AOBA-S
アプリケーション
利用手順書

日本電気株式会社

2023 年 8 月 4 日

第二版

＜＜　改版履歴　＞＞

版数	改版日	改版理由	改版者
第一版	2023.8.1	第一版として作成.	NEC
第二版	2023.8.4	「2.6 コンテナジョブ実行方法」、「3.2 ジョブ実行」において、投入用スクリプトの記載を変更。 「2.3 コンテナイメージの準備」において、ビルドの際にメモリ上限値を変更する記載を追加。	NEC

目次

1　はじめに

本書に記載している操作はフロントエンドサーバ（sfront、shpcif）での実行を前提としております。

2　Singularity の利用方法

2.1　環境変数 PATH の設定

Singularity は、AOBA-S システムの以下のディレクトリにインストールされています。
/mnt/lustre/ap/singularity/3.11

ログイン時の環境変数 PATH には含まれていないため、「module」コマンドを使用して環境変数 PATH への追加を行ってください。
module load Singularity/3.11

2.2　Singularity の主要コマンド

$ singularity [global option] **<command>** [option] ...

コマンド名	説明
build	イメージのビルド(作成や更新)を実施します
exec	コンテナ内で指定コマンドを実行します
run	コンテナ内の標準コマンドを実行します
shell	コンテナ内でシェルを起動します
pull	URI 指定して、イメージを取得します。
search	ライブラリからコンテナを検索します
inspect	イメージのメタデータを表示します。
help	コマンドヘルプを表示します。

Singularity コマンド実行例

・Singularity Library 上のイメージを検索する

[書式]　　$ singularity search <search_query>

[実行例]　　$ singularity search gromacs

・コンテナ内のメタデータを参照する

[書式]　　$ singularity inspect <image_name>

[実行例]　　$ singularity inspect centos.sif

・コンテナ内でシェルを起動する(コンテナの中身を確認する)

[書式]　　$ singularity shell <image_name>

[実行例]　　$ singularity shell centos.sif

・コンテナの標準コマンドを実行する

[書式]　　$ singularity run <image_name>

[実行例]　　$ singularity run centos.sif

2.3　コンテナイメージの準備

コンテナイメージは、利用者様が準備されたもの以外にも、DockerHub や SingularityHub などコンテナレジストリにあるイメージをダウンロードして使用することも可能です。

・Singularity Library からイメージを取得する場合

① イメージを検索する

```
$ singularity search centos8
...
        library://apisith.won/default/centos8.3-py38-cuda11.1:latest
        library://apisith.won/default/centos8.3-py38:latest
...
```

[書式] $ singularity search <検索クエリ>

② イメージを取得する

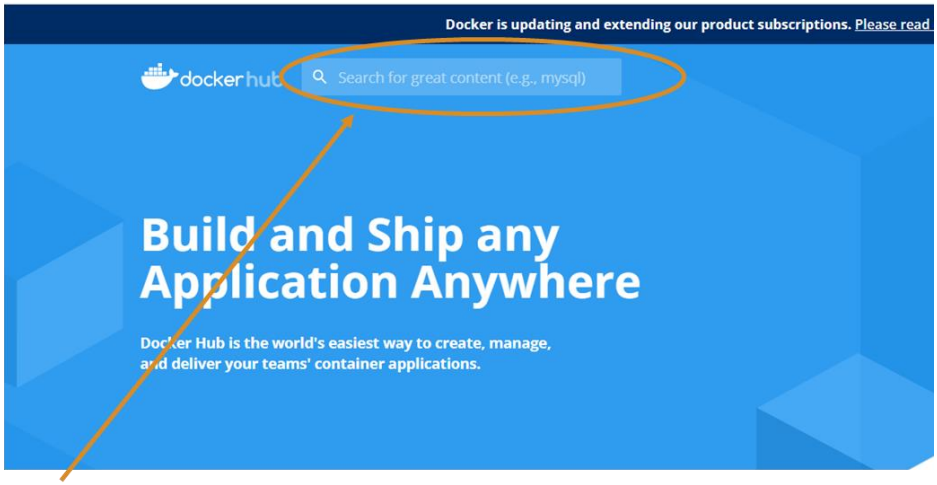
```
$ singularity build centos.sif library://apisith.won/default/centos8.3-py38:latest
```

[書式] $ singularity build <イメージファイル名> library://<イメージパス>:<tag 名>

※コマンドの実行前に「ulimit -S -v 67108864」を実行し、メモリ上限値を変更してください

・Docker Hub 上のイメージを Singularity 用に変換して利用する場合

① Docker Hub (https://hub.docker.com/)でイメージを検索



検索BOXにキーワードを入力し検索

pullコマンドのイメージパスを
確認します。(この場合は「centos」)

② Docker Hub からイメージを取得

```
$ singularity build centos.sif docker://docker.io/centos:latest
```

[書式] $ singularity build <イメージファイル名> ¥

docker://docker.io/<コンテナイメージパス>:<tag 名>

2.4　コンテナ起動方法

コンテナ起動の主要なコマンドは run、exec、shell の 3 種類です。

(1) singularity run

指定したコンテナイメージのコンテナを起動し、コンテナ起動時に実行するスクリプトがコンテナ内にあらかじめ定義されている場合は、そのスクリプトを自動的に実行します。

```
$ singularity run container.sif
```

(2) singularity exec

指定したコンテナイメージのコンテナ内でコマンドを実行します。

```
$ singularity exec container.sif ./a.out
```

(3) singularity shell

指定したコンテナイメージのコンテナ内に新しい対話型シェルが生成され、対話形式でコマンドの実行ができます。

```
$ singularity shell container.sif
```

2.5　環境設定

(1) ディレクトリのバインド指定

Singularity では、ホスト側のディレクトリをコンテナにバインドすることで、ホスト側のディレクトリをコンテナ内から参照することが可能となります。

デフォルトで以下のディレクトリがバインドされ、コンテナ内で参照が可能となります。

- $HOME
- $PWD
- /tmp
- /etc
- /proc
- /sys
- /dev

コンテナ側にバインドするディレクトリを指定する場合には、以下の方法で行います。

環境変数 SINGULARITY_BIND で指定：

```
$ export SINGULARITY_BIND=/mnt/lustre/comp,/usr/lib64
```

コンテナ起動時のオプション--bind で指定：

```
$ singularity run --bind /mnt/lustre/comp,/usr/lib64 container.sif
```

(2) 環境変数の設定

Singularity では、ホスト側で設定されている環境変数はコンテナに伝搬します。
ただし、ホスト側で環境変数 PATH に格納されている値は、コンテナ内では USER_PATH という別の環境変数に格納されています。コンテナを起動し、コンテナ内で USER_PATH に格納されている値を PATH に格納する場合は、例えば以下のようにコマンドを実行します。

```
$ singularity shell --bind /mnt/lustre/comp container.sif
Singularity > export PATH=$USER_PATH:$PATH
```

シェルスクリプトをあらかじめ作成して、コンテナ内でシェルスクリプトを実行する方法もあります。

```
$ cat commands.sh
export PATH=$USER_PATH:$PATH
python sample.py
$ singularity exec --bind /mnt/lustre/comp container.sif ./commands.sh
```

2.6 コンテナジョブ実行方法

Singularity を利用したコンテナジョブを、バッチシステム(NQSV)で実行します。

概要

Ubuntu 20.4、GNU コンパイラ、OpenMPI 4.0.6 のコンテナイメージを作成し、イメージ作成時にサンプルプログラムをコンパイルし、イメージ内に配置します。

① プログラムの準備

以下のサンプルプログラムを MPI 用コンパイラでコンパイルします。

サンプルプログラム：mpitest.c

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv) {
        int rc;
        int size;
        int myrank;
```

```
        rc = MPI_Init (&argc, &argv);
        if (rc != MPI_SUCCESS) {
                fprintf (stderr, "MPI_Init() failed");
                return EXIT_FAILURE;
        }

        rc = MPI_Comm_size (MPI_COMM_WORLD, &size);
        if (rc != MPI_SUCCESS) {
                fprintf (stderr, "MPI_Comm_size() failed");
                goto exit_with_error;
        }

        rc = MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
        if (rc != MPI_SUCCESS) {
                fprintf (stderr, "MPI_Comm_rank() failed");
                goto exit_with_error;
        }

        fprintf (stdout, "Hello, I am rank %d/%d", myrank, size);

        MPI_Finalize();

        return EXIT_SUCCESS;

 exit_with_error:
        MPI_Finalize();
        return EXIT_FAILURE;
}
```

② definition file の作成

以下の definition file を作成します。

※definition file とサンプルプログラムは同じディレクトリに置いてください。

definition file : ubuntu-openmpi.def

```
Bootstrap: docker
From: ubuntu:latest

%files
    mpitest.c /opt

%environment
    export OMPI_DIR=/opt/ompi
    export SINGULARITY_OMPI_DIR=$OMPI_DIR
    export SINGULARITYENV_APPEND_PATH=$OMPI_DIR/bin
    export SINGULAIRTYENV_APPEND_LD_LIBRARY_PATH=$OMPI_DIR/lib
    export PATH=$OMPI_DIR/bin:$PATH
    export LD_LIBRARY_PATH=$OMPI_DIR/lib:$LD_LIBRARY_PATH

%post
    echo "Installing required packages..."
    apt-get update && apt-get install -y wget git bash gcc gfortran g++ make file bzip2

    echo "Installing Open MPI"
    export OMPI_DIR=/opt/ompi
    export OMPI_VERSION=4.0.1
    export OMPI_URL="https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-$OMPI_VERSION.tar.bz2"
    mkdir -p /tmp/ompi
    mkdir -p /opt
    # Download
    cd /tmp/ompi && wget -O openmpi-$OMPI_VERSION.tar.bz2 $OMPI_URL && tar -xjf openmpi-$OMPI_VERSION.tar.bz2
    # Compile and install
    cd /tmp/ompi/openmpi-$OMPI_VERSION && ./configure --prefix=$OMPI_DIR && make install
    # Set env variables so we can compile our application
    export PATH=$OMPI_DIR/bin:$PATH
    export LD_LIBRARY_PATH=$OMPI_DIR/lib:$LD_LIBRARY_PATH
```

DockerHub より ubuntu のイメージを入手します。

サンプルプログラム「mpitest.c」をコンテナイメージ内の/opt にコピーします。

コンテナイメージ内の環境変数を設定します。

コンテナイメージのベース OS インストール後の処理を記載します。

OpenMPI インストール後、サンプルプログラムをコンパイルします。

```
    export MANPATH=$OMPI_DIR/share/man:$MANPATH

    echo "Compiling the MPI application..."
    cd /opt && mpicc -o mpitest mpitest.c
    rm -fr /tmp/ompi
```

注意事項：
/tmp のような実環境の共有領域にファイルを作成した場合、singularity を介して作成されたファイルの所有者情報は、実行ユーザとは異なるユーザ ID/グループ ID となり、削除できなくなる可能性がありますので、build 実行後には、共有領域上に作成したファイルは削除するようにしてください。

③ コンテナイメージの作成

definition file をもとにコンテナイメージを作成します。

```
$ singularity build --fakeroot ubuntu-openmpi.sif ubuntu-openmpi.def
```

[書式] $ singularity build --fakeroot <イメージファイル名> < definition file >

④ コンテナイメージのメタデータ確認

作成したコンテナイメージのメタデータを確認します。

```
$ singularity inspect ubuntu-openmpi.sif
org.label-schema.build-arch: amd64
org.label-schema.build-date: Wednesday_24_May_2023_17:41:24_JST
org.label-schema.schema-version: 1.0
org.label-schema.usage.singularity.deffile.bootstrap: docker
org.label-schema.usage.singularity.deffile.from: ubuntu:latest
org.label-schema.usage.singularity.version: 3.7.3
```

[書式] $ singularity inspect <イメージファイル名>

⑤ コンテナイメージのシェル起動

```
$ singularity shell ubuntu-openmpi.sif
Singularity> cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
```

```
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"
UBUNTU_CODENAME=jammy
Singularity> mpirun --version
mpirun (Open MPI) 4.0.1

Report bugs to http://www.open-mpi.org/community/help/
Singularity> ls -l /opt
total 17
-rwxrwxr-x 1 root root 16336 May 24 17:41 mpitest
-rw-r--r-- 1 root root   906 May 24 17:31 mpitest.c
drwxrwxr-x 7 root root   100 May 24 17:36 ompi
```

[書式] $ singularity shell <イメージファイル名>

⑥ プログラムの実行

バッチシステムからコンテナイメージの起動と、イメージ内のプログラムを実行します。

以下の投入用スクリプトを作成し、リクエストを投入します。

投入用スクリプト

```
#!/bin/sh
#PBS -N Test-JOB
#PBS -T openmpi
#PBS -q sxs
#PBS -b 1
#PBS -v OMP_NUM_THREADS=1
#PBS -l elapstim_req=00:10:00

IMAGE=/uhome/w20195/Singularity/sif/ubuntu-openmpi.sif
LM=/opt/mpitest
```

```
module load Singularity/3.11
/usr/mpi/gcc/openmpi-4.1.5a1/bin/mpirun $NQSV_MPIOPTS -np 16
singularity exec $IMAGE $LM
```

※ログイン環境によっては module の load が失敗する場合があります。投入用スクリプト内で「module load ～」行の前に、以下を記載することで回避できます。

source /etc/profile.d/modules.sh（投入用スクリプトが bash の場合）
source /etc/profile.d/modules.csh（投入用スクリプトが csh の場合）

プログラムの実行後、標準出力で以下の結果が得られることを確認します。

```
Hello, I am rank 11/16
Hello, I am rank 2/16
Hello, I am rank 3/16
Hello, I am rank 9/16
Hello, I am rank 1/16
Hello, I am rank 10/16
Hello, I am rank 5/16
Hello, I am rank 0/16
Hello, I am rank 8/16
Hello, I am rank 4/16
Hello, I am rank 13/16
Hello, I am rank 12/16
Hello, I am rank 14/16
Hello, I am rank 6/16
Hello, I am rank 15/16
Hello, I am rank 7/16
```

3　Quantum Espresso の利用方法

3.1　環境変数 PATH の設定

Quantum Espresso は、AOBA システムの以下のディレクトリにインストールされています。
/mnt/lustre/ap/QE/

ログイン時の環境変数 PATH には含まれていないため、「module」コマンドを使用して環境変数 PATH への追加を行ってください。※バージョンによって読み込む module が異なります
module load Quantum_ESPRESSO/6.3
module load Quantum_ESPRESSO/6.6

3.2　ジョブ実行

以下のサンプルデータファイルを作成します。
サンプルデータ：atom.in

```
 &control
     calculation='scf',
 /
 &system
    ibrav=1,
    celldm(1)=10.0,
    nat=1,
    ntyp=1,
    nbnd=6,
    ecutwfc=25.0,
    ecutrho=200.0,
    occupations='from_input',
 /
 &electrons
    mixing_beta=0.25,
 /
ATOMIC_SPECIES
O       15.99994     O.pz-rrkjus.UPF
ATOMIC_POSITIONS alat
O     0.000000000     0.000000000     0.000000000
```

```
K_POINTS {gamma}
OCCUPATIONS
2.0 1.3333333333 1.3333333333 1.3333333333 0.0 0.0
```

続けてジョブ投入用スクリプトを作成します。

```
#!/bin/bash
#PBS -q sxs
#PBS -T necmpi
#PBS -l elapstim_req=00:05:00
#PBS --venode=1
#PBS -N qe_test

export LM=pw.x
cd ${PBS_O_WORKDIR}
module load Quantum_ESPRESSO/6.6

mpirun -np 8 -path $PATH ${LM} -input atom.in
```

※ログイン環境によっては module の load が失敗する場合があります。投入用スクリプト内で「module load ～」行の前に、以下を記載することで回避できます。

source /etc/profile.d/modules.sh（投入用スクリプトが bash の場合）
source /etc/profile.d/modules.csh（投入用スクリプトが csh の場合）

プログラム実行後、標準出力で以下の結果が得られることを確認します。

```
     Program PWSCF v.6.6 starts on 25May2023 at  9:26: 8

     This program is part of the open-source Quantum ESPRESSO suite
     for quantum simulation of materials; please cite
          "P. Giannozzi et al., J. Phys.:Condens. Matter 21 395502 (2009);
          "P. Giannozzi et al., J. Phys.:Condens. Matter 29 465901 (2017);
           URL http://www.quantum-espresso.org",
     in publications or presentations arising from this work. More details at
```

```
     http://www.quantum-espresso.org/quote

     Parallel version (MPI), running on       8 processors

     MPI processes distributed on       8 nodes
     R & G space division:   proc/nbgrp/npool/nimage =          8
     Fft bands division:       nmany       =          1
     Reading input from atom.in

     Current dimensions of program PWSCF are:
     Max number of different atomic species (ntypx) = 10
     Max number of k-points (npk) =   40000
     Max angular momentum in pseudopotentials (lmaxx) =   3
     file O.pz-rrkjus.UPF: wavefunction(s)   2S renormalized

     gamma-point specific algorithms are used

     Subspace diagonalization in iterative solution of the eigenvalue problem:
     a serial algorithm will be used


     Parallelization info
     --------------------
     sticks:    dense   smooth      PW      G-vecs:     dense    smooth       PW
     Min            198        98       23                    5975      2106       260
     Max            200       100       26                    5980      2112       266
     Sum           1597       793      193                   47833     16879      2103
...
```

4　参考

各アプリケーションの詳細なマニュアルは以下にあります。

マニュアル名	URL
Singularity ユーザガイド	https://repo.sylabs.io/guides/pro-3.11/user-guide/index.html
Quantum Espresso ユーザガイド	https://www.quantum-espresso.org/Doc/pw_user_guide/