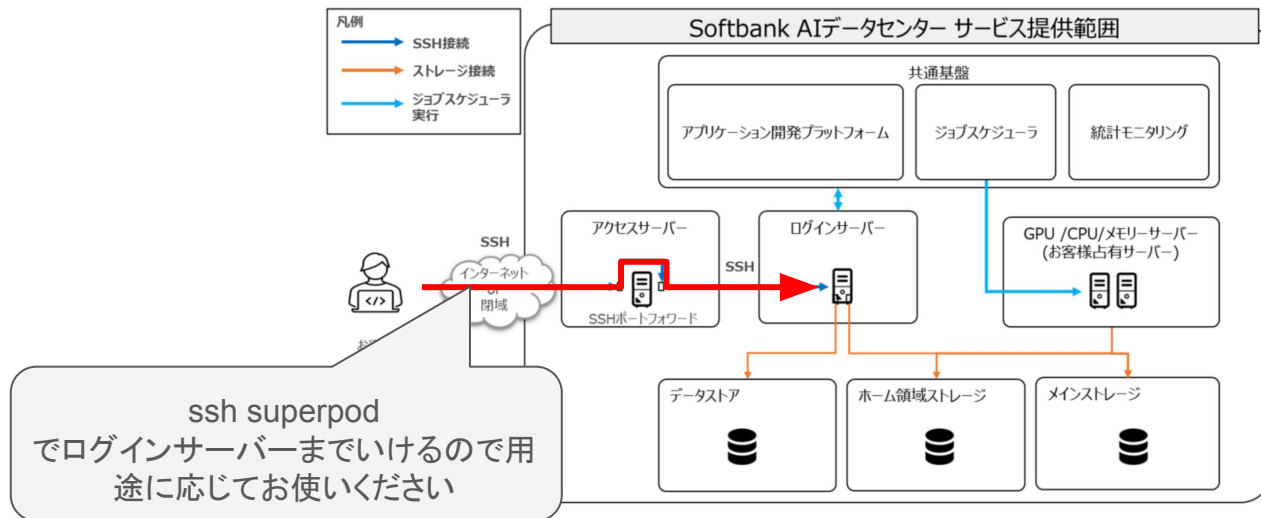


1. SuperPOD へのアクセス
2. Docker 利用について
3. Slurm 利用について

1. SuperPOD へのアクセス
2. Docker 利用について
3. Slurm 利用について

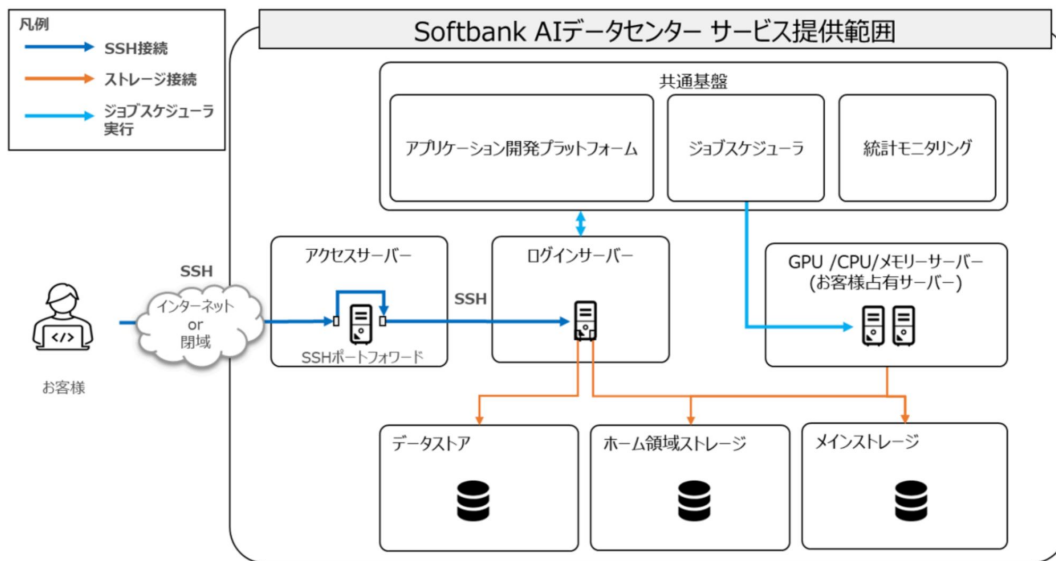
- .ssh/config の設定について
 - アクセスサーバー（踏み台）を介してログインサーバーにアクセスするための設定

```
Host superpod
  HostName 10.XXX.YY.ZZ  ← ログインサーバーのIPアドレス
  User user330XX         ← ログインサーバーのユーザー名
  IdentityFile ~/.ssh/superpod_ssh_key
  ProxyCommand ssh -i ~/.ssh/superpod_ssh_key -p 50024 -W %h:%p user330XX@210.AAA.BB.CC ← アクセスサーバーの情報
```

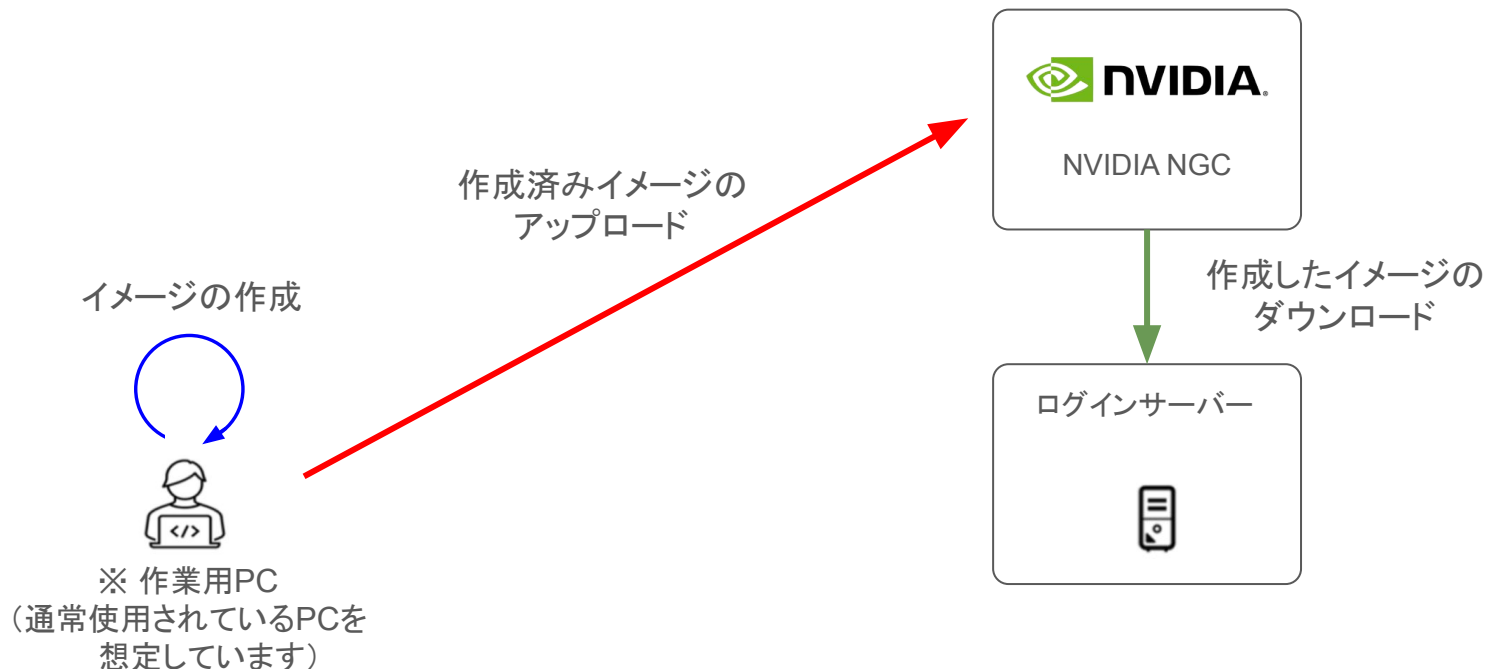


1. SuperPOD へのアクセス
2. Docker 利用について
3. Slurm 利用について

- 開発環境の構築には Docker を利用する必要があります
 - apt, pip 等のコマンドはサーバー上で直接実行できないため、
Docker コンテナの仮想環境を使用する必要があります



- SuperPOD でDocker利用のための前準備について、本手順で説明します
 - → : イメージの作成
 - → : イメージのアップロード
 - → : イメージのダウンロード



1. SuperPOD へのアクセス
2. Docker 利用について
 - a. イメージの作成
 - b. イメージのアップロード
 - c. イメージのダウンロード
 - d. コンテナの利用
3. Slurm 利用について

- 任意のディレクトリに Dockerfile （=設定用テキストファイル）を作成
 - このDockerfile 内に、こういった仮想環境を作成するかを指定します
 - 下記の内容を Dockerfile というファイル名で保存して下さい
 - GPUを用いた解析を想定したサンプルです

```
FROM nvidia/cuda:12.9.1-cudnn-devel-ubuntu20.04

RUN apt-get update && \
    apt-get install -y python3 python3-pip git && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

RUN python3 -m pip install --upgrade pip

RUN pip install torch torchvision torchaudio

RUN pip install numpy matplotlib scikit-learn

WORKDIR /workspace

CMD ["bash"]
```

今回は [nvidia/cuda](#) からベースとするイメージを選択しました

解析ニーズに合わせて、python用ライブラリは適宜修正してお使い下さい

コンテナを実行すると /workspace というディレクトリに自動的に移動します。こちらも適宜修正・削除してお使い下さい

- 実行コマンド

```
$ docker build --platform=linux/amd64 -f Dockerfile -t superpod_hello_world .
```

※ 注意点

作業用PCのアーキテクチャが
arm64 の場合にはこのオプションを付けて下さい

作成したファイル名
(任意)

作成するイメージ名
(任意)

- 実行例

```
$ docker build --platform=linux/amd64 -f Dockerfile -t superpod_hello_world .
[+] Building 5.1s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 82B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/1] FROM docker.io/library/ubuntu:latest@sha256:440dcf6a5640b2ae5c77724e68787a906afb8ddee98bf86db94eea8528c2c076
=> => resolve docker.io/library/ubuntu:latest@sha256:440dcf6a5640b2ae5c77724e68787a906afb8ddee98bf86db94eea8528c2c076
=> => sha256:b08e2ff4391ef70ca747960a731d1f21a75febbd86edc403cd1514a099615808 29.72MB / 29.72MB
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:986bae76105d693e4bb6c6b176a55d7bb23ec21b7aa1797156dc8e9bb39b3fc8
=> => exporting config sha256:d9c53e5d00c2058f164b63e05b40e3cfe9821e41b5b644179ad21509799ea84b
=> => exporting attestation manifest sha256:99b7c26ccd48bc3c2ebe6345399e482eaa8288d77c89b93056a315dc3e62c323
=> => exporting manifest list sha256:7268b6c8c52dc249aa49fb44fe71f099024e12f4038d29fb8767d11d30cf5dcb
=> => naming to docker.io/library/superpod_hello_world:latest

1 warning found (use docker --debug to expand):
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 3)
```

- --platform=linux/amd64 を使用しない場合の実行例

○ 実行例

```
$ docker build -f Dockerfile -t superpod_hello_world .
[+] Building 0.8s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 82B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/1] FROM docker.io/library/ubuntu:latest@sha256:440dcf6a5640b2ae5c77724e68787a906afb8ddee98bf86db94eea8528c2c076
=> => resolve docker.io/library/ubuntu:latest@sha256:440dcf6a5640b2ae5c77724e68787a906afb8ddee98bf86db94eea8528c2c076
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:447eb1391c150d9322c3227cb550df422e01af619b75cd3037056bc58dd1080e
=> => exporting config sha256:edacb68c70e7a83bbf7426d7a29ba83430342146a2d9532a7b3a4b4cadbb89f6
=> => exporting attestation manifest sha256:0a5e509f0fae51e2fe275c9c65267a40bd52dcd3878b00995ffa7b2cecd53e0d
=> => exporting manifest list sha256:6ad8758d5c2290d0b04afb04c97e08234a1ecaa9d319e3ad9e8328c90252d65b
=> => naming to docker.io/library/superpod_hello_world:latest
=> => unpacking to docker.io/library/superpod_hello_world:latest

1 warning found (use docker --debug to expand):
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 3)
```

- 作成したコンテナが amd64 に対応していれば ok

```
$ docker image inspect superpod_hello_world --format '{{.Architecture}}'
amd64
```


- アップロードのために作成したイメージ名を変更します

```
$ docker tag superpod_hello_world nvcr.io/z4pymjpdqjsj/sbXXXXXXXX-X/superpod_hello_world:v0.0.0
```

先ほど作成したイメージ名

NVIDIA GPU Cloud (NGC) チーム名

設定したいイメージ名とバージョン番号

1. SuperPOD へのアクセス
2. Docker 利用について
 - a. イメージの作成
 - b. イメージのアップロード**
 - c. イメージのダウンロード
 - d. コンテナの利用
3. Slurm 利用について

前準備：NGC アカウントについて

19

- NVIDIA NGC へ [sing in](#)
 - ※ ご利用ガイドv1.1 p.41～



Portal of enterprise services, software, and support for AI, digital twins, and high-performance computing.

Log In

Enter your email to log in or create a NVIDIA profile.

Email Address

user@domain.com

Continue

By proceeding, you agree to the

[NVIDIA Technology Access Terms of Use](#)

[Login Help](#)



Select Your NVIDIA Cloud Account / Team

Organizations within NVIDIA GPU Cloud allow you to share registries and resources with colleagues.

Select One

Search

Softbank Corp

No Team

sb15106230-1

sb15106230-2

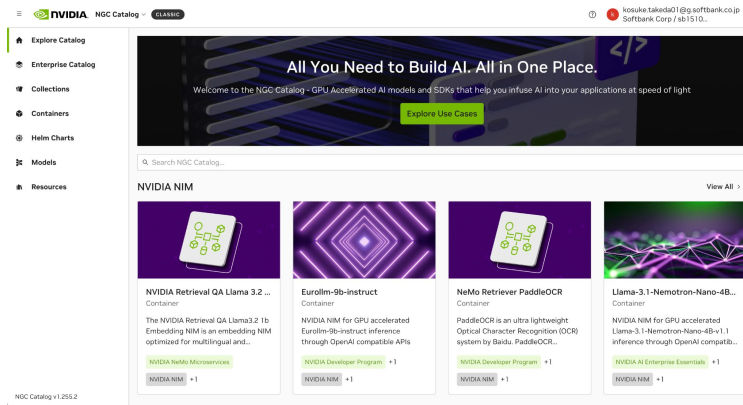
sb15082150

sb15106230-4

moj

khwrpun0cmzh

Continue



ログイン画面

各 NVIDIA NGC チーム名の選択
※ 実際の表示とは異なる場合があります

The screenshot shows the NVIDIA NGC Catalog website. The user profile dropdown menu is open, and the 'Setup' option is highlighted with a red box. An arrow points from the dropdown menu to a separate list of options, which also has the 'Setup' option highlighted with a red box.

NGC Catalog v1.255.2

Explore Catalog

- Enterprise Catalog
- Collections
- Containers
- Helm Charts
- Models
- Resources

All You Need to Build AI. All in One Place.

Welcome to the NGC Catalog - GPU Accelerated AI models and SDKs that help you infuse AI into your applications at speed of light

[Explore Use Cases](#)

Search NGC Catalog...

NVIDIA NIM

[View All >](#)

NVIDIA Retrieval QA Llama 3.2 ...
Container
The NVIDIA Retrieval QA Llama3.2 1b Embedding NIM is an embedding NIM optimized for multilingual and...

[NVIDIA NeMo Microservices](#) +1

Eurolm-9b-instruct
Container
NVIDIA NIM for GPU accelerated Eurolm-9b-instruct inference through OpenAI compatible APIs

[NVIDIA Developer Program](#) +1

NeMo Retriever PaddleOCR
Container
PaddleOCR is an ultra lightweight Optical Character Recognition (OCR) system by Baidu. PaddleOCR...

[NVIDIA Developer Program](#) +1

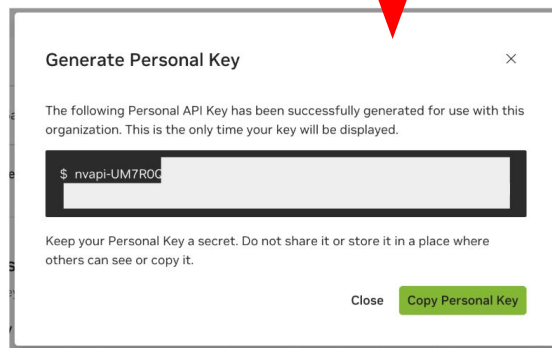
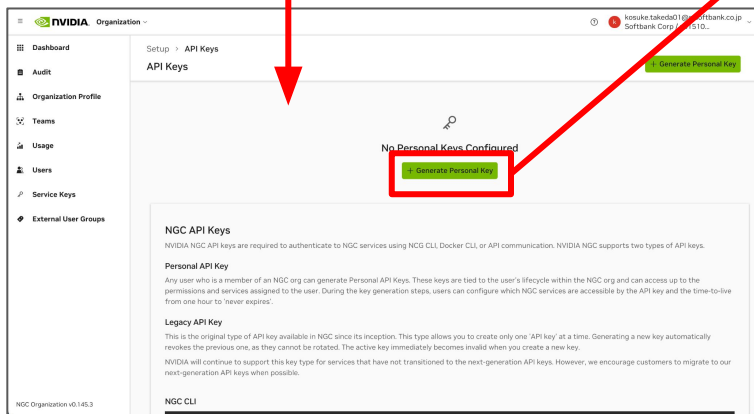
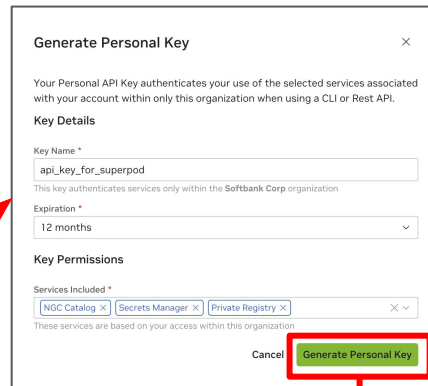
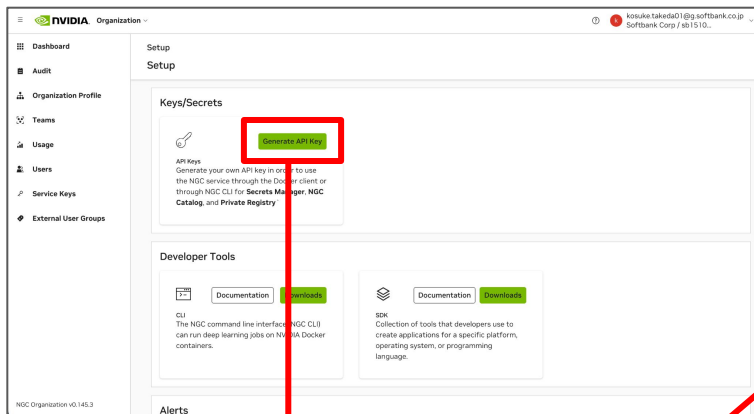
Llama-3.1-Nemotron-Nano-4B...
Container
NVIDIA NIM for GPU accelerated Llama-3.1-Nemotron-Nano-4B-v1.1 inference through OpenAI compatib...

[NVIDIA AI Enterprise Essentials](#) +1

User Profile Dropdown:

- Softbank Corp / sb15106230-2 >
- Contact Admin
- Account Settings
- Setup**
- Organization
- Terms of Use
- Privacy Policy
- Sign Out

- キーの作成、コピーして控えておいて下さい



- NVIDIA コンテナレジストリへログイン
 - アップロードのための認証を実施しておく必要があります

```
$ docker login nvcr.io
```

○ 実行例

```
$ docker login nvcr.io
Authenticating with existing credentials...
Stored credentials invalid or expired
Username ($oauthtoken): $oauthtoken
Password: 
Login Succeeded
```

`$oauthtoken` と記入

作成した API キーをペースト

ログインが成功していればOK

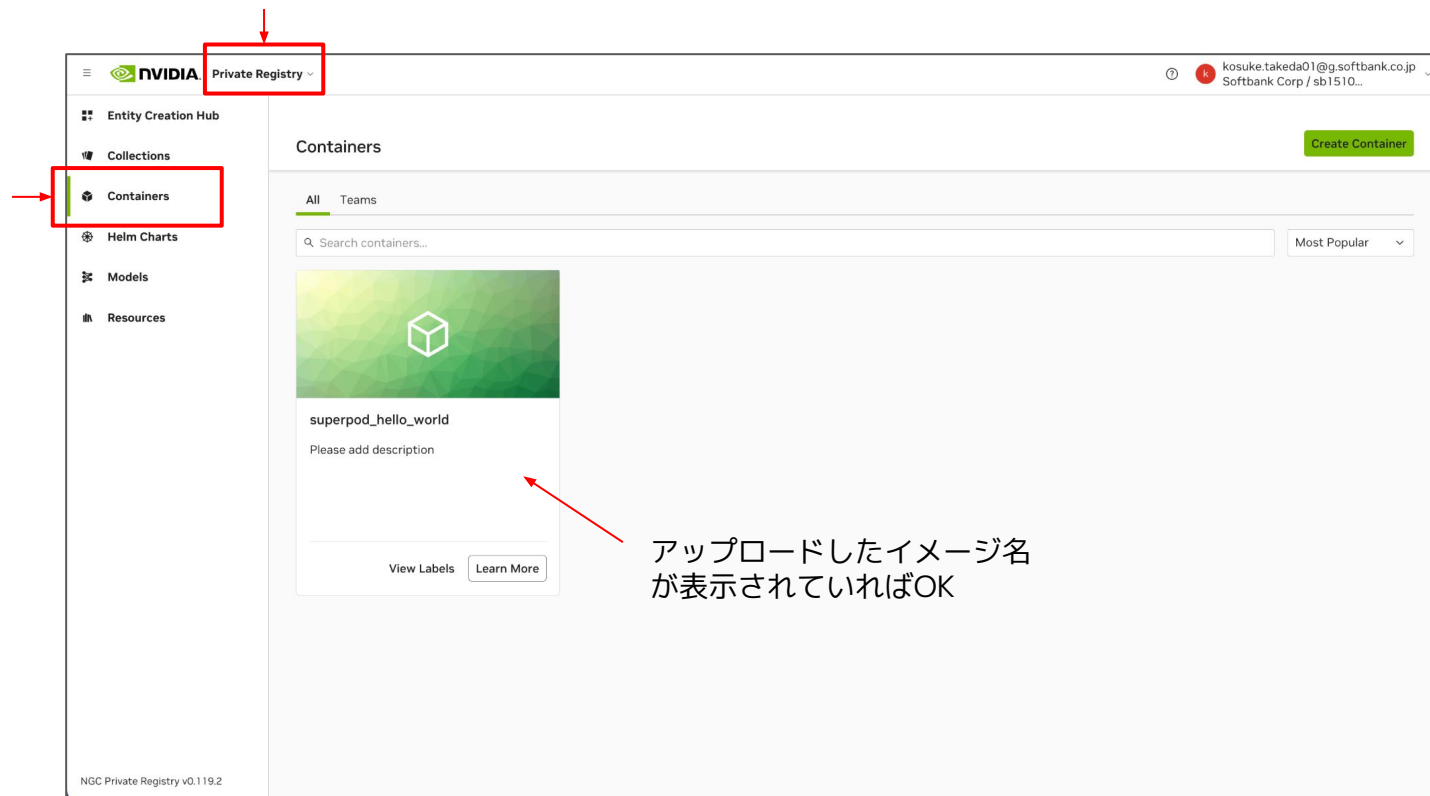
- アップロード

```
$ docker push nvcr.io/z4pymjpdqjsj/sbXXXXXXXX-X/superpod_hello_world:v0.0.0
```

○ 実行例

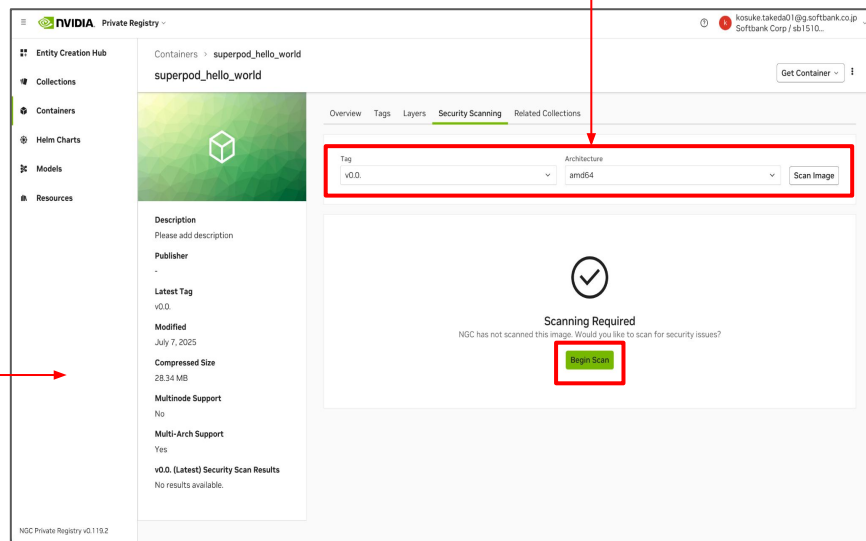
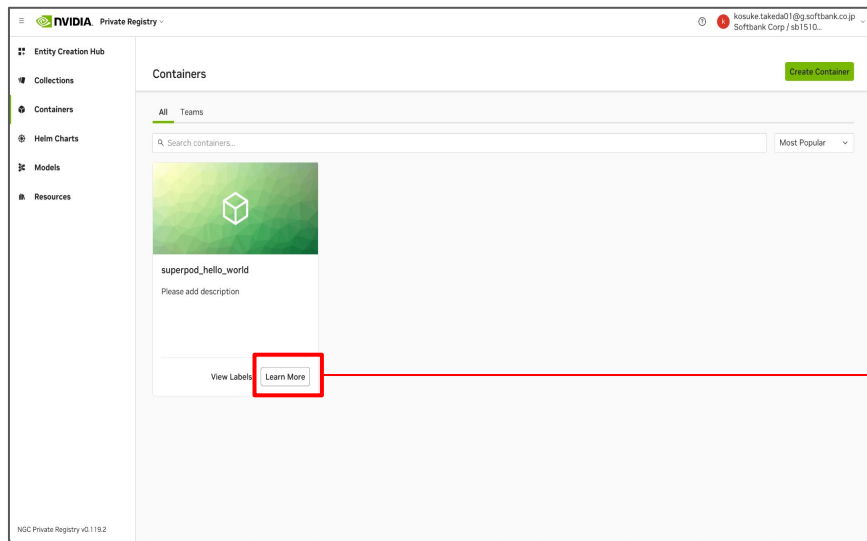
```
$ docker push nvcr.io/z4pymjpdqjsj/sb15106230-2/superpod_hello_world:v0.0.0
The push refers to repository [nvcr.io/z4pymjpdqjsj/sb15106230-2/superpod_hello_world]
207225a435ce: Pushed
3eff7d219313: Pushed
v0.0.0: digest: sha256:6ad8758d5c2290d0b04afb04c97e08234a1ecaa9d319e3ad9e8328c90252d65b size: 855
```

- NGCのページから確認できればアップロード成功です

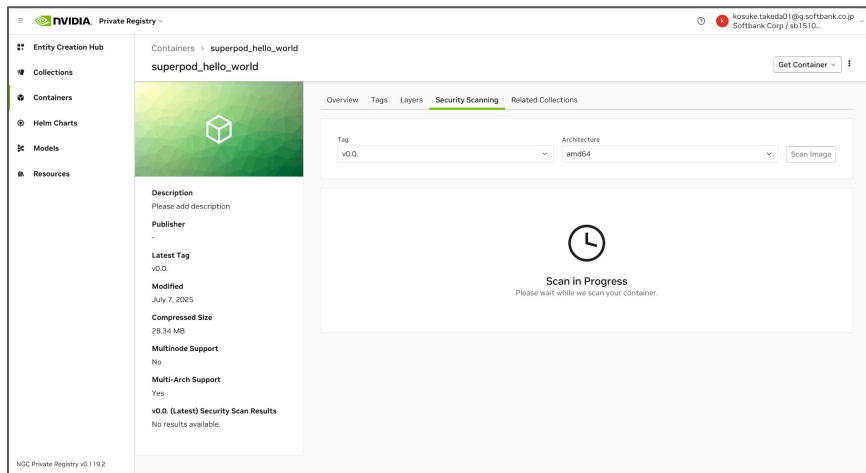


- 該当のコンテナのセキュリティスキャンを実行して下さい

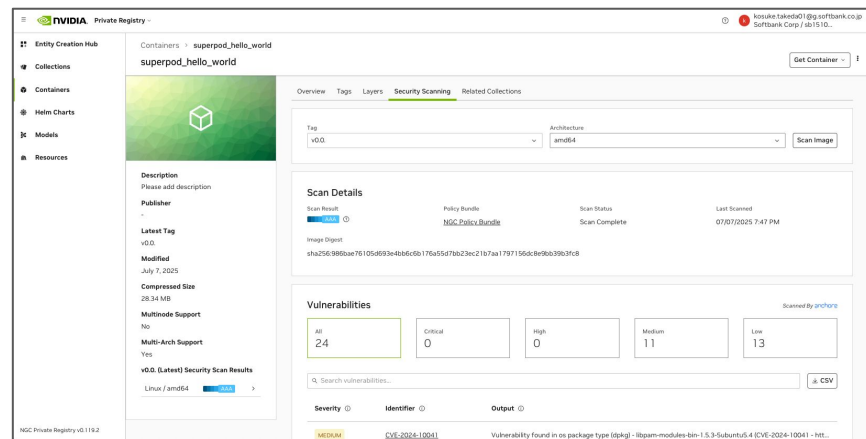
スキャン実行対象としたい
タグを選択して下さい



- スキャンが自動実行・完了となるまでしばらくお待ち下さい

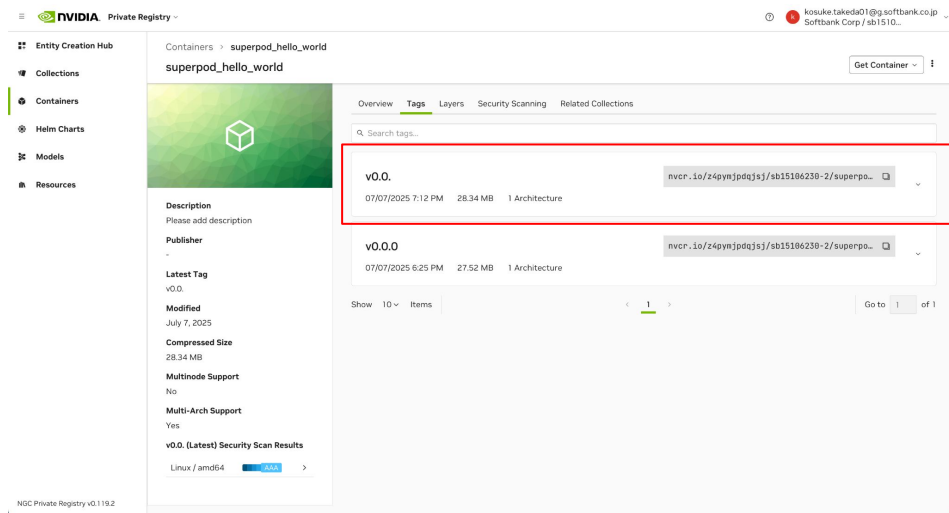


スキャン待ちの画面



スキャン完了後の画面

- ex. 間違ってタグを登録してしまった場合



これを削除したい場合

- ngc コマンドで該当タグを削除できます

```
$ ngc registry image remove nvcr.io/z4pymjpdqjsj/sb15106230-2/superpod_hello_world:v0.0.  
Are you sure you would like to remove z4pymjpdqjsj/sb15106230-2/superpod_hello_world:v0.0.? [y/n]  
Successfully removed image 'z4pymjpdqjsj/sb15106230-2/superpod_hello_world:v0.0.'
```

1. SuperPOD へのアクセス
2. Docker 利用について
 - a. イメージの作成
 - b. イメージのアップロード
 - c. イメージのダウンロード
 - d. コンテナの利用
3. Slurm 利用について

- SuperPOD 上では [enroot コマンド](#) を介して Docker を使用します
 - (※ docker コマンドは直接使用できません)

- 設定ファイルの作成

- フォルダの作成

```
user32002@fcpv00166:~$ mkdir -p ~/.config/enroot
```

- 設定ファイルの作成

```
user32002@fcpv00166:~$ vim ~/.config/enroot/.credentials
```

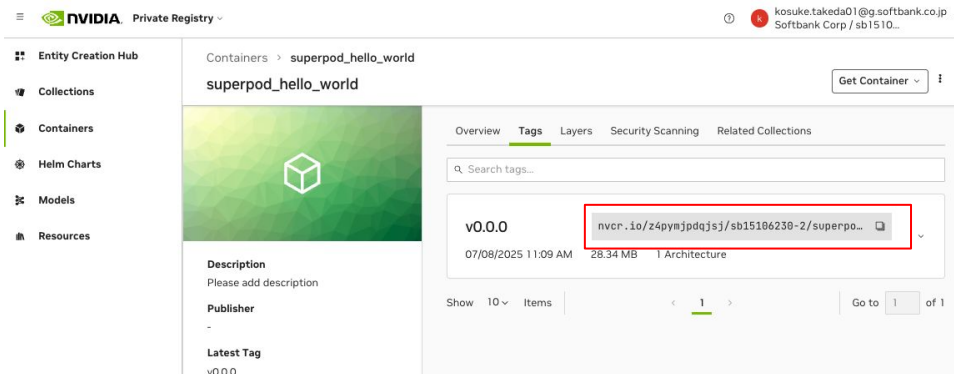
← エディタでファイルを
新規作成して下さい

- .credentials ファイル内容

```
machine nvcr.io login $oauthtoken password <API Key>  
machine authn.nvidia.com login $oauthtoken password <API Key>
```

作成したAPIキーを書き込
んで下さい

- 実行コマンド
 - NVIDIA Private Registry から
該当のイメージ名を下記のように
転記して下さい



ダウンロードしたいイメージ名を記入

```
user32002@fcv00166:~$ enroot import "docker://nvcr.io/z4pymjpdqjsj/sb15106230-2/superpod_hello_world:v0.0.0"
```

```
user32002@fcpv00166:~$ enroot import "docker://nvcr.io/z4pymjpdqjsj/sb15106230-2/pytorch-cuda12.9.1:v0.0.0"
```

```
[INFO] Querying registry for permission grant
```

```
[INFO] Authenticating with user: $sauthtoken
```

```
[INFO] Using credentials from file: /home/user32002/.config/enroot/.credentials
```

正しく設定した credentials が読めているか確認する

```
[INFO] Authentication succeeded
```

```
[INFO] Fetching image manifest list
```

```
[INFO] Fetching image manifest
```

```
[INFO] Downloading 18 missing layers...
```

```
100% 18:0=0s 13b7e930469f6d3575a320709035c6acf6f5485a76abcf03d1b92a64c09c2476
```

```
[INFO] Extracting image layers...
```

```
100% 17:0=0s 13b7e930469f6d3575a320709035c6acf6f5485a76abcf03d1b92a64c09c2476
```

```
[INFO] Converting whiteouts...
```

```
100% 17:0=0s 13b7e930469f6d3575a320709035c6acf6f5485a76abcf03d1b92a64c09c2476
```

```
[INFO] Creating squashfs filesystem...
```

```
Parallel mksquashfs: Using 8 processors
```

```
Creating 4.0 filesystem on /home/user32002/z4pymjpdqjsj/sb15106230-2+pytorch-cuda12.9.1+v0.0.0.sqsh, block size 131072.
```

```
[=====/] 187552/187552 100%
```

```
Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 131072
```

```
uncompressed data, uncompressed metadata, uncompressed fragments,
```

```
uncompressed xattrs, uncompressed ids
```

```
duplicates are not removed
```

```
Filesystem size 19296890.54 Kbytes (18844.62 Mbytes)
```

```
99.97% of uncompressed filesystem size (19302938.39 Kbytes)
```

```
Inode table size 2129473 bytes (2079.56 Kbytes)
```

```
100.00% of uncompressed inode table size (2129473 bytes)
```

```
Directory table size 1283662 bytes (1253.58 Kbytes)
```

```
100.00% of uncompressed directory table size (1283662 bytes)
```

```
No duplicate files removed
```

```
Number of inodes 46552
```

```
Number of files 40788
```

```
Number of fragments 3540
```

```
Number of symbolic links 1085
```

```
Number of device nodes 0
```

```
Number of fifo nodes 0
```

```
Number of socket nodes 0
```

```
Number of directories 4679
```

```
Number of ids (unique uids + gids) 1
```

```
Number of uids 1
```

```
root (0)
```

```
Number of gids 1
```

```
root (0)
```

- ダウンロード後、該当のイメージは sqsh ファイルとして確認できます

```
user32002@fcvp00166:~$ ls -lh
```

```
total 19G
```

※ フォルダ構成は各開発環境によって異なります

```
-rw-rw---- 1 user32002 group032 1.8K Jul  3 12:29 how_to.md
-rw-rw---- 1 user32002 group032 533 Jul  3 15:24 pyenv_install_verbose_log_final.txt
-rw-rw---- 1 user32002 group032 68 Jun 27 10:43 README.md
drwxr----- 2 user32002 group032  0 Jun  4 2024 sbai-env-script
drwxrwx--- 5 user32002 group032  0 Jul  3 09:57 venv
drwxrwx--- 4 user32002 group032  0 Jul  9 09:42 work
-rw-r----- 1 user32002 group032 19G Jul  9 09:27 z4pymjpdqjsj+sb15106230-2+pytorch-cuda12.9.1+v0.0.0.sqsh
-rw-r----- 1 user32002 group032 75M Jul  8 14:20 z4pymjpdqjsj+sb15106230-2+superpod_hello_world+v0.0.0.sqsh
```


1. SuperPOD へのアクセス
2. Docker 利用について
 - a. イメージの作成
 - b. イメージのアップロード
 - c. イメージのダウンロード
 - d. コンテナの利用
3. Slurm 利用について

- (再掲) SuperPOD 上では [enroot コマンド](#) を介して Docker を使用します
 - (※ docker コマンドは直接使用できません)

- コンテナの作成

```
user32002@fcvp00166:~$ enroot create --name pytorch-cuda12.9.1 z4pymjpdqjsj+sb15106230-2+pytorch-cuda12.9.1+v0.0.0.sqsh
```

コンテナ名
= 仮想環境名

使用するイメージファイル

```
user32002@fcvp00166:~$ enroot create --name pytorch-cuda12.9.1 z4pymjpdqjsj+sb15106230-2+pytorch-cuda12.9.1+v0.0.0.sqsh
[INFO] Extracting squashfs filesystem...

Parallel unsquashfs: Using 8 processors
41982 inodes (188637 blocks) to write

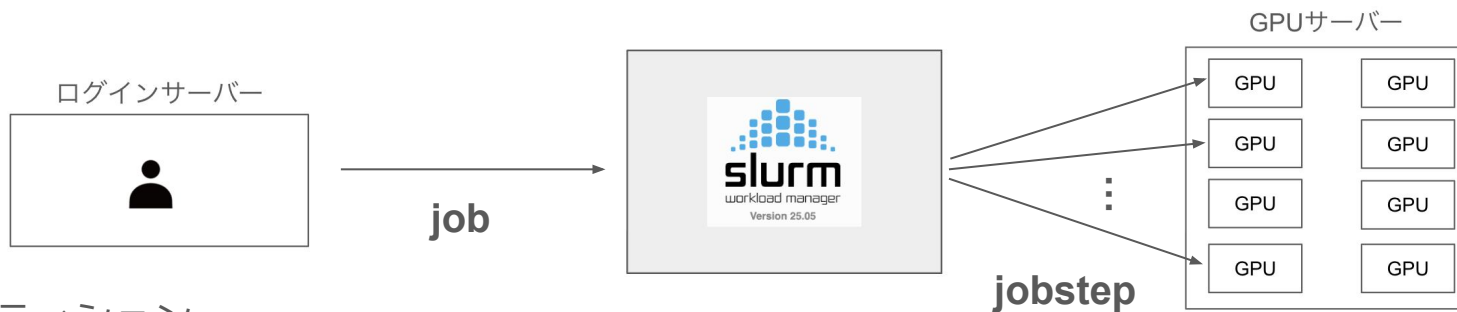
[=====/] 188637/188637 100%

created 40788 files
created 4679 directories
created 1085 symlinks
created 0 devices
created 0 fifos
created 0 sockets
user32002@fcvp00166:~$
```

1. SuperPOD へのアクセス
2. Docker 利用について
3. Slurm 利用について

- SuperPOD では slurm を介することで GPU 資源を使用できます
 - [slurm](#) … ジョブスケジューラの一つで、マルチユーザー環境での計算資源管理を担います
- slurm では2種類のジョブ（計算作業の単位）があります
 - インタラクティブジョブ：対話的に実行されるもの
 - 対話型に計算ノード上でコマンドを実行する方式
 - バッチジョブ
 - あらかじめスクリプトに記述されたジョブをでキューに投入し、Slurm が計算リソースの空き状況に応じて自動的に実行する方式

- ジョブ/ジョブステップ
 - ジョブ：計算ノードへの投入単位
 - ジョブステップ：ジョブ内での実行単位



- パーティション
 - 計算ノード群を論理的にまとめた単位
 - 各ジョブを投入するときに指定する必要があります

```
user33005@fcv00086:~/superpod_tutorial$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
033-partition  up    infinite    1     mix fcdgx00085
```

パーティション名
※環境によって変わります

パーティションに所属している
ノードのホスト名
※環境によって変わります

- MNIST での学習を実施することを想定

```
#!/bin/bash
set -euo pipefail

DATA_DIR="mnist_data"
RAW_DIR="${DATA_DIR}/MNIST/raw"
```

data_download.sh

```
# 必要なディレクトリを作成
mkdir -p "${RAW_DIR}"
```

```
echo "Downloading MNIST dataset..."
```

```
curl -fLo "${RAW_DIR}/train-images-idx3-ubyte.gz" https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
curl -fLo "${RAW_DIR}/train-labels-idx1-ubyte.gz" https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
curl -fLo "${RAW_DIR}/t10k-images-idx3-ubyte.gz" https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
curl -fLo "${RAW_DIR}/t10k-labels-idx1-ubyte.gz" https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
```

```
# prepare_mnist_offline.py
import gzip
import os
import struct
```

```
import torch
```

```
DATA_DIR = "./data" # 例: "/work/sec001/data"
raw = os.path.join(DATA_DIR, "MNIST", "raw")
pro = os.path.join(DATA_DIR, "MNIST", "processed")
os.makedirs(pro, exist_ok=True)
```

```
def read_images(path):
    with gzip.open(path, "rb") as f:
        assert struct.unpack(">I", f.read(4))[0] == 2851
        n, r, c = struct.unpack(">III", f.read(12))
        buf = f.read(n * r * c)
        return torch.frombuffer(memoryview(buf), dtype=torch.uint8).reshape(n, r, c)
```

```
def read_labels(path):
    with gzip.open(path, "rb") as f:
        assert struct.unpack(">I", f.read(4))[0] == 2049
        n = struct.unpack(">I", f.read(4))[0]
        buf = f.read(n)
        return torch.frombuffer(memoryview(buf), dtype=torch.uint8).to(torch.long)
```

```
train_x = read_images(os.path.join(raw, "train-images-idx3-ubyte.gz"))
train_y = read_labels(os.path.join(raw, "train-labels-idx1-ubyte.gz"))
test_x = read_images(os.path.join(raw, "t10k-images-idx3-ubyte.gz"))
test_y = read_labels(os.path.join(raw, "t10k-labels-idx1-ubyte.gz"))
```

```
torch.save((train_x, train_y), os.path.join(pro, "training.pt"))
torch.save((test_x, test_y), os.path.join(pro, "test.pt"))
print("OK: processed/training.pt, processed/test.pt を生成しました")
```

preprocess.py

```
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

main.py

```
# ===== Parameters =====
```

```
DATA_DIR = "./data"
BATCH_SIZE = 128
EPOCHS = 5
LR = 1e-3
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

```
# ===== GPU 情報を表示 =====
```

```
print(f"Using device: {DEVICE}")
if DEVICE == "cuda":
    print(f"GPU: {torch.cuda.get_device_name(0)}")
    print(f"CUDA version: {torch.version.cuda}")
    print(f"Available GPU count: {torch.cuda.device_count()}")
```

```
# ===== Dataset =====
```

```
transform = transforms.Compose([transforms.ToTensor()])
train_ds = datasets.MNIST(DATA_DIR, train=True, transform=transform, download=False)
test_ds = datasets.MNIST(DATA_DIR, train=False, transform=transform, download=False)
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE)
```

```
# ===== Model =====
```

```
model = nn.Sequential(
    nn.Flatten(), nn.Linear(28 * 28, 256), nn.ReLU(), nn.Linear(256, 10)
).to(DEVICE)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LR)
```

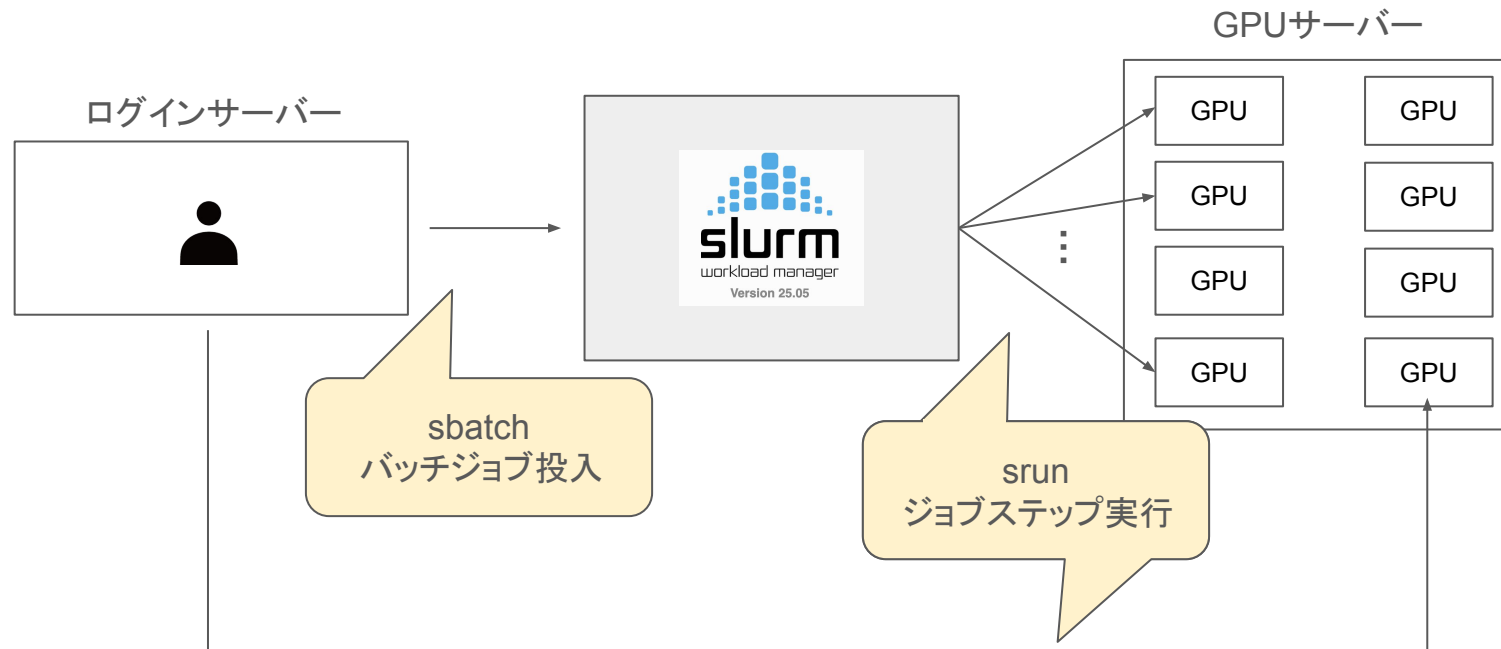
```
# ===== Train =====
```

```
for epoch in range(1, EPOCHS + 1):
    model.train()
    total_loss = 0.0
    for x, y in train_loader:
        x, y = x.to(DEVICE), y.to(DEVICE)
        optimizer.zero_grad()
        loss = criterion(model(x), y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * x.size(0)
    print(f"Epoch {epoch}/{EPOCHS} - loss: {total_loss/len(train_ds):.4f}")
```

```
# ===== Validation =====
```

```
model.eval()
correct = total = 0
with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(DEVICE), y.to(DEVICE)
        pred = model(x).argmax(1)
        correct += (pred == y).sum().item()
        total += y.size(0)
print(f"Test accuracy: {correct/total*100:.2f}%")
```

- バッチジョブとインタラクティブジョブの概要
 - **sbatch** : 「バッチジョブ投入～ジョブステップ実行」まで一括で実行
 - **srun** : 「ジョブステップ実行」

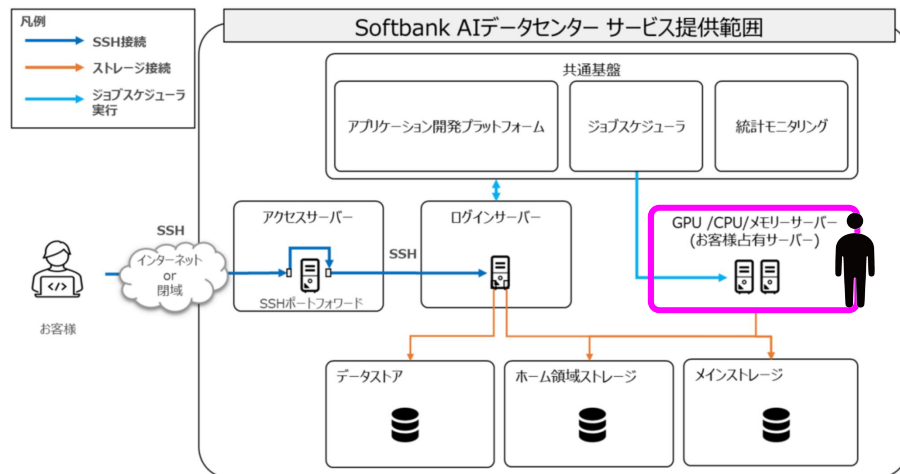


- インタラクティブジョブの起動 (srun コマンド)

```
srun --partition=033-partition --gres=gpu:1 --time=3:00:00 --pty bash
```

```
user33005@fcvp00086:~$ srun --partition=033-partition --gres=gpu:1 --time=3:00:00 --pty bash
srun: job 1039500 queued and waiting for resources
srun: job 1039500 has been allocated resources
user33005@fcdgx00085:~$
```

作業場所がDGX上に
移動しました
(fcvp → fcdgx)



```
user33005@fcdgx00085:~$ nvidia-smi
Fri Aug 29 17:31:29 2025
```

NVIDIA-SMI 535.161.08			Driver Version: 535.161.08			CUDA Version: 12.2		
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp		Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute M.	MIG M.
0	NVIDIA A100-SXM4-80GB	P0	63W / 400W	00000000:47:00.0	Off	0%	Default	0
N/A	30C			0MiB / 81920MiB			Disabled	

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	Usage
ID	ID	ID					
No running processes found							

nvidia-smi から GPU が認識できていることも
確認できます

- 学習スクリプトの実行
 - ログインノードとGPUノードはホームを共有しているので、準備した ~/superpod_tutorial/ 以下のファイルがそのまま使えます

```
user33005@fcpv00086:~$ srun --partition=033-partition --gres=gpu:1 --time=3:00:00 --pty bash
srun: job 1039500 queued and waiting for resources
srun: job 1039500 has been allocated resources
user33005@fcdgx00085:~$
```

インタラクティブジョブの実行

```
user33005@fcdgx00085:~/superpod_tutorial$ enroot start --mount /home/user33005/superpod_tutorial/:/work cudaenv

=====
== CUDA ==
=====

CUDA Version 12.9.1

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.

bash: module: command not found
user33005@fcdgx00085:/work$ python3 main.py
Using device: cuda
GPU: NVIDIA A100-SXM4-80GB
CUDA version: 12.6
Available GPU count: 1
Epoch 1/5 - loss: 0.3591
Epoch 2/5 - loss: 0.1589
Epoch 3/5 - loss: 0.1074
Epoch 4/5 - loss: 0.0805
Epoch 5/5 - loss: 0.0628
Test accuracy: 97.41%
user33005@fcdgx00085:/work$
```

Docker コンテナを起動

← GPU 認識できています

← 学習が回っています

学習スクリプトの実行

- バッチジョブ (sbatch)
 - 大量のジョブを自動実行して欲しい場合などに使用すると便利です
 - ex. パラメーターを変えて同じ計算用スクリプトを実行したい

- 実行方法

- 下記のようなファイルを用意してジョブを投入します

```
$ sbatch sample.sbatch
```

```
#!/bin/bash
#SBATCH --partition=033-partition
#SBATCH --job-name=echo-test
#SBATCH --output=echo_%j.out
#SBATCH --time=00:01:00
#SBATCH --ntasks=1

echo "Hello from SLURM job $SLURM_JOB_ID"
```

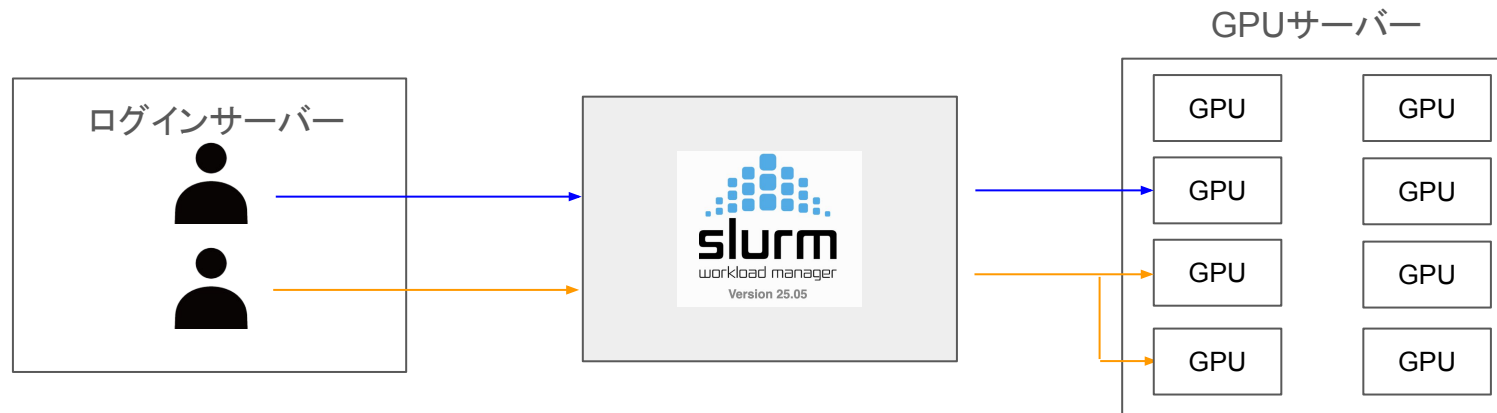
ジョブの設定ファイル
(ここではファイル名を sample.sbatch としました)

```
user33005@fcvp00086:~/superpod_tutorial$ sbatch sample.sbatch
Submitted batch job 1039726
```

```
user33005@fcvp00086:~/superpod_tutorial$ ls -lhtr
total 21G
-rw-r----- 1 user33005 group033 21G Aug 26 11:39 z4pymjpdqjsj+sb15106230-2+cuda12.9.1-cudnn-devel-ubuntu24.04+v0.0.0.sqsh
drwxrwx--- 3 user33005 group033  0 Aug 26 13:27 data
-rw-rw---- 1 user33005 group033 849 Aug 26 13:53 README.md
drwxrwx--- 2 user33005 group033  0 Aug 26 13:59 sec002
drwxrwx--- 2 user33005 group033  0 Aug 29 17:13 sec001
-rw-rw---- 1 user33005 group033 1.9K Aug 29 17:17 main.py
-rw-rw---- 1 user33005 group033 29 Aug 29 17:49 echo_1039726.out
-rw-rw---- 1 user33005 group033 191 Aug 29 17:50 sample.sbatch
user33005@fcvp00086:~/superpod_tutorial$ cat echo_1039726.out
Hello from SLURM job 1039726
```

ジョブの投入、
出力ログが生成されていることが分かります

- **sbatch** : バッチスクリプト（設定ファイル）を Slurm に投入する
 - ジョブIDの発行～実行までを Slurm が管理してくれる



- **squeue** : 投入したジョブのステータスを確認する

```
user33005@fcpv00086:~/superpod_tutorial$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	ODELIST(Reason)
1040149_[9-10%8]	033-partition	train_job.sbatch	user33005	PD	0:00	1	(JobArrayTaskLimit)
1040149_1	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_2	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_3	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_4	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_5	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_6	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_7	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085
1040149_8	033-partition	train_job.sbatch	user33005	R	0:02	1	fcdgx00085

- バッチジョブを大量投入する例

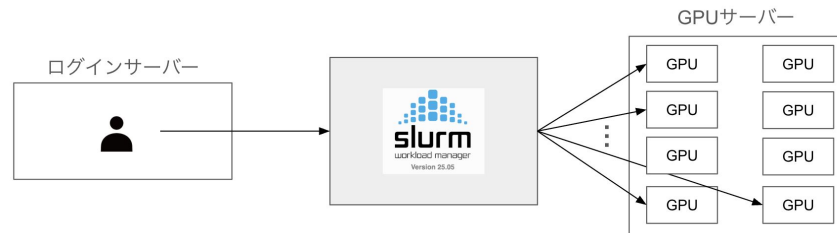
```
#!/bin/bash
#SBATCH --job-name=param-scan
#SBATCH --partition=033-partition
#SBATCH --output=logs/scan_%A_%a.out
#SBATCH --error=logs/scan_%A_%a.err
#SBATCH --time=00:10:00
#SBATCH --gres=gpu:1
#SBATCH --array=0-23%8

# ===== 探索パラメータ =====
BATCH_SIZES=(16 32 128 256)
LRS=(0.001 0.0001 0.00001)
SEEDS=(0 1)

# 配列IDを使って組み合わせを展開
TASK_ID=${SLURM_ARRAY_TASK_ID}
BATCH_SIZE=${BATCH_SIZES[${TASK_ID} % ${#BATCH_SIZES[@]}]}
LR=${LRS[${TASK_ID} / ${#BATCH_SIZES[@]} % ${#LRS[@]}]}
SEED=${SEEDS[${TASK_ID} / (${#BATCH_SIZES[@]} * ${#LRS[@]}) % ${#SEEDS[@]}]}

echo "Node=$(hostname)"
echo "JobID=$SLURM_JOB_ID TaskID=$SLURM_ARRAY_TASK_ID"
echo "Params: batch_size=$BATCH_SIZE, lr=$LR, seed=$SEED"

# enroot 内で main.py 実行
srun enroot start \
  --mount /home/user33005/superpod_tutorial:/work \
  cudaenv \
  bash -lc "cd /work && python3 main02.py --data_dir /work/data --batch_size ${BATCH_SIZE} \
  --seed ${SEED} --lr ${LR}"
```



- 例) パラメータスキャン
 - 学習スクリプト固定で
パラメーターごとにジョブを投入する

```
user33005@fcv00085:~/superpod_tutorial$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	MODELIST(REASON)
1040210 [8-23]	033-partition	param-scan	user33005	PD	0:00	1	(Resources)
1040210_0	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_1	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_2	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_3	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_4	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_5	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_6	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085
1040210_7	033-partition	param-scan	user33005	R	0:00	1	fcgdx00085

- パイプラインとして実行
 - 前処理（preprocess）→ 学習（train）→ 評価（evaluate）
 - `--dependency=afterok` を利用

```
#!/bin/bash
```

```
# 前処理ジョブを投げる
```

```
jid1=$(sbatch preprocess.sh | awk '{print $4}')  
echo "Submitted preprocess.sh as JobID=$jid1"
```

```
# 前処理成功後に学習ジョブを投げる
```

```
jid2=$(sbatch --dependency=afterok:$jid1 train.sh | awk '{print $4}')  
echo "Submitted train.sh as JobID=$jid2 (after preprocess)"
```

```
# 学習成功後に評価ジョブを投げる
```

```
jid3=$(sbatch --dependency=afterok:$jid2 evaluate.sh | awk '{print $4}')  
echo "Submitted evaluate.sh as JobID=$jid3 (after train)"
```

① 前処理のジョブを投入
ジョブIDをスクリプトで取得

② 学習のジョブを投入
ジョブ投入①で取得するIDと紐づける

③ 評価のジョブを投入
ジョブ投入②で取得するIDと紐づける

```
user33005@fcpv00086:~/superpod_tutorial/sec003$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	ODELIST(REASON)
1040235	033-partition	train	user33005	PD	0:00	1	(Dependency)
1040236	033-partition	evaluate	user33005	PD	0:00	1	(Dependency)
1040234	033-partition	preprocess	user33005	R	0:01	1	fcdgx00085

- Slurm 公式ドキュメント ([link](#))

EOF