# AOBA-B

# Singularity 利用手順書

日本電気株式会社

2026 年 5 月 18 日

四版

＜＜　改版履歴　＞＞

| 版数 | 改版日 | 改版理由 | 改版者 |
|---|---|---|---|
| 初版 | 2021.11.12 | 初版として作成 | NEC |
| 二版 | 2022.02.09 | コンテナイメージのカスタマイズ追加 | NEC |
| 三版 | 2022.04.15 | 2.1 環境変数 PATH 設定の修正 | NEC |
| 四版 | 2026.05.18 | Singularity 利用時の制限事項を追加 | NEC |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 目次

※制限事項

本システムにおける Singularity 利用にあたっては、以下の制限事項を遵守してください。

(1) コンテナ形式の利用

・Singularity で実行時に指定するコンテナは「.sif 形式」を使用してください。

(2) sandbox 形式の利用

・sandbox 形式は、コンテナイメージのカスタマイズを行う場合にのみ使用してください。

・カスタマイズはフロントエンド上の /tmp 配下で実施し、ユーザホーム領域（/uhome/xxxxxx）配下では実施しないでください。

・AOBA-B（LX）では sandbox 形式は使用しないでください。

(3) /tmp 利用時の注意

・/tmp は一時領域のため、作業完了後は作成したファイルを削除してください。

# 1　仮想化技術概要

仮想化技術とは、物理的な環境にあるハードウェアに仮想化ソフトウェア使った基盤をつくり、その上に仮想化されたハードウェアを配置します。仮想化サーバ上では、それぞれ別々の OS やアプリケーションを作動させることができます。これによって限られた物理的なリソース（CPU、メモリ、ハードディスク、ネットワークなど）の能力を有効活用することができます。
仮想化の方式としては、ハイパーバイザー型、コンテナ型があり、Singularity は、HPC 環境向けに設計されたオープンソースのコンテナ型プラットフォームとなります。

## 1.1　コンテナ型

コンテナ型仮想化技術とは、ホストとなるコンピュータの OS に、仮想的なユーザ空間（コンテナ）を構築し、ホスト OS 上にインストールされたコンテナ管理ソフトウェアを介して、アプリケーションを実行させる方法です。
コンテナの中で各ユーザのアプリケーションが独立して動作するため、仮想サーバというよりは、隔離された別々のアプリケーション実行環境というほうがイメージは近いです。
また、コンテナはホスト OS のカーネルを利用して動作しているため、ゲスト OS を持ちません。その分ハイパーバイザー型よりオーバーヘッドが少なく、軽く速い処理が行えます。
コンテナ型の例としては、「Docker」「Singularity」が該当し、研究室のサーバ上にあるアプリケーションの実行環境をコンテナ化(コンテナイメージ作成)し、別サーバにコンテナを持っていくことで、異なるサーバでアプリケーションの実行を可能にします。

## 1.2　ハイパーバイザー型

ハイパーバイザーとは仮想化のための OS のようなもので、サーバにインストールし、その OS の上で仮想マシン(ゲスト OS)を稼働させる方法です。

- ホスト OS が不要でハードウェアの直接制御が可能
- システム全体の観点から見てリソースの使用効率がよい
- 管理する物理サーバの台数削減が可能

といったメリットがあります。
ハイパーバイザー型の例としては、「Vmware ESXi」「Linux KVM」が該当し、サーバ上で複数の仮想サーバを立ち上げ運用するといった用途となります。

# 2　Singularity の利用方法

## 2.1　環境変数 PATH の設定

Singularity 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おり、ログイン時に環境変数 PATH に設定されています。
/mnt/stfs/ap/singularity

PATH に含まれていない場合は、以下の設定で PATH への追加を行ってください。
bash の場合
export PATH=/mnt/stfs/ap/singularity/bin:$PATH

csh の場合
setenv PATH /mnt/stfs/ap/singularity/bin:$PATH

## 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
(1) イメージを検索する
[書式] $ singularity search <検索クエリ>

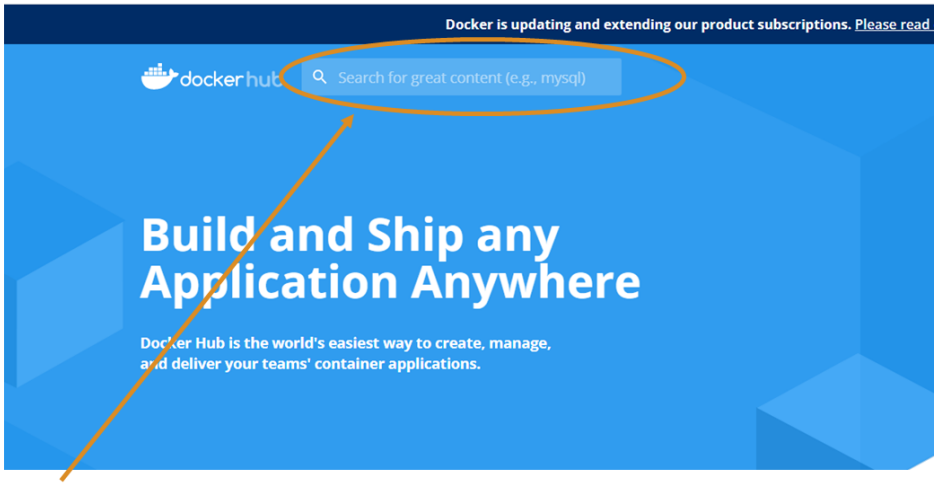
```
[実行例]
$ singularity search centos8
…
        library://apisith.won/default/centos8.3-py38-cuda11.1:latest
        library://apisith.won/default/centos8.3-py38:latest
…
```

(2) イメージを取得する
[書式] $ singularity build <イメージファイル名> library://<イメージパス>:<tag 名>

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

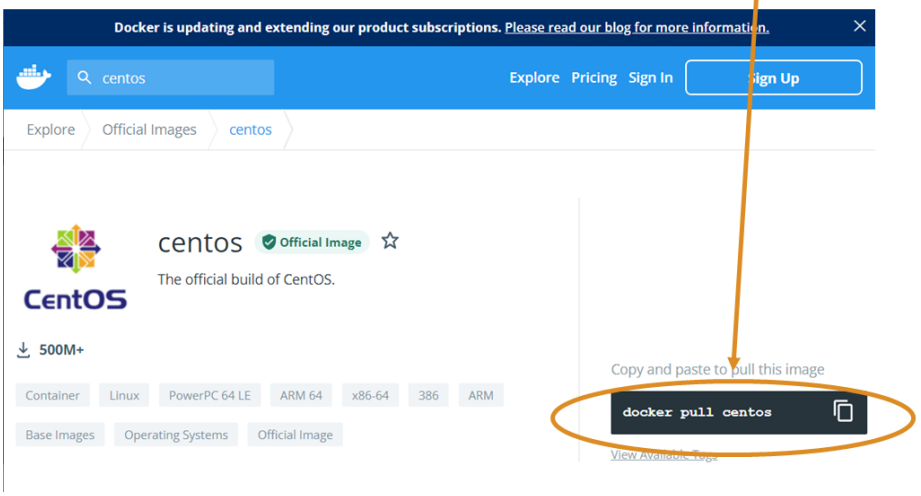
■Docker Hub 上のイメージを Singularity 用に変換して利用する

(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のパスを
確認します。（この場合は、centos）




(2) Docker Hub からイメージを取得

[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>

[実行例]

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

## 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。

(1-1)　環境変数 singularity_bindpath で指定

例

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

(1-2) コンテナ起動時のオプション--bind で指定

例

```
$ singularity run --bind /mnt/stfs/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/stfs/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/stfs/comp container.sif ./commands.sh
```

# 3　コンテナジョブの実行方法

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
Ubuntsu 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。

## 3.1　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：

```
#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;
}
```

### 3.2　コンテナイメージの作成

以下の definition file を使用し、コンテナイメージを作成します。
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。

```
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

    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
```

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。

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

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

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。

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。

コンテナイメージの作成

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

```
[実行例]
$ singularity build --fakeroot ubuntu-openmpi.sif ubuntu-openmpi.def
```

コンテナイメージのメタデータ確認

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

```
[実行例]
$ singularity inspect ubuntu-openmpi.sif
org.label-schema.build-arch: amd64
org.label-schema.build-date: Friday_12_November_2021_6:23:32_UTC
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
```

コンテナイメージのシェル起動

[書式] $ singularity shell <イメージファイル名>

```
[実行例]
$ singularity shell ubuntu-openmpi.sif
Singularity> cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
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"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
Singularity> mpirun --version
mpirun (Open MPI) 4.0.1

Report bugs to http://www.open-mpi.org/community/help/
Singularity> ls -l /opt
total 18
-rwxrwxr-x 1 root root 17072 Nov 12 15:23 mpitest
-rw-r--r-- 1 root root   909 Nov 12 15:11 mpitest.c
drwxrwxr-x 7 root root   100 Nov 12 15:17 ompi
```

### 3.3　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。

投入用スクリプト

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

IMAGE=/uhome/w20195/Singularity/sif/ubuntu-openmpi.sif      ← コンテナイメージ
SINGULARITY=/mnt/stfs/ap/singularity/bin/singularity        ← singularity コマンド
LM=/opt/mpitest                  ← コンテナ内の実行プログラム
```

```
mpirun $NQSV_MPIOPTS -np 16 $SINGULARITY exec $IMAGE $LM
```

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。

```
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
```

# 4　コンテナイメージのカスタマイズ

作成したコンテナイメージを sandbox と呼ばれる形式に変換し、カスタマイズすることが可能です。

【重要】
sandbox 形式は、コンテナイメージのカスタマイズを行う場合にのみ使用してください。
カスタマイズは、フロントエンド上の /tmp 配下で実施し、ユーザホーム領域（/uhome/xxxxxx）配下では実施しないでください。
本手順終了後は /tmp 配下の sandbox を必ず削除してください。

概要
CentOS8 のコンテナイメージを sandbox 形式に変換し、GNU Fortran のインストールと実行モジュールのコンパイルを実施します。
再度コンテナイメージに変換し、カスタマイズの内容が反映されていることを確認します。

## 4.1　コンテナイメージのカスタマイズ

(1) sandbox 形式への変換
[書式] $ singularity build --fakeroot --sandbox --fix-perms <sandbox 名> <イメージファイル名>

[実行例]
$ cd /tmp

$ singularity build --fakeroot --sandbox --fix-perms centos8 centos8.sif

(2) sandbox 形式イメージでの起動
[書式] $ singularity shell --fakeroot --writable <sandbox 名>

[実行例]
$ singularity shell --fakeroot --writable centos8

(3) sandbox 形式イメージ内のカスタマイズ
--fakeroot オプションにより、管理者権限でイメージを起動していますので、dnf コマンド等でパッケージのインストールが可能です。
注意事項：
dnf コマンドによるパッケージインストールが permission denied で失敗する場合があります。
/usr/lib や/usr/bin のパーミッションに write 権限が無いために発生している可能性がありますの

で、ディレクトリのパーミッションを確認してください。

[実行例]
GNU Fortran パッケージをインストール
Singularity> dnf install gcc-gfortran

/home 上にサンプルプログラムの実行モジュールを作成
Singularity> cd /home
Singularity> vi hello.f90
---
```
program hello
  print *, 'Hello World!'
end program
```
---
Singularity> gfortran -o hello.exe hello.f90

(4) sandbox 形式から再イメージファイル化
[書式] $ singularity build --fakeroot <イメージファイル名> <sandbox 名>

[実行例]
$ singularity build --fakeroot centos8v2.sif centos8

イメージファイルから、カスタマイズ時に作成した実行モジュールが実行できることを確認
$ singularity exec centos8v2.sif /home/hello.exe
INFO:  Converting SIF file to temporary sandbox...
 Hello World!
INFO:  Cleaning up image...

(5) 作業後の削除
[書式] $ rm -rf /tmp/<sandbox 名>

[実行例]
$ rm -rf /tmp/centos8