\Orchestrating a brighter world NEC

大阪大学　サイバーメディアセンター
利用者講習会

コンテナ入門

2021年10月21日
日本電気株式会社
第一官公ソリューション事業部

\Orchestrating a brighter world

NECは、安全・安心・公平・効率という社会価値を創造し、
誰もが人間性を十分に発揮できる持続可能な社会の実現を目指します。

目次

1. コンテナの概要

なぜコンテナを使うのか

コンテナの利点

ソフトウェア環境準備のし易さ

- **rootに近い操作が可能**
  コンテナ内はroot に近い操作が可能。必要なパッケージを**管理者依頼することなくインストール**できます。

- **活用例**
  - ✓ 古いバージョンのGCCを使いたい
  - ✓ 別のLinux ディストリビューションを使いたい

環境依存しにくい実行環境

- **アプリと実行環境をまとめられる**
  アプリケーションを実行環境とセットで同梱することで、**実行の再現性が向上**します。

- **活用例**
  - ✓ コンテナごと別システムに持って行って実行する
  - ✓ バージョンアップ等の影響をうけず実行の再現性を維持

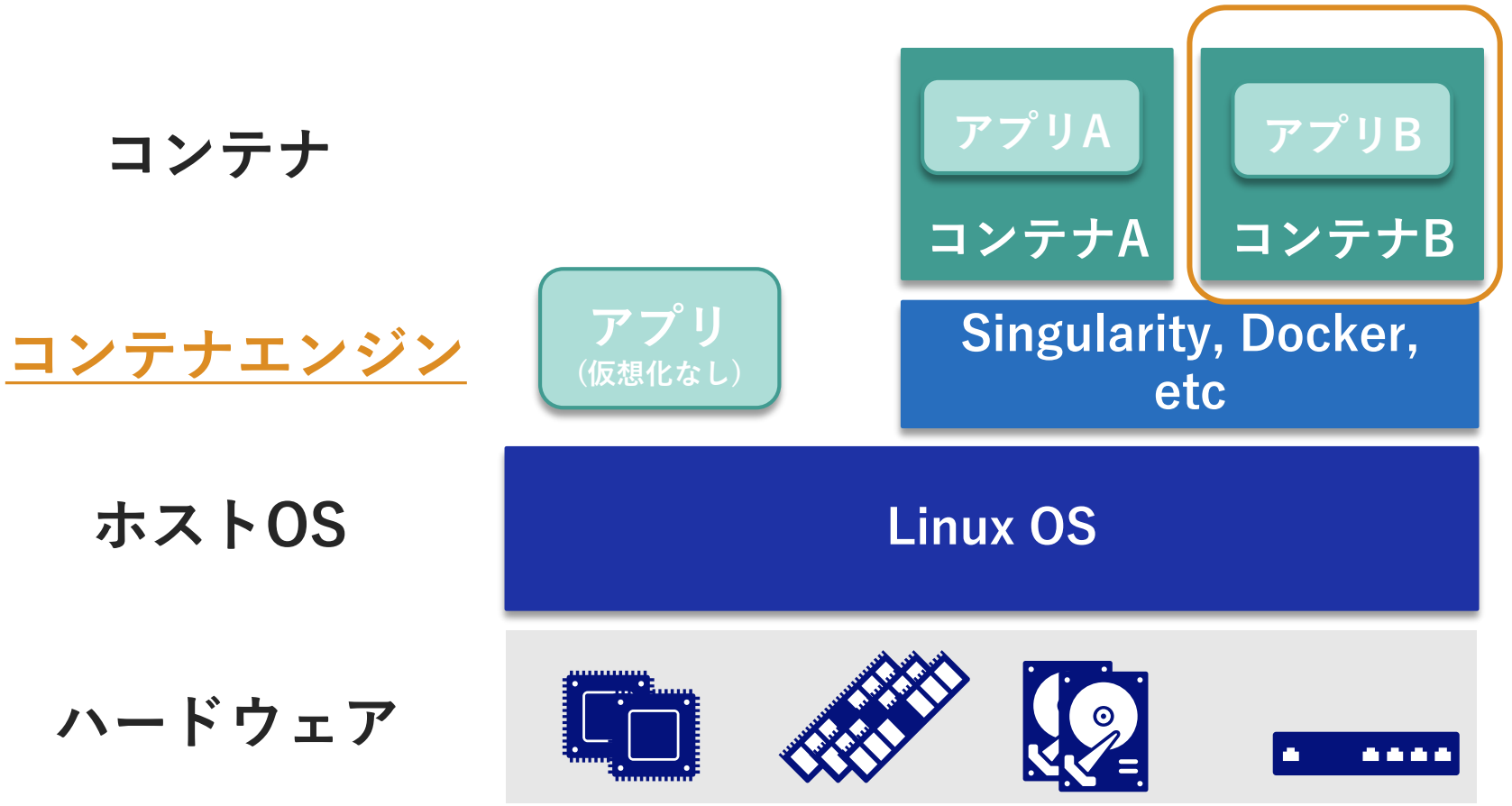
コミュニティ資産の再利用

- **公開されているコンテナ資産を利用可能**
  環境依存しにくいため、**公開イメージをそのまま活用**できる

- **活用例**
  - ✓ Docker HUB のコンテナイメージを使ってアプリを実行する
  - ✓ GPU対応イメージを使い、GPUの煩雑な環境整備の手間を省く

コンテナの仕組み

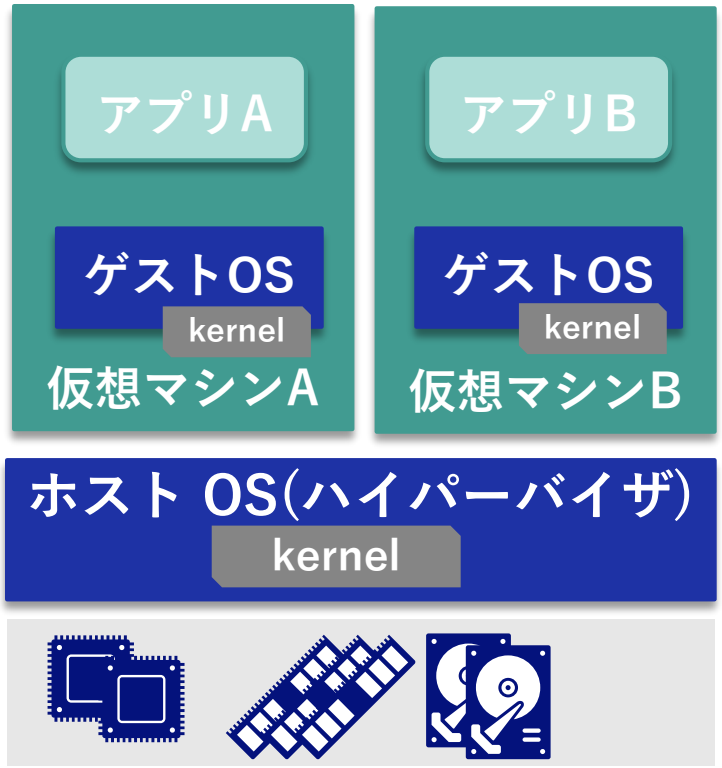
コンテナとはホストOS上に軽量な仮想環境を実現する仕組みです。
root権限がない環境でも、仮想環境を作ることも可能です。



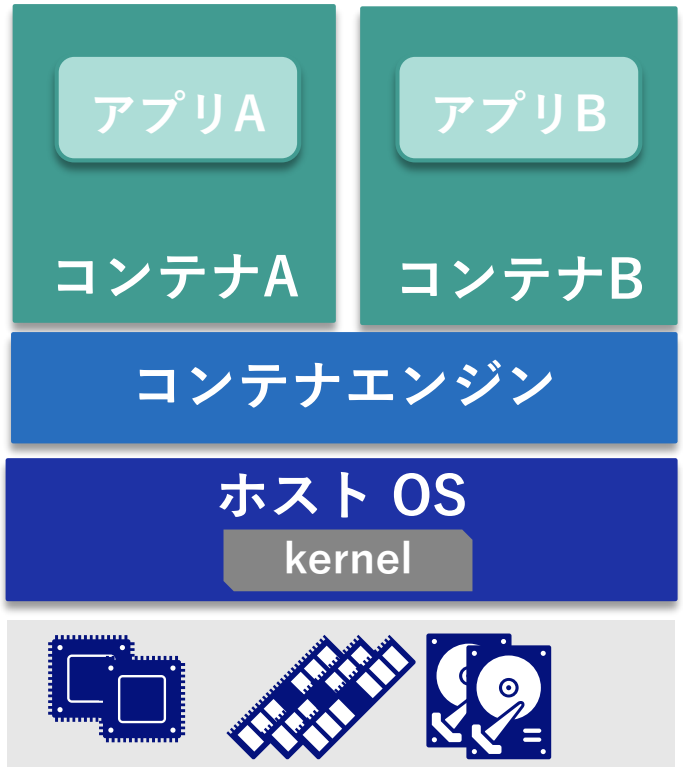
- **コンテナエンジン**
  - コンテナとホストOSの間に入り、環境の分離やシステムコールを制御するソフトウェア

コンテナと仮想マシンの違い

仮想マシンでは、ホストOSのkernelとゲストOSは分離されます。
コンテナでは、OSのkernelをホストとコンテナで共有します。



仮想マシン型仮想化

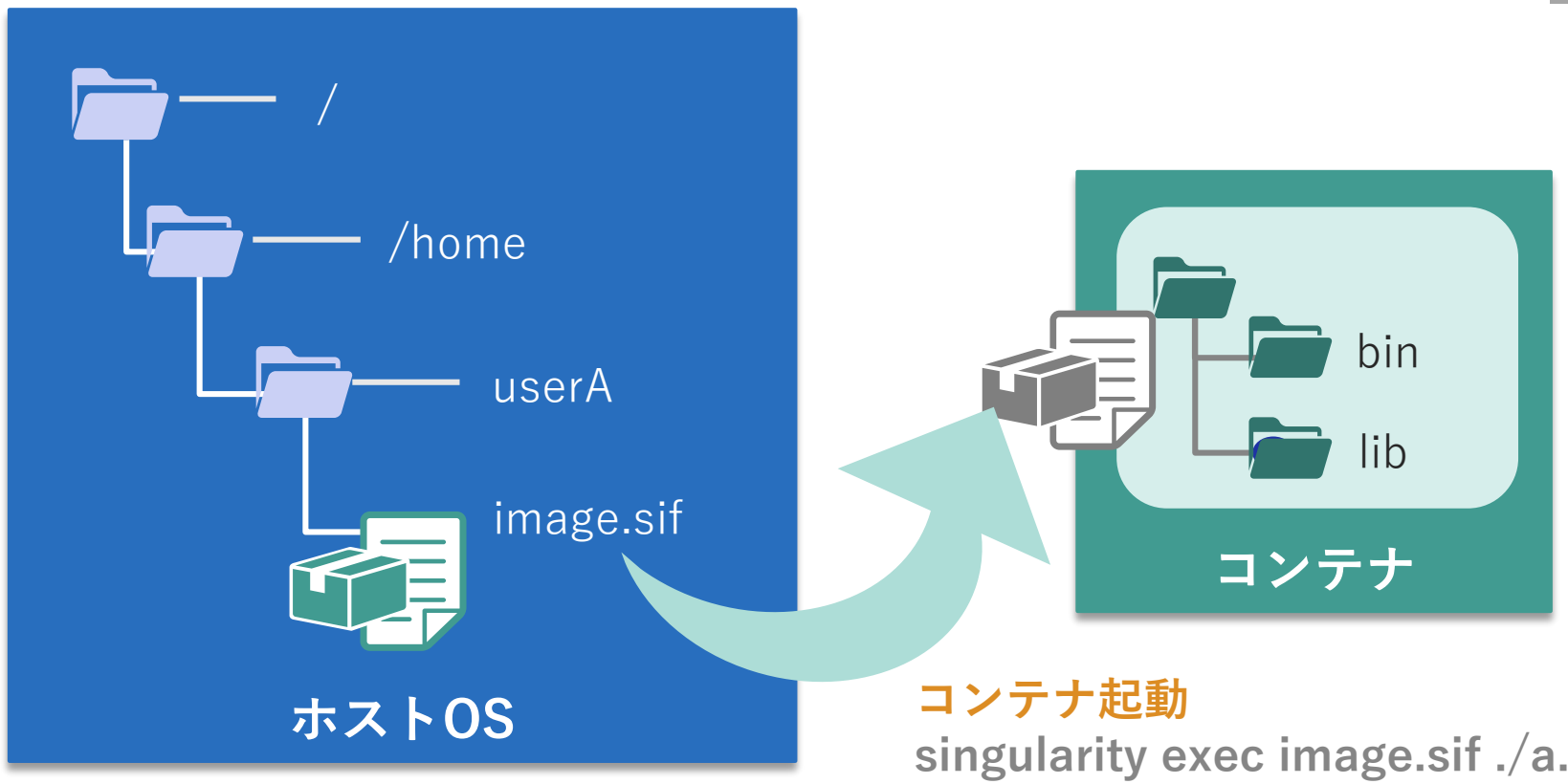


コンテナ型仮想化

- **このためコンテナは、**
  - ✓ 仮想環境が小さく作れる
  - ✓ **ドライバやカーネルモジュールはホストOSを使う**
    →ホストの実装に影響をうける
  - ✓ 異種OSのコンテナは実行できない(例：Linuxの上にWindows)
  - ✓ コンテナ内にOS機能(systemd等)は不要
    →**ホストとして起動しない**ことも多い

コンテナの振る舞い - ファイルシステム

コンテナ内ではファイルシステムがホストOSと分離されます。
イメージ内の独立したファイルシステムを、起動時にマウントします。



- **コンテナ内では、**
  - ✓ イメージを展開し仮想的なファイルシステムをマウント
  - ✓ アプリが必要とする必要最低限のファイルのみ展開

コンテナの振る舞い - その他

コンテナの振る舞いの概要を記載します。
※ 本日のメインのSingularity を使う場合はあまり意識しない内容になります

- **プロセス空間が分離される**
  - コンテナ内からコンテナ外のプロセスを表示できないようにすることが可能
- **ネットワークスタックが分離される**
  - コンテナに、外部通信できないNWやホストOSを経由するNWなど、NW層を仮想化することも可能
- **コンテナのリソース(CPU, メモリ等消費制限が可能**
  - Linuxの cgroup の機能を併用して、コンテナのリソース消費を制限可能
- **権限の分離が可能**
  - コンテナ内でroot 権限を与えつつ、外部影響がある操作(例：NFSマウント)は制限をかけるといった制御が可能

2. Singularity と Docker

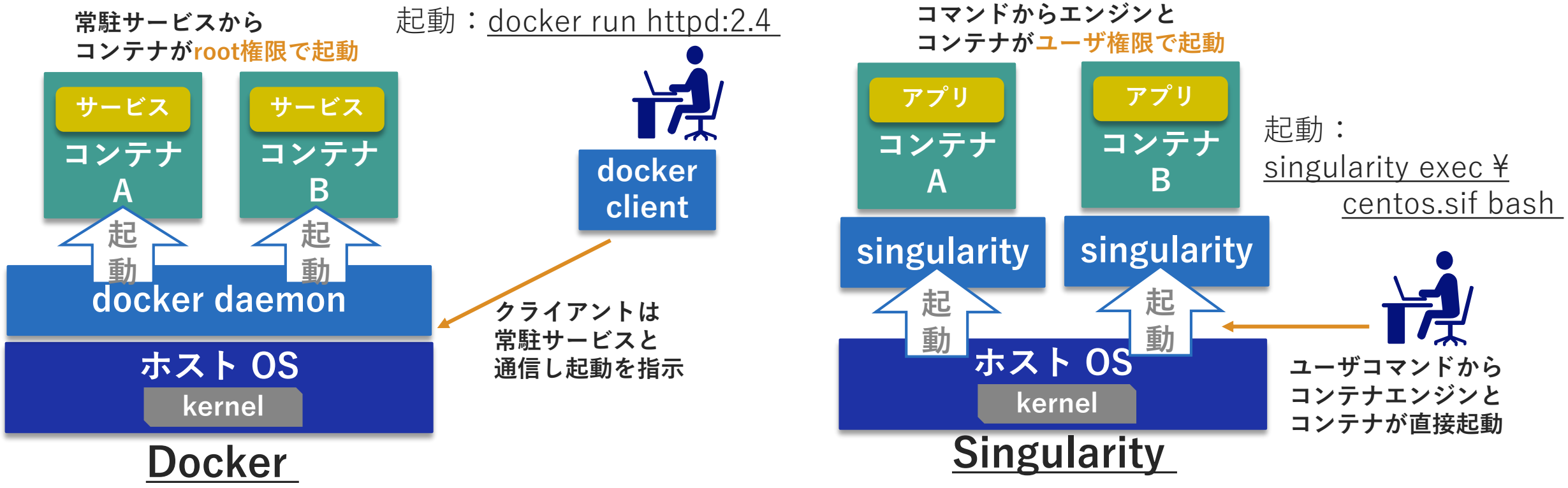
Docker と Singularity

Docker はWEBやクラウドで広く普及しているコンテナ実装です。
Singularityは計算機センター用に開発され、HPC分野で普及しています。

	Docker	Singularity
公式ページ	https://www.docker.com/	https://sylabs.io/singularity/
開発元	Docker, Inc	Lawrence Berkeley National Laboratory
ライセンス	Apache License 2.0	3-clause BSD License
公式レジストリ	https://hub.docker.com/	https://cloud.sylabs.io/library
特徴	コンテナエンジンが常駐サービスとして起動 サービスを介してroot権限でコンテナが起動 豊富なエコシステム	ユーザコマンドとしてコンテナエンジンが起動 ユーザ権限でコンテナが起動 **dockerfile 利用可能**

Docker と Singularity

コンテナエンジンは、Dockerでは常駐サービスとして起動するのに対し、Singularity ではユーザコマンドとして実装されています。



ユーザが自分の権限でコンテナを起動できるため、
root権限を使えないスパコンシステムでSingularityによるコンテナ利用が普及しています。

SQUID におけるコンテナ対応

SQUIDでは、標準のコンテナ利用方法として、Singularity が利用可能です。Docker コンテナは特殊用途のため、標準利用はできません。

	Docker	Singularity
バージョン	20.10.3-3	3.7-2
利用方法	特殊なコンテナ起動(標準利用不可)	**フロントエンド上で利用可能**
利用可能イメージ	管理者による事前登録が必要	利用者自身で**持ち込みと実行が可能**

以降では、Singularity の利用方法に絞ってご説明します。

※ docker のrootless やsingularity のinstance等、双方の長所を取り込むような機能も増えてきており、特徴に合致しないケースも出てきています

3. Singularity の基本操作

Singularity の基本操作

Singularity の主要コマンド

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

コマンド名	説明	備考
build	イメージのビルド(作成や更新)を実施します	5章 で説明予定
cache	キャッシュの操作	cache clean でキャッシュクリア
exec	コンテナ内で指定コマンドを実行します	6章 で説明予定
help	コマンドヘルプを表示します。	singularity help <command>でコマンド詳細
inspect	イメージのメタデータを表示します。	
pull	URI指定して、イメージを取得します。	4章 で説明予定
push	イメージをアップロードします。	
run	コンテナ内の標準コマンドを実行します	
search	ライブラリからコンテナを検索します	
shell	コンテナ内でシェルを起動します	
sif	イメージファイルの操作に使用します	

Singularity の基本操作

Singularity コマンド実行例です。

- ✓ **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 gromacs**

- ✓ **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 gromacs**

- ✓ **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 gromacs**

4. コンテナイメージの準備

コンテナイメージの準備

SQUIDフロントエンド上で可能な、イメージ準備方法を説明します。

- **ローカルのワークステーションからシステム内に転送する**
  - コンテナイメージは、通常のファイルと同じようにscpコマンド等で転送が可能です。
- **SQUID ローカルレジストリからイメージを取得する**
  - SQUIDのローカルレジストリで公開しているコンテナイメージを取得する
- **Singularity Libraryからイメージを取得する**
  - Singularity Library で公開されているコンテナイメージを取得する
- **Docker Hubからイメージを取得する**
  - Docker Hub で公開されているコンテナイメージを取得する
- **NVIDIA GPU CLOUD(NGC)からイメージを取得する**
  - NVIDIA GPU CLOUD で公開されているコンテナイメージを取得する

ローカルのワークステーションからシステム内に転送する

ローカルのワークステーションからコンテナイメージをコピー可能です。

- ✓ イメージのビルド作業は、root権限を持っていないフロントエンド上では操作の制限を受けるケースがあります。root権限のある手元のワークステーションでビルドした際は、本手順で転送することになります。
- ✓ 以下は、SCPによる転送例です。ファイル転送の詳細手順については、「SQUIDの利用方法 >ファイル転送方法（ローカル⇔SQUID）」をご参照ください。
  http://www.hpc.cmc.osaka-u.ac.jp/system/manual/squid-use/transfer/

1. SCPコマンドでのファイル転送

```
$ scp（イメージファイル名）（ユーザ名）@squidhpc.hpc.cmc.osaka-u.ac.jp:~/
```

ローカルのワークステーションからシステム内に転送する

ローカルのワークステーションからコンテナイメージをコピー可能です。

✓ **以下は、SCPによる転送例です。ファイル転送の詳細手順については、「SQUIDの利用方法 >ファイル転送方法（ローカル⇔SQUID）」をご参照ください。**
http://www.hpc.cmc.osaka-u.ac.jp/system/manual/squid-use/transfer/

2. **パスワード入力**
   パスワードを聞かれますので、利用者管理システムと同じパスワードを入力します。

```
xxxxxx@squidhpc.hpc.cmc.osaka-u.ac.jp's password: XXXXXXXX
```

3. **ワンタイムパスワード入力**
   Verification code に、2段階認証アプリから取得したワンタイムパスワードを入力します。

```
Verification code: ***** (ワンタイムパスワードを入力)
```

4. **SCP通信開始**
   ファイル転送が行われ、転送状況、転送ファイルサイズ、転送時間が表示されます。

```
myimage.sif 100% |*****************************| 1326 00:01
```

SQUID ローカルレジストリからイメージを取得する

SQUID固有設定があるコンテナをローカルレジストリに公開しています。

- ✓ **ローカルのレジストリからコンテナイメージを取得する**

[書式] $ **singularity build** <イメージファイル名> ¥
oras://cntm:5000/<コンテナイメージパス>:<tag名>

[実行例] $ **singularity build test.sif** ¥
oras://cntm:5000/master_image/test:1.0

Singularity Libraryからイメージを取得する

Singularity Libraryからイメージを取得可能です。

1. イメージを検索する

[書式] $ **singularity search** <検索クエリ>

[実行例] $ **singularity search** **centos**

```
...
        library://emmeff/centos/centos:8
...
```

2. イメージを取得する

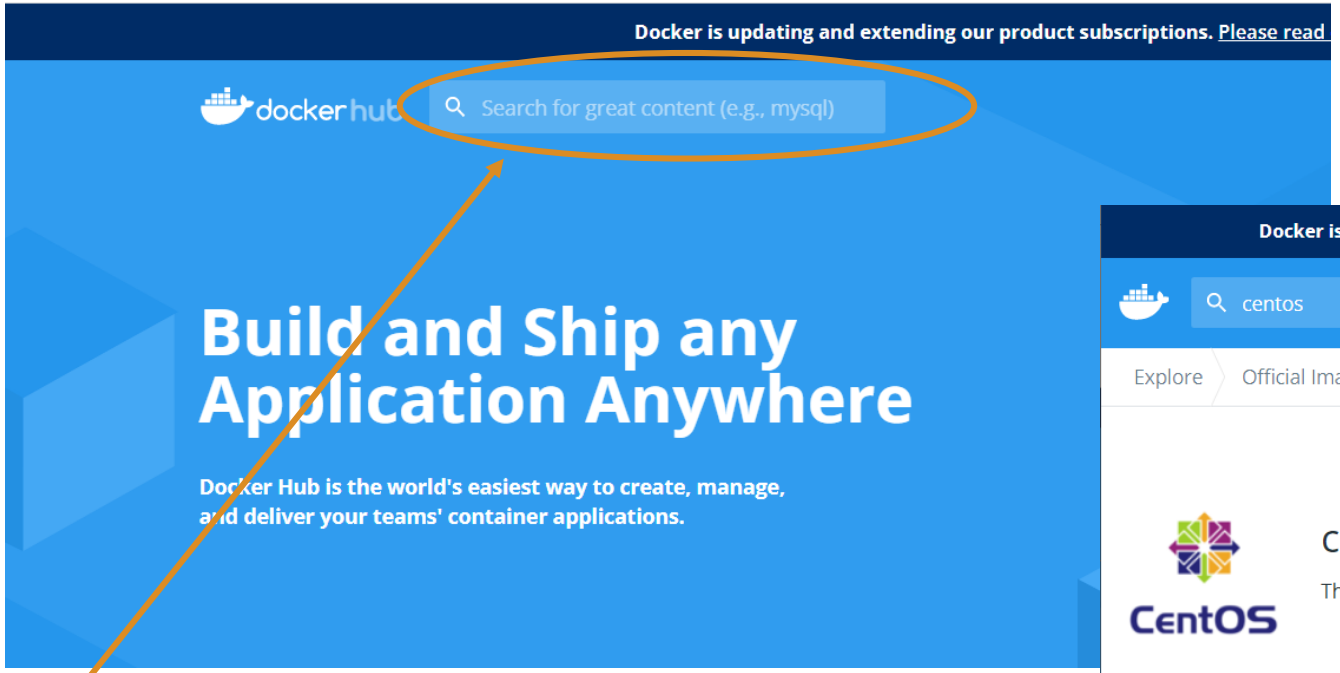
[書式] $ **singularity build** <イメージファイル名> **library://**<イメージパス>:<tag名>

[実行例] $ **singularity build** **centos.sif** **library://emmeff/centos/centos:8**

Docker Hubからイメージを取得する

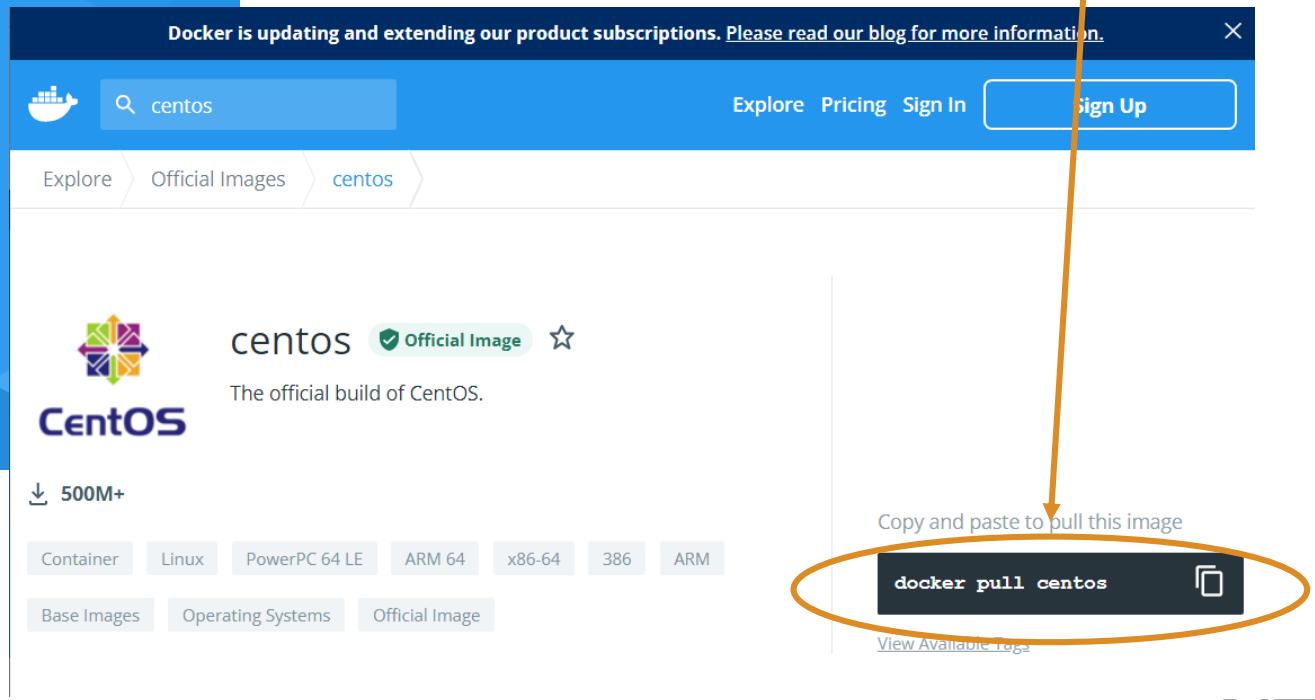
Docker Hub 上のイメージをSingularity に変換して利用可能です。

https://hub.docker.com/



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）



Docker Hubからイメージを取得する

Docker Hub 上のイメージをSingularity に変換して利用可能です。

- ✓ **Docker Hub からイメージを取得**

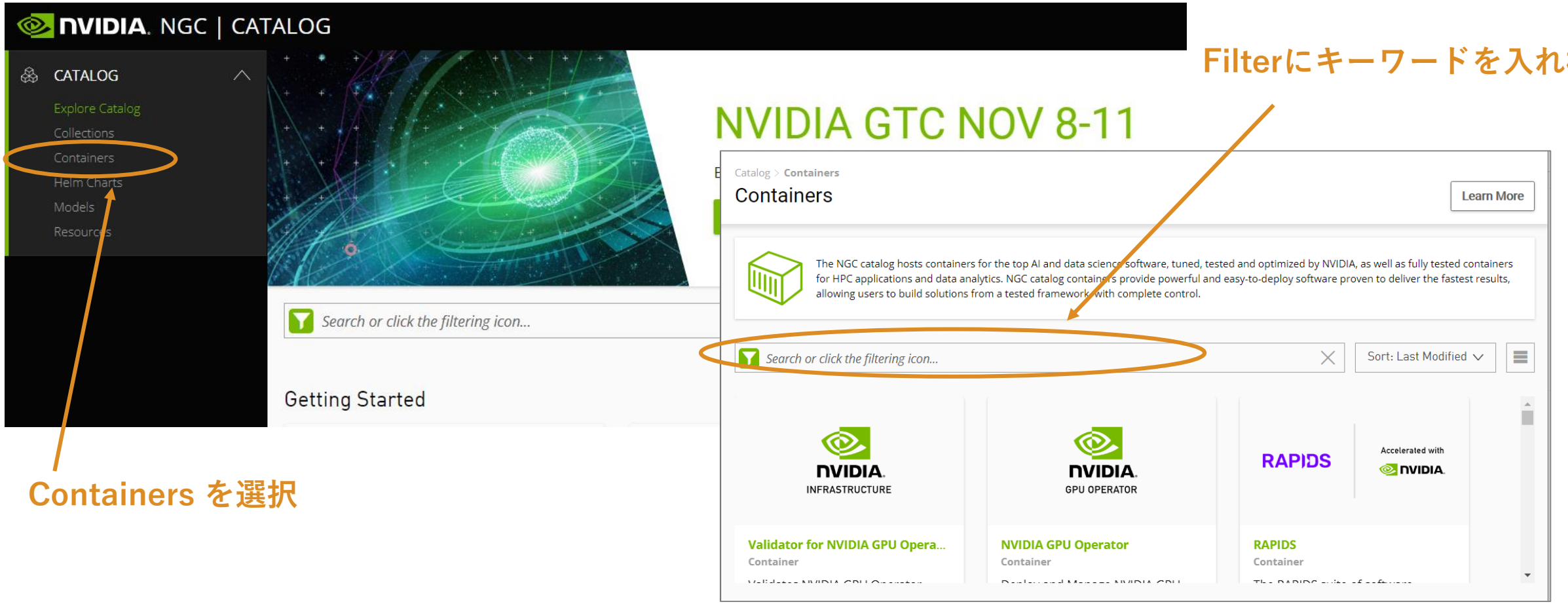
[書式] $ **singularity build <イメージファイル名> ¥**
docker://docker.io/<コンテナイメージパス>:<tag名>

[実行例] $ **singularity build centos.sif ¥**
docker://docker.io/centos:latest

NVIDIA GPU CLOUD(NGC)からイメージを取得する

NGC上のGPU向けイメージを変換して利用可能です。

https://ngc.nvidia.com/catalog (アカウント登録が必要です)



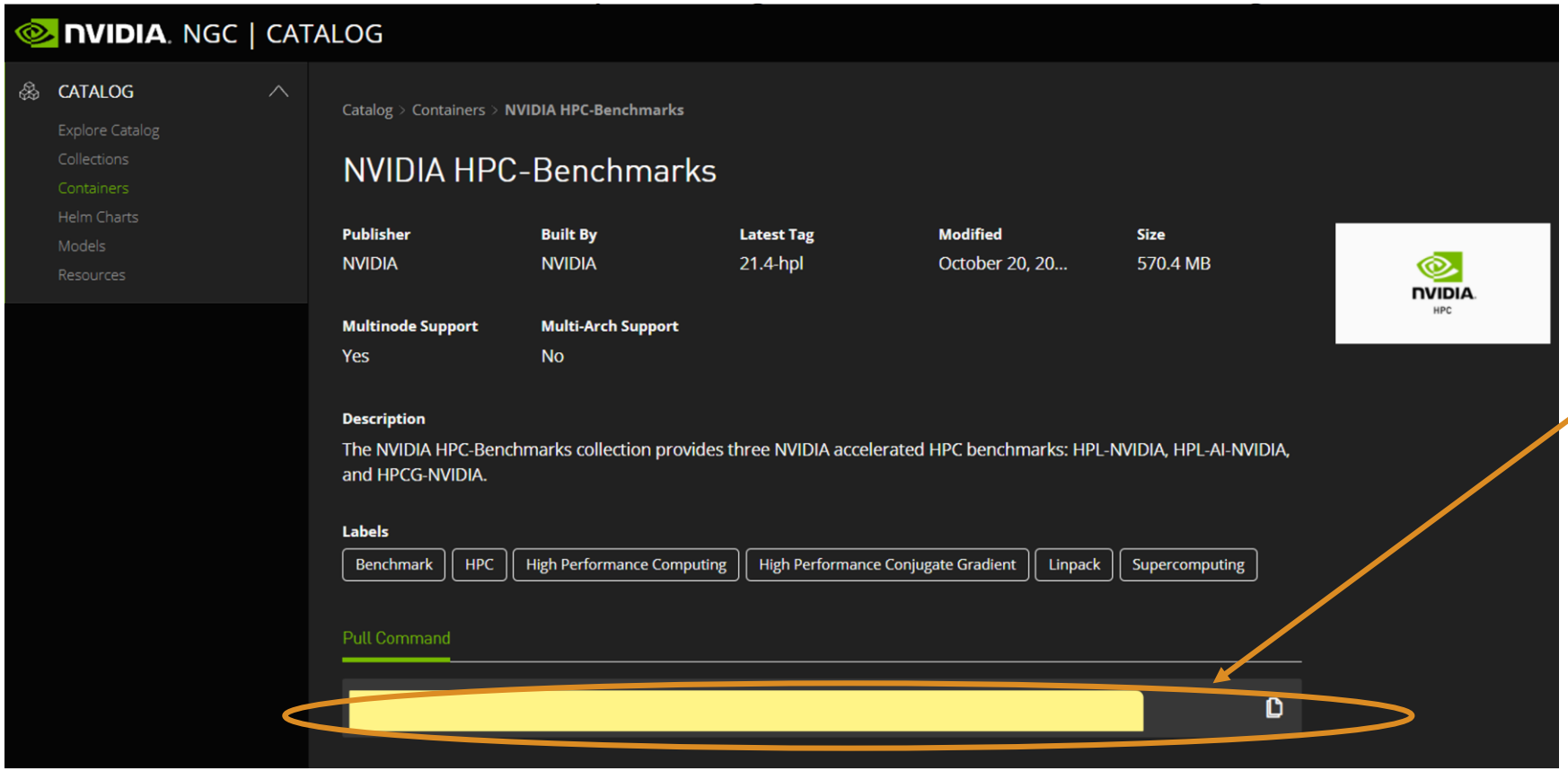
Filterにキーワードを入れ検索

Containers を選択

NVIDIA GPU CLOUD(NGC)からイメージを取得する

NGC上のGPU向けイメージを変換して利用可能です。

https://ngc.nvidia.com/catalog



pull コマンドのイメージのパスを確認します。(確認はログインが必要)

NVIDIA GPU CLOUD(NGC)からイメージを取得する

NGC上のGPU向けイメージを変換して利用可能です。

1. **環境変数の設定**

```
$ export SINGULARITY_DOCKER_USERNAME='$oauthtoken'
$ export SINGULARITY_DOCKER_PASSWORD=<API Key>
```

※ユーザ名の $oauthtoken は固定で、API KeyはNGCサイト内で生成したKeyを入力します。
NGC Catalog Containers
https://ngc.nvidia.com/catalog/containers

2. **コンテナの取得**

```
$ singularity build hpc-benchmarks:21.4-hpl.sif ¥
                docker://nvcr.io/nvidia/<コンテナイメージパス>:<tag名>
```

5. コンテナイメージの作成とカスタマイズ

コンテナイメージの作成とカスタマイズ

SQUIDフロントエンド上可能な、イメージ更新方法を説明します。

Sandbox 経由でカスタマイズする方法

- sandbox としてコンテナの内容をファイルに展開し、ファイルを直接操作することが可能です。

定義ファイルにカスタマイズ内容を記載する方法

- カスタマイズ内容を定義ファイル(def)に記述し、カスタマイズする方法です。

Sandbox 経由でカスタマイズする方法

コンテナの内容をsandbox と呼ばれる形式で変換し、ファイルを直接操作します。

1. **sanbox の作成**

[書式] $ **singularity build -f --sandbox --fix-perms <sandbox名> <イメージファイル名>**

✓ **ホーム領域でsandboxを作成する場合**

```
$ cd ~/mySandbox
$ singularity build -f --sandbox --fix-perms   test    test.sif
```

✓ **拡張領域・高速領域でsandboxを作成する場合**

```
$ cd /sqfs/work/<グループ名>/<利用者番号>/mySandbox
$ newgrp <グループ名>
$ singularity build -f --sandbox --fix-perms  test   test.sif
```

Sandbox 経由でカスタマイズする方法

コンテナの内容をsandbox と呼ばれる形式で変換し、ファイルを直接操作します。

2. コンテナの起動

[書式] $ **singularity run -f -w <sandbox名>**
[実行例] $ **singularity run -f -w test**
Singularity>

※ 上記のプロンプトより、dnfやpip等のパッケージ操作や、コンテナ内のファイル修正が可能となります。

3. 修正後のイメージファイル化

[書式] $ **singularity build -f <イメージファイル名> <sandbox名>**
[実行例] $ **singularity build -f test.sif test**

定義ファイルにカスタマイズ内容を記載する方法

定義ファイルに修正内容を記載し、ビルド時にカスタマイズします。

1. 定義ファイルの作成例 (sample.def)

```
Bootstrap: oras
From: cntm:5000/master_image/test:1.0
%files
        ./test.conf /opt/test.conf
        ./test_start.sh /opt/test_start.sh
%post
        dnf install -y net-tools
        chmod 755 /opt/test_start.sh
%runscript
        /opt/test_start.sh
```

- `Bootstrap: oras` / `From: cntm:5000/master_image/test:1.0` : ベースイメージの種類、場所
- `%files` 以下 : ホストOSからコンテナにコピーしたいファイル
- `%post` 以下 : カスタマイズ用のコマンド
- `%runscript` 以下 : コンテナ起動時に自動実行する処理

2. 定義ファイルを使用して、コンテナイメージをビルド

[書式] $ **singularity build -f** <イメージファイル名> <定義ファイル名>
[実行例] $ **singularity build -f** test.sif sample.def

定義ファイルにカスタマイズ内容を記載する方法

定義ファイルの記載概要です。詳細はオンラインマニュアルを参照ください。 https://sylabs.io/guides/3.7/user-guide/definition_files.html#

Bootstrap句

- 主要なBootstrap 句は以下です。

Bootstrap	説明	From 句
library	Singularity Library上のイメージを指定する	<entity>/<collection>/<container>:<tag> 例： emmeff/centos/centos:8
docker	Docker Hub 上のイメージを指定する	<registry>/<namespace>/<container>:<tag>@<digest> 例：docker.io/centos:latest
oras	OCI対応したレジストリを指定する	<registry>/<namespace>/<container>:<tag> 例：cntm:5000/master_image/test:1.0
localimage	ローカルのイメージファイルを指定する	/path/to/container/file/or/directory 例： ./test.sif
scratch	FROM指定なし	

定義ファイルにカスタマイズ内容を記載する方法

定義ファイルの記載概要です。詳細はオンラインマニュアルを参照ください。 https://sylabs.io/guides/3.7/user-guide/definition_files.html#

%setup

- ビルド開始時に**ホスト上で実行する**コマンドを記述します。

```
%setup
        touch /file1
```

%files

- ビルド開始時にホストのファイルをコンテナ内に配置します。配置先が無い場合はカレントディレクトリに配置されます。

```
%files
        /file1
        /file1   /opt
```

%post

- イメージの準備後に、**コンテナ内で実行する**コマンドを記述します。

```
%post
        dnf -y install net-tools
        mkdir   /opt/dir1
```

6. SQUID 上のコンテナジョブの基本操作

汎用CPU計算環境でのコンテナジョブ実行方法

汎用CPU計算環境でノード内スレッド並列プログラム実行する際のジョブスクリプト

- ✓ **汎用CPU計算環境用のサンプルスクリプト**

```
#!/bin/bash
#-------qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -v OMP_NUM_THREADS=76

#-------Program execution -----------
cd $PBS_O_WORKDIR
singularity exec --bind `pwd` image.sif ./a.out
```

※ OMP_NUM_THREADS 環境変数がコンテナ内にも引き継がれ、スレッド並列実行されます。

GPGPU計算環境でのコンテナジョブ実行方法

GPGPU計算環境でノード内スレッド並列プログラム実行をするジョブスクリプト。

✓ **GPGPU計算環境用のサンプルスクリプト**

```
#!/bin/bash
#-------qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -l gpunum_job=8
#PBS -v OMP_NUM_THREADS=76

#-------Program execution -----------
cd $PBS_O_WORKDIR
singularity exec --nv --bind `pwd` image.sif   ./a.out
```

nv オプションにてGPU利用を指定

※ OMP_NUM_THREADS 環境変数がコンテナ内にも引き継がれ、スレッド並列実行されます。

コンテナの実行方法の概要

コンテナの実行時の注意点について説明します。

■ 実行コマンド

- exec コマンドでは、コンテナ開始時に実行するコマンドを指定します。**指定するコマンドは、コンテナ内の実行コマンド**となっている点にご注意ください。
- コマンドがパス指定なしであれば、コンテナ内のPATH環境変数で探索されたコマンドが実行されます。

[書式] $ **singularity exec <イメージファイル名> <コンテナ内のコマンド>**
[実行例] $ **singularity exec centos.sif hostname**

※ 実行例はcentos.sif コンテナイメージ内のhostname コマンドを実行しています。

コンテナの実行方法の概

コンテナの実行時の注意点について説明します。

■ 環境変数

- **コンテナ外での環境変数はコンテナ内に引き継がれます**。
  コンテナのメタデータで明示的に定義されている環境変数はコンテナ側定義に従います。
- メタデータで定義されている環境変数を上書きする場合は、--env オプションや、--env-fileオプションを使います。
- --env オプションによる個別指定

```
$ singularity exec --env MYVAR="My Value!" centos.sif  myprog.exe
```

- --env-file による一括指定

```
$ cat myenvfile
MYVAR="My Value!"
$ singularity exec --env-file=myenvfile  centos.sif  myprog.exe
```

環境変数に関する詳細な内容は、下記の公式ドキュメントを参照ください。
https://sylabs.io/guides/3.7/user-guide/environment_and_metadata.html

コンテナ実行の注意点

コンテナの実行時の注意点について説明します。

ホストOSのマウント

- **下記のディレクトリは標準でマウント**されており、コンテナ内でも同じパスで利用が可能です。
  ホームディレクトリ：/sqfs/home/(利用者番号)　　テンポラリ領域：/tmp
- --bind オプションにて、特定のディレクトリをマウント可能です。
  --bind <ホストOSのパス>:<コンテナ内のパス>:<モード>

例：ホストのhome にあるプログラム(a.out)を実行する

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

※上記例では、カレントディレクトリが、コンテナ内でhomeに移動しています。

例：拡張領域上のディレクトリに置かれたプログラム(a.out)を実行する

```
$ cd /sqfs/work/(グループ名)/(利用者番号)
$ singularity exec --bind `pwd`  centos.sif  ./a.out
```

※上記例では、コンテナ外での環境変数PWDが、コンテナ内へも引き継がれており、コンテナ内のカレントディレクトリが、拡張領域上のディレクトリになっています。

6. SQUID 上のコンテナの応用

コンテナの応用

SQUID上のコンテナの応用的な使い方として、下記を説明します。

SX-Aurora TSUBASA の利用

- ベクトルノードで、VE(ベクトルエンジン)を使ったコンテナの実行方法

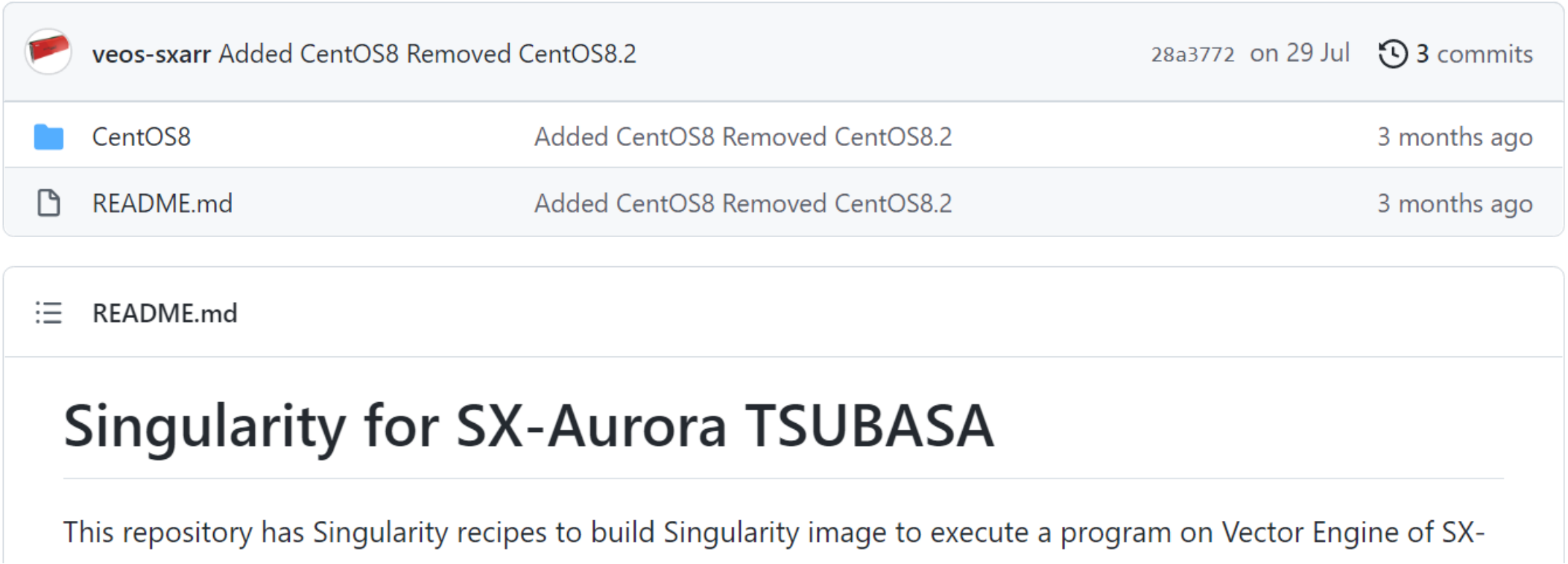
マルチノードジョブの実行

- MPIを利用してマルチノードのコンテナジョブを実行する際の注意点

SX-Aurora TSUBASA の利用

SX-Aurora TSUBASA をSingularity から利用する方法は、Git-HUB上で公開されています。

https://github.com/veos-sxarr-NEC/singularity



SX-Aurora TSUBASA の利用

コンテナイメージビルド時の注意事項です。

シングル版とMPI対応版の2種類のイメージのビルド手順があります

- Build the singularity image of VEOS
- Build the singularity image of NEC MPI

下記の互換性の注意事項があります。

- 実行ホストのVEOS バージョン >= コンテナ内のVEOSバージョン
- **実行ホストのNEC MPIバージョン >= コンテナ内のNEC MPI バージョン** 【MPI対応版のみ】
- 実行ホストのOFEDバージョン == コンテナ内のOFED バージョン 【MPI対応版のみ】
- **コンテナ内のNEC MPI バージョン >= プログラムコンパイルホストのNEC MPIバージョン** 【MPI対応版のみ】
- コンテナ内のNEC SDK バージョン >= プログラムのコンパイルホストのNEC SDKバージョン

SQUID の2021年11月時点のバージョン

VEOS : 2.7.6-1　NEC MPI : 2.16.0　NEC SDK : 3.2.1　Mellanox OFED : 4.9-0.1.7.0

SX-Aurora TSUBASA の利用

シングル版のビルド例です

1. レジストリをクローン

```
$ git clone https://github.com/veos-sxarr-NEC/singularity.git
```

2. release パッケージの取得

```
$ cd singularity/CentOS8
$ curl -O https://www.hpc.nec/repos/TSUBASA-soft-release-2.4-1.noarch.rpm
```

3. 定義ファイルを必要に応じて修正

```
$ vi Singularity
```

4. イメージをビルド

```
$ singularity build --fakeroot veos.sif Singularity
```

SX-Aurora TSUBASA の利用

定義ファイル(Singularity) の内容

```
Bootstrap: docker
From: centos:8.3.2011
%files
        dnf.conf /etc/dnf
        TSUBASA-soft-release-*.noarch.rpm /etc/yum.repos.d
        TSUBASA-repo.repo /etc/yum.repos.d
        TSUBASA-restricted.repo /etc/yum.repos.d

%post
        mv /etc/yum.repos.d/TSUBASA-repo.repo /etc/yum.repos.d/TSUBASA-repo.repo.bk
        mv /etc/yum.repos.d/TSUBASA-restricted.repo /etc/yum.repos.d/TSUBASA-restricted.repo.bk
        yum -y install /etc/yum.repos.d/TSUBASA-soft-release-*.noarch.rpm
        mv /etc/yum.repos.d/TSUBASA-repo.repo.bk /etc/yum.repos.d/TSUBASA-repo.repo
        mv /etc/yum.repos.d/TSUBASA-restricted.repo.bk /etc/yum.repos.d/TSUBASA-restricted.repo
        yum clean all
        yum -y group install ve-container nec-sdk-runtime -d 10
        sed -i -e "s|username=.*|username=|" -e "s|password=.*|password=|" ¥
          /etc/yum.repos.d/TSUBASA-restricted.repo
```

**システム環境に合わせて
インストールバージョンを
調整する必要があります。**

SX-Aurora TSUBASA の利用

Git Hubの内容を基に、SQUID 用にバージョンを揃えたものを配置しました。

- ✓ **シングル版**

```
$ singularity pull oras://cntm:5000/master_image/lecture/sq-veos.sif:1.0
```

- ✓ **MPI対応版**

```
$ singularity pull oras://cntm:5000/master_image/lecture/sq-veos-mpi.sif:1.0
```

ご興味がある方は、お試しください。

SX-Aurora TSUBASA の利用

VEを利用するコンテナジョブでHPCGを実行するジョブスクリプト

```
#!/bin/bash
#PBS -q SQUID
#PBS --group=G01234
#PBS -T necmpi
#PBS -l coresz_prc=10
#PBS -l elapstim_req=00:10:00
#PBS --venode=16
#PBS -j o
#PBS -N cHPCG

export HPCG_MATFMT=2
export HPCG_REORDER=0
export HPCG_GSALG=1
module load BaseVEC
IMAGE=~/container/sq-veos-mpi.sif

cd $PBS_O_WORKDIR

date
mpirun -venode -np 16 /usr/bin/singularity exec --bind /var/opt/nec/ve/veos --bind `pwd` $IMAGE ./xhpcg
date
```

※ 16VEの指定

※ mpirun の起動プログラムに
singularity を指定

HPCGの測定結果

- **HPCG（短時間のテストです。）**

項目	コンテナ無し	コンテナあり
GFLOP/s Summary::Total with convergence overhead=	347.193	347.461
GFLOP/s Summary::Total with convergence and optimization phase overhead=	312.884	313.349

マルチノードジョブの実行

公式マニュアルでは、2つのコンテナの作成方針が提示されています。

https://sylabs.io/guides/3.7/user-guide/mpi.html

ハイブリッド型コンテナ

- Infiniband (OFED) のライブラリや、MPIライブラリなどをコンテナ内に同梱する方式です。
- ソフトウェアスタックの調整は少なくなりますが、下記の観点で互換性問題が出る可能性があります。
  Infiniband ドライバ　と　Infiniband ライブラリ間の互換性
  スケジューラが起動するMPI Luncher(orte, hydra)　と　MPIライブラリ間の互換性

ホストバインド型コンテナ

- Infiniband (OFED) のライブラリや、MPIライブラリをコンテナ内にいれず、ホストOSのものを利用する方式です。
- 環境が変わるごとに、コンテナ内アプリケーションの再コンパイルが発生します。

以降では、ホストバインド型コンテナの実行方法を説明をします。

マルチノードジョブの実行

ホストライブラリを利用し、マルチノードジョブを実行する例です。

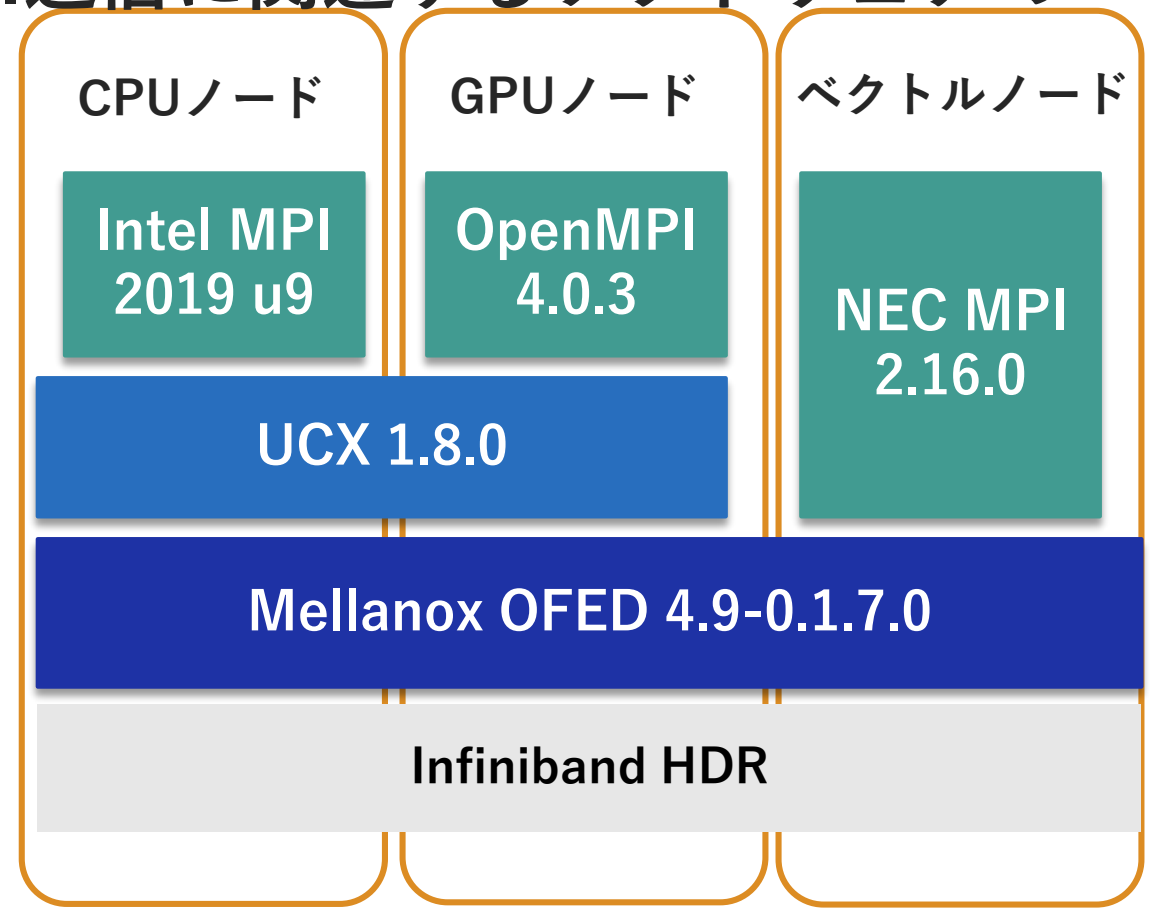
```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:01:00
#PBS -b 2
#PBS -T intmpi
#PBS -j o
#PBS -N cIMB

#------- Program execution -----------
export SINGULARITY_BINDPATH="/system,/usr/lib64:/usr/lib64/host,/etc/libibverbs.d"
export LD_LIBRARY_PATH=/usr/lib64:/usr/lib64/host

module load BaseCPU/2021

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 2 -ppn 1 singularity exec ¥
      --env LD_LIBRARY_PATH=${LD_LIBRARY_PATH} ¥
      imb-test.sif /opt/imb/IMB-MPI1 PingPong
```

※ bindオプションを、環境変数で指定しています。
ホストOSのlib64 を別パスでコンテナへバインドしています。

※ 別パスの lib64 の読込を明示するために、
LD_LIBRARY_PATHをコンテナへ渡しています。

マルチノードジョブの実行

ハイブリッド型コンテナとする場合は、互換性問題回避のため
バージョンをできる限り合わせる必要があります。
SQUID のMPI通信に関連するソフトウェアのバージョン情報です。

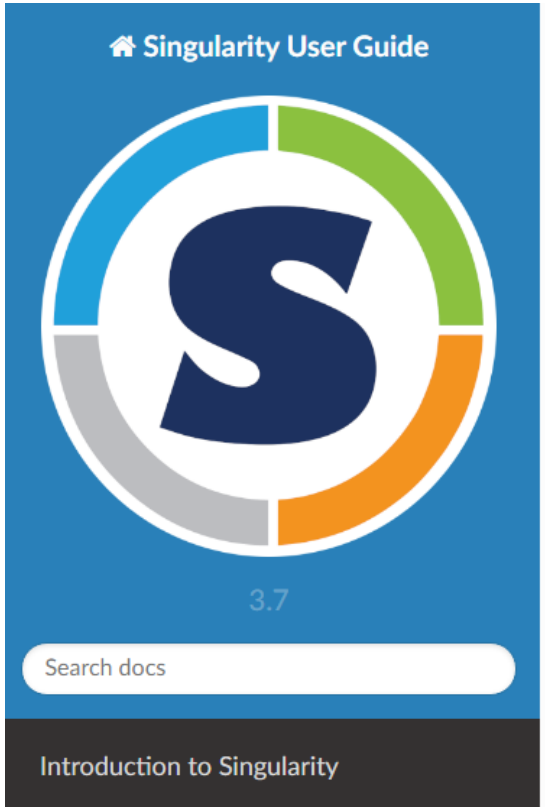


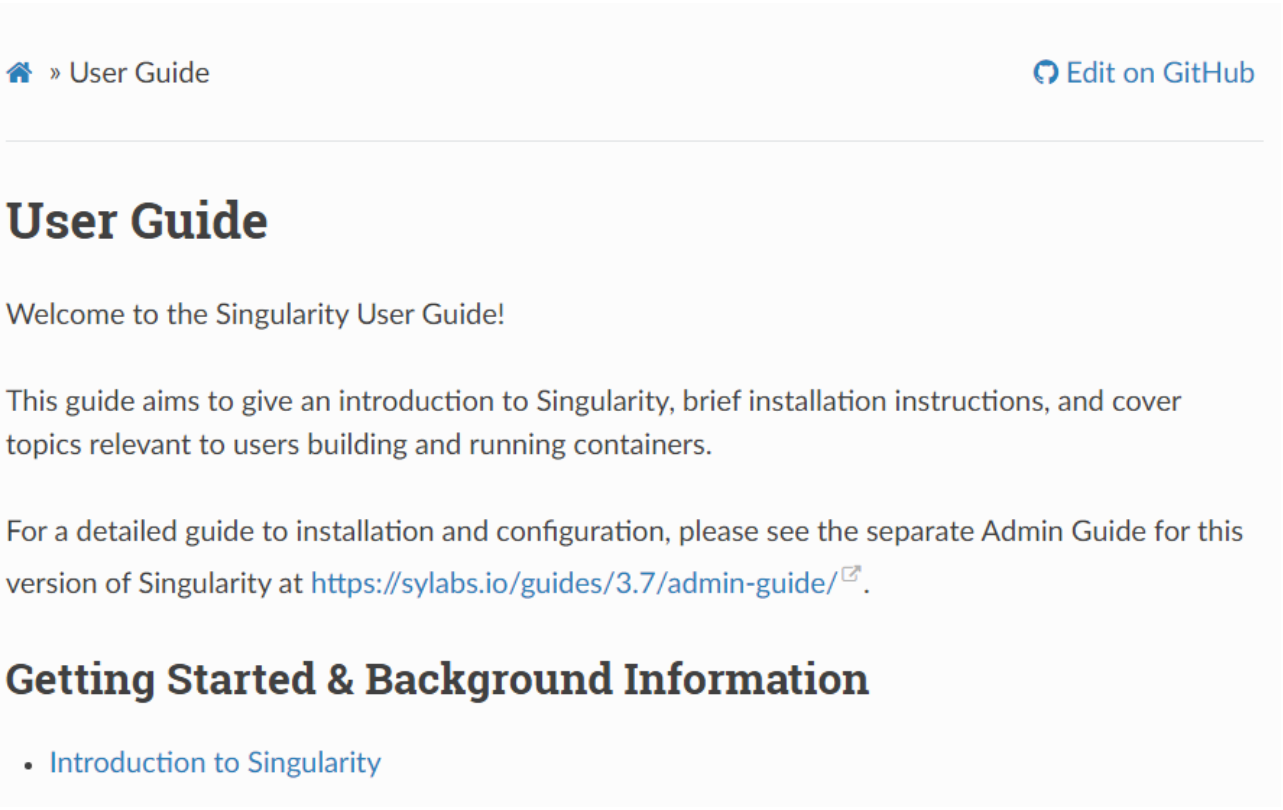
8. 情報参照先

情報参照先 - Singulatiry

Singularity はオンラインマニュアル(英語)の整備が進んでいます。
「Singularity 3.7 User's Guide」を参照ください。

https://sylabs.io/guides/3.7/user-guide/index.html





情報参照先 - SQUIDの利用全般

SQUIDの利用全般に関するマニュアルは、センターホームページで、順次更新されています。「SQUIDの利用方法」を参照ください。

http://www.hpc.cmc.osaka-u.ac.jp/system/manual/squid-use/


Orchestrating a brighter world
NEC