容器技术在超算运维中的经验分享

上海交通大学高性能计算中心
程盛淦

2020年11月

目录

问题背景 – 多集群/硬件异构管理上的挑战

以上海交通大学校级计算平台二期为例，拥有多个硬件异构平台

我们在运维中面临的困难：如何管理好这些异构计算平台？

高性能计算平台

- CPU：26240 核
- 内存：125 TB
- 网络：Intel OPA
- 面向理工生医科研应用

人工智能平台

- 8台DGX-2
- 显卡：128块V100
- 内存：12 TB
- 网络：Mellanox
- 面向人工智能应用

ARM HPC平台

- CPU：～1万核
- 内存：～20 TB
- 网络：Mellanox

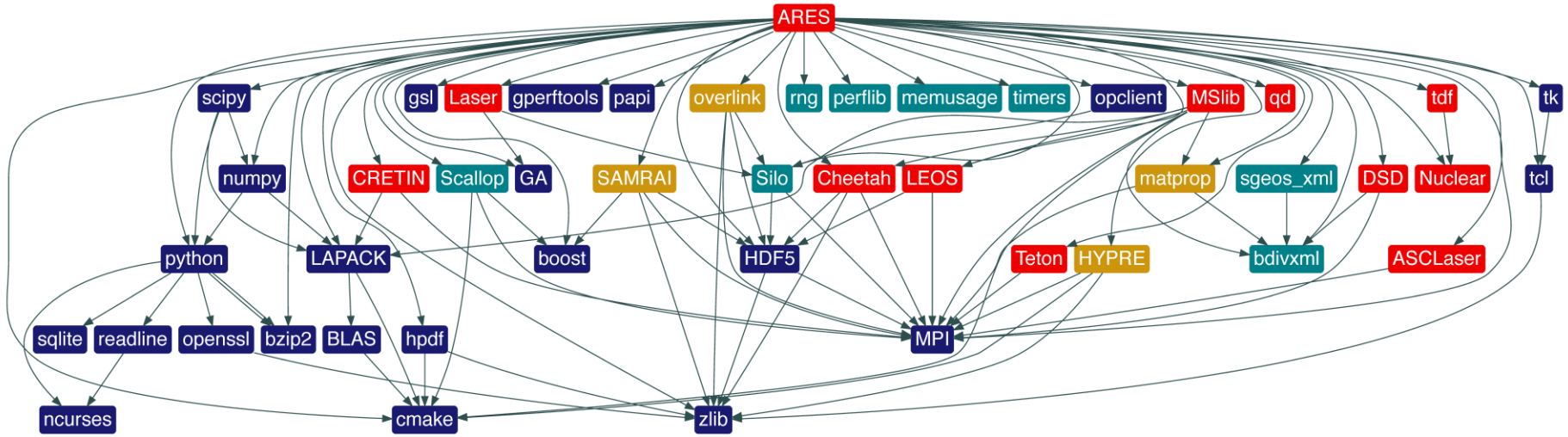
建设中

问题背景 – 软件堆栈维护与更新的挑战

同时，在 **HPC → HPC + AI + Big Data** 的背景下，

维护超算中心的软件堆栈环境，面临几个维度的新压力：

1. 硬件上复杂异构（计算上x86/arm/gpu, 通信上OPA/MLNX）
2. 软件背景更加复杂（MPI/OpenMP，Python，可视化软件）
3. 基础软件版本跨度大（GCC 4.8 → 9.2 / CUDA 6 → 11）



问题背景 – spack只解决了一部分的问题

spack取代了传统手工编译，提供了：

1. 编译依赖管理
2. 自动编译和部署
3. 版本管理

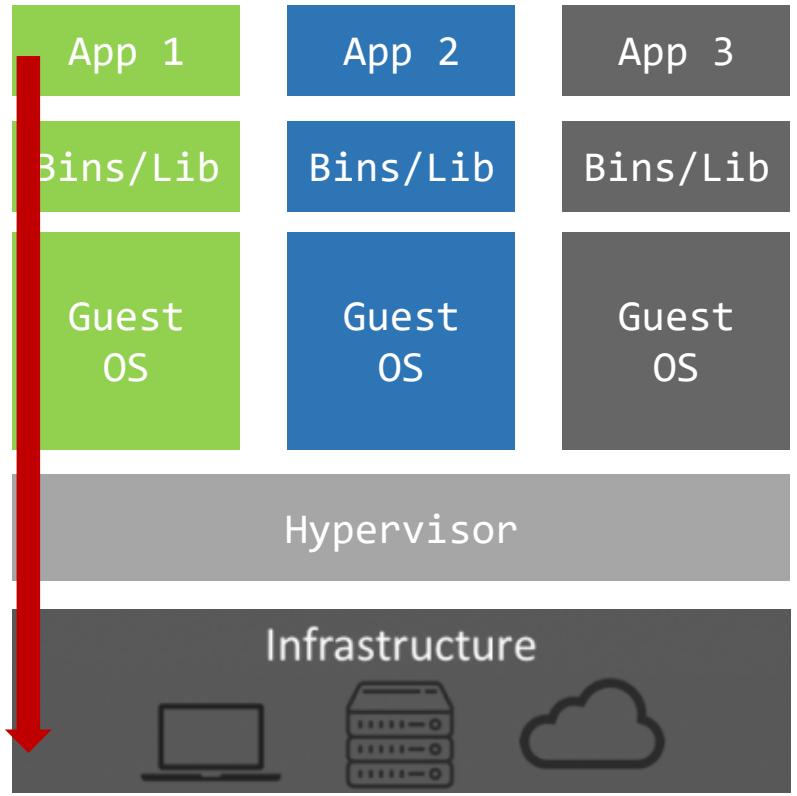
但spack还是会遇到以下问题：

1. 以管理员为中心，依赖管理员的维护
2. 容易受到spack上游社区的影响
3. 无法满足长尾用户（应用）的需求

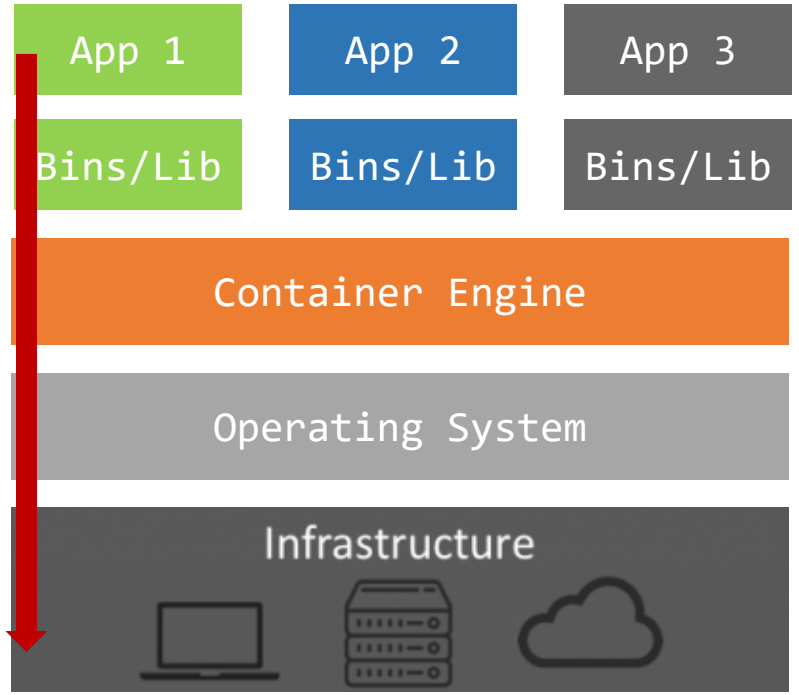
我们需要一个新技术来完整全面的解决这个问题

新技术：与虚拟机相比，容器更有优势

- **容器**是一种利用操作系统的资源隔离特性而实现的**轻量级虚拟化技术**。
- 容器应用共享宿主机的**内核**以及其他关键库(如硬件**驱动程序**)。
- 不同容器方案会在镜像格式、隔离的名字空间、启动方式上有区别，较为常用的包括：Docker、Singularity、Shifter等。



虚拟机



容器

现有的三种主流容器技术的对比

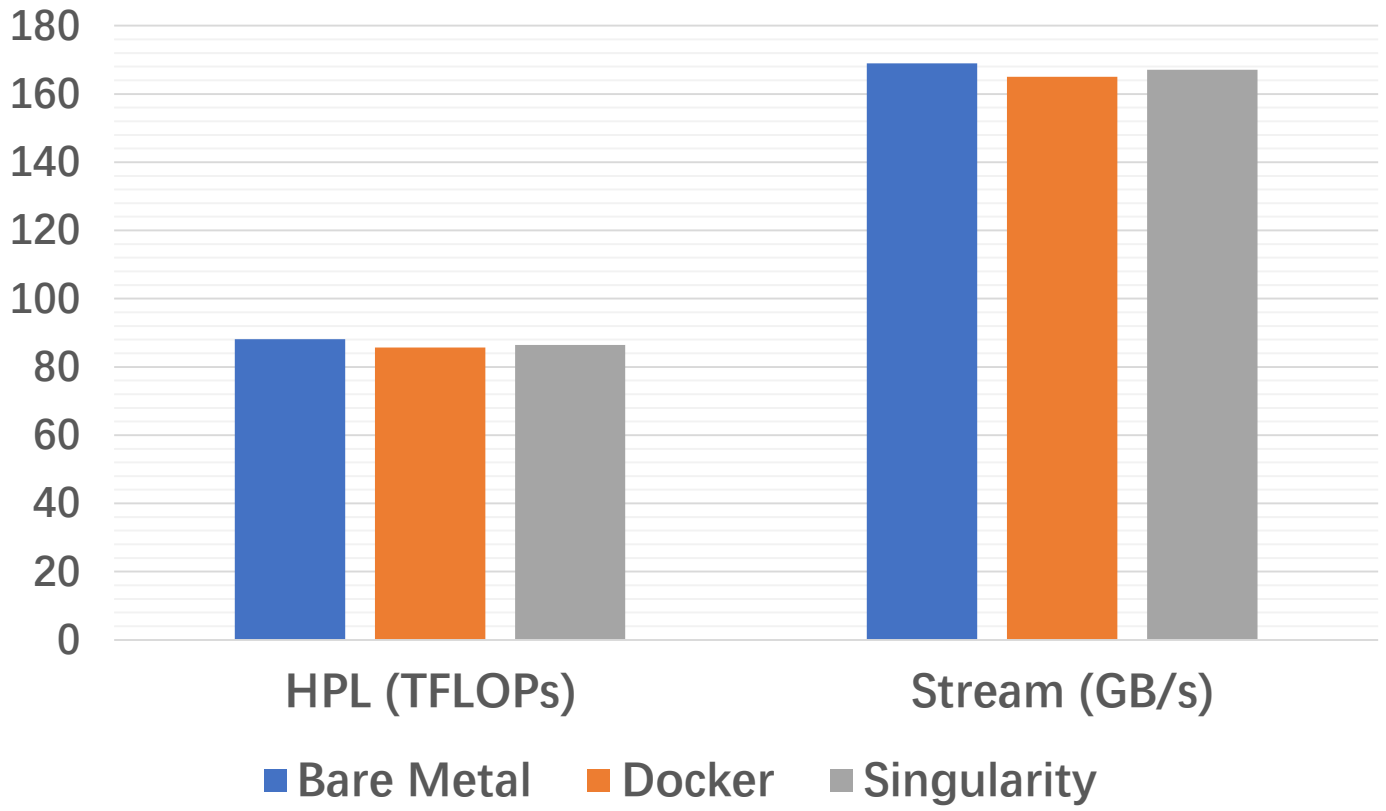
容器实现	无特权进程	隔离名字空间	MPI	GPU	InfiniBand	可额外配置网络	安装难度
Docker	否	PID, NET, IPC, MNT, UTC	原生支持	不原生支持	原生支持	是	中等
Singularity	是	PID, MNT, USR	原生支持	原生支持	原生支持	否	简单
Shifter	是	MNT	原生支持	不原生支持	原生支持	否	困难

我们选用Singularity的四个原因

- 真实应用性能开销低
- 深度学习训练性能更高
- 原生支持MPI和Slurm
- 安全性更高

原因一：性能开销低

	Bare Metal	Docker	Singularity
HPL (TFlops)	**88.12**	85.64	86.44
Stream (GB/s)	**169.00**	164.99	167.10



使用容器技术的开销均小于3%，singularity性能略好。

原因二：深度学习训练性能高

使用PyTorch在ImageNet数据集上训练ResNet50，**NVIDIA GPU Cloud**的容器镜像**明显优于**开源版本，但**不开源**，无法从Dockerfile重新构建镜像。

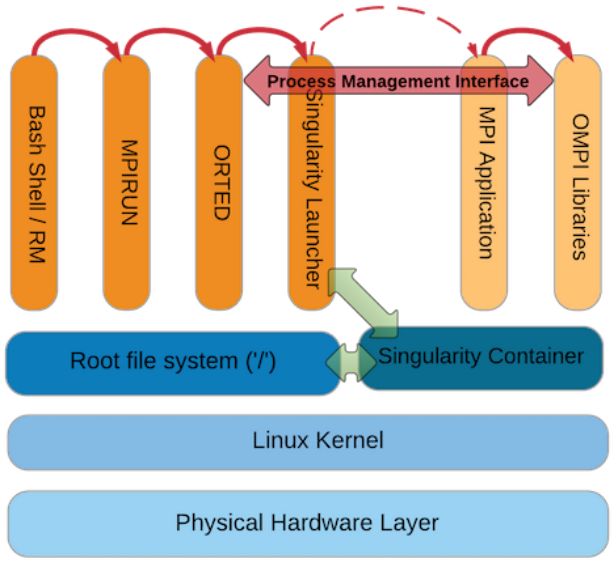
Images/s	Bare-metal (Non-NGC)	Docker (Non-NGC)	Docker (NGC)	Singularity (NGC)
1节点(FP16)	12363	12086	13639	**13891**
2节点(FP16)	24310	23623	25608	**25784**

原因三：原生支持MPI和Slurm

- Singularity：支持MPI和slurm作业提交

```
mpirun –np 2 singularity exec test.sif hostname
srun -N 2 -p dgx2 --exclusive singularity exec test.sif
hostname
```

Singularity与**SLURM**结合紧密，可直接使用srun启动多节点MPI应用。



原因四：安全性更高

- Docker的安全隐患
  - Docker需要由root用户启动守护进程
  - 普通用户可以通过`--privileged`选项获得特权并执行未经授权的操作

```
docker run --privileged -it –rm
```

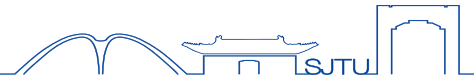
- Singularity对比Docker有更好的安全性
  - 无需守护进程
  - 用户权限在容器内外保持一致

目录

我们在实践中遇到的三个问题

- 如何利用容器技术搭建高性能计算集群的软件堆栈？
- 如何让用户在高性能计算集群上方便的构建和使用容器？
- 如何扩展容器的服务范围，让不了解容器技术的用户享受到容器带来的好处？

总体解决方案：交大超算容器服务体系



Environment Module

HPC Studio

NVIDIA GPU Cloud

(PyTorch, TensorFlow)

Application Container
(Lammps,Gromacs,OpenFoam..)

User-build Container

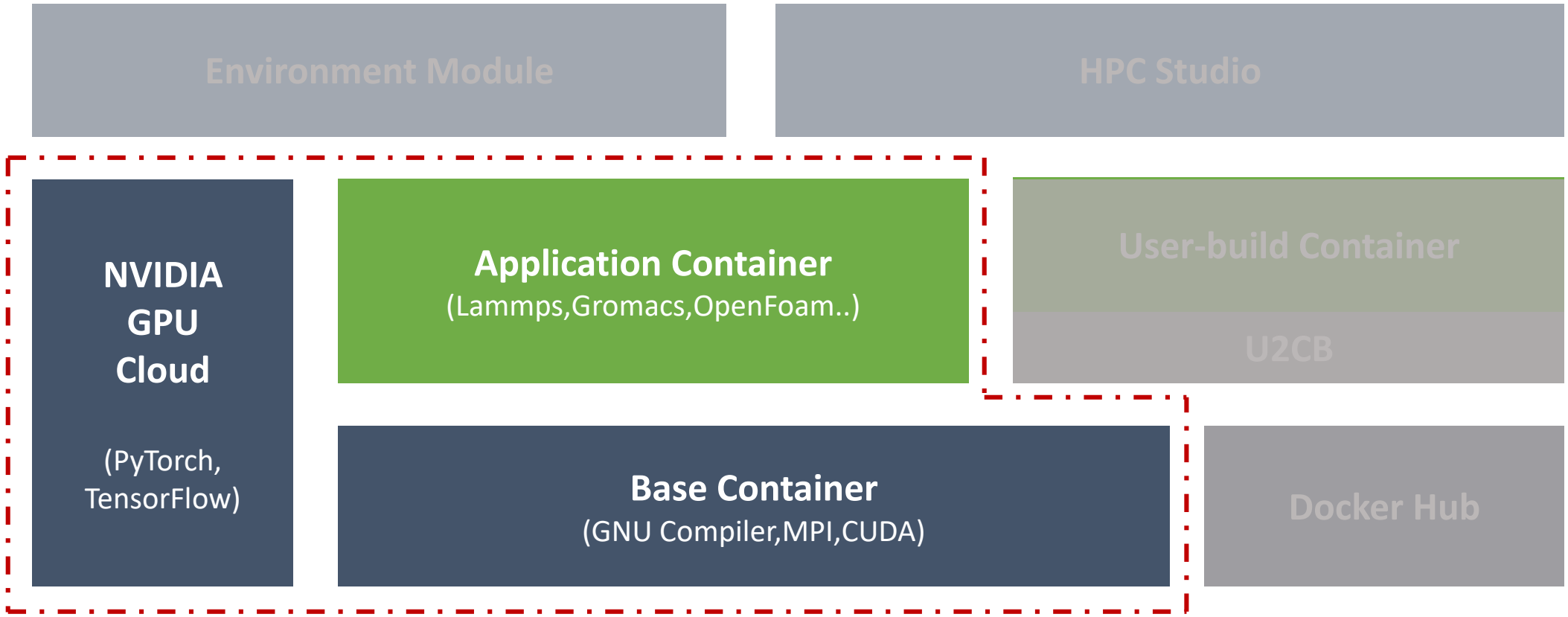
U2CB

Base Container
(GNU Compiler,MPI,CUDA)

Docker Hub

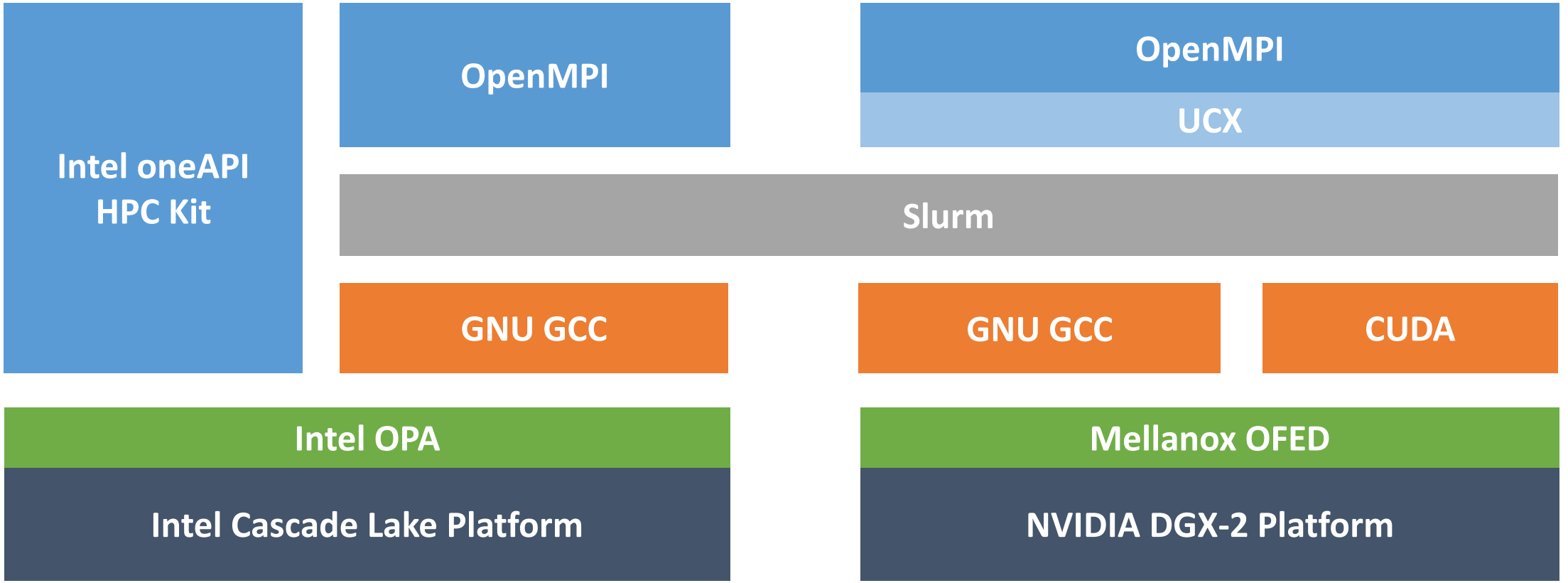
针对问题一，我们使用基础应用镜像

问题一：如何利用容器技术搭建高性能计算集群的软件堆栈?



构建基础镜像的具体方法

基础镜像中包括基本的HPC组件，例如Compiler / MPI / CUDA / Mellanox OFED，以及在HPC软件堆栈中常用的其他组件。



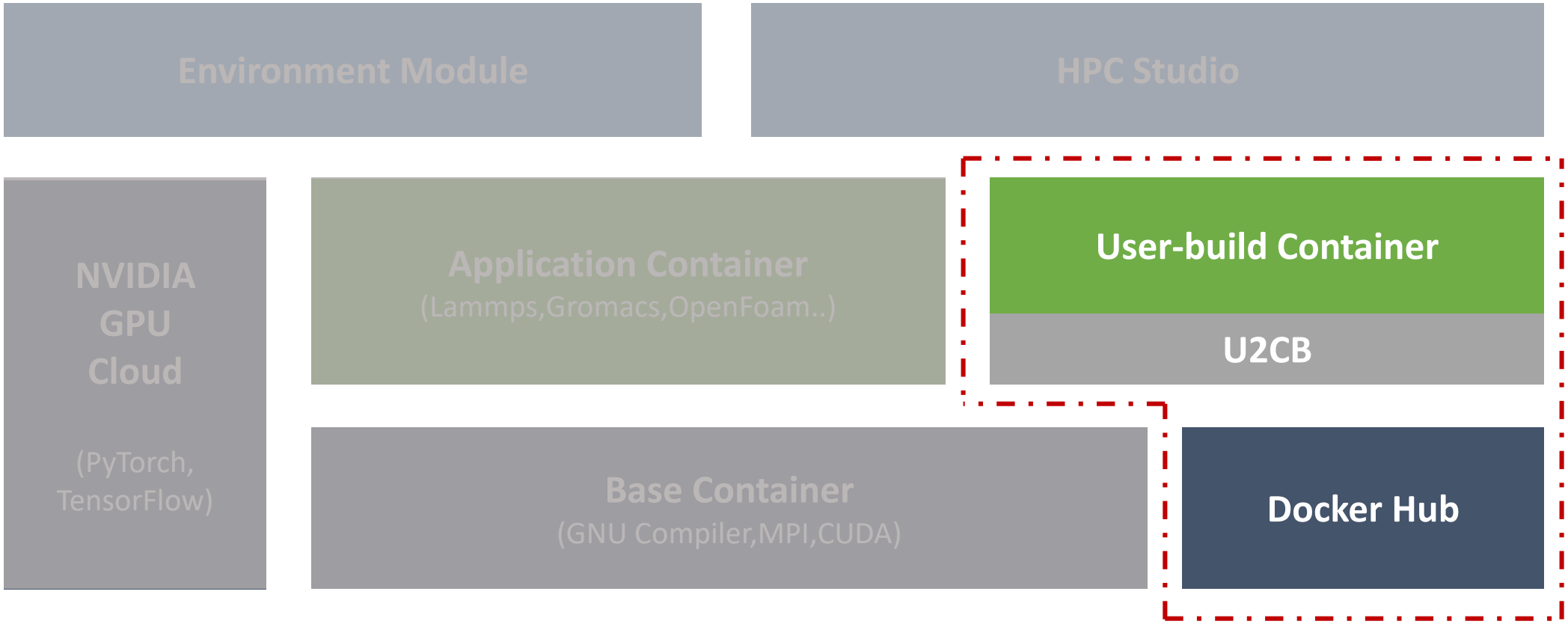
构建应用镜像的具体方法

- 针对架构的编译优化（GPU / CPU）
- 多阶段构建（优化容器尺寸）

```
#============================#
# multi-stage: build
#============================#
FROM sjtu/hpc-base:cuda-10.2.ompi-4.0 AS build

# compile
RUN . . .
   cmake -DCUDA=ON -DCudaTexture=ON -DCUDA_ARCH=70 ..

#============================#
# multi-stage: install
#============================#
FROM sjtu/hpc-base:cuda-10.2.ompi-4.0

COPY --from=build /opt/relion /opt/relion
ENV PATH=/opt/relion/bin:$PATH
```

针对问题二，我们提供非特权用户镜像构建的方法

问题二：如何让用户在高性能计算集群上方便的构建和使用容器?



Singularity现有的局限

- Singularity出于在HPC场景下的安全性保证

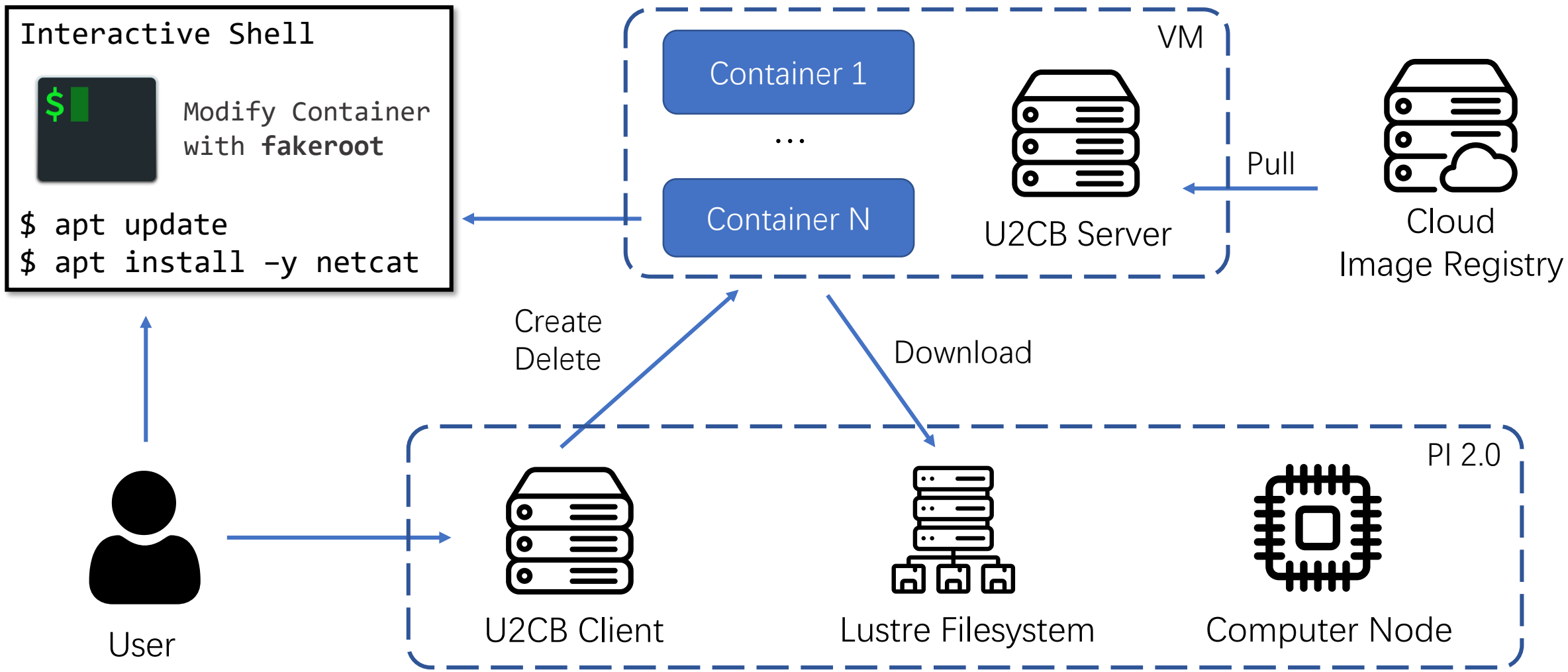
  设计时保证了用户在容器内外保持一致的低权限。

- Singularity提供的现有解决方案是

  用户在自有的服务器上使用root构建镜像并上传集群。

需要一套便捷、透明、安全的非特权用户容器构建方案

如何支持非特权用户镜像构建



非特权用户容器构建的关键技术 - Fakeroot Feature

- The fakeroot feature (commonly referred as rootless mode) allows an unprivileged user to run a container as a ***fake root***
- *user namespace UID/GID mapping* from Linux kernel

UID inside container	UID outside container
0 (root)	1000 (user)
1 (daemon)	131072 (non-existent)
2 (bin)	131073 (non-existent)
...	...
65536	196607

Requirements:

Singularity >= 3.5
Linux kernel >= 3.8

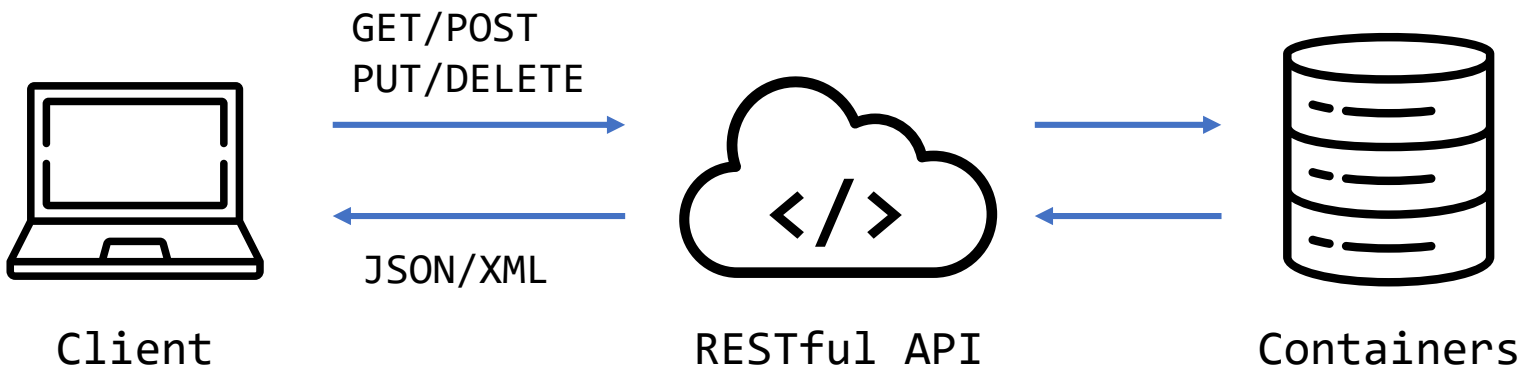
非特权用户容器构建的三个优势

一、基于RESTful API设计

- 轻量。直接基于http，不需要任何其他的消息协议。
- 无状态。在调用一个接口（访问、操作资源）的时候，不用考虑上下文，极大的降低了系统的复杂程度。

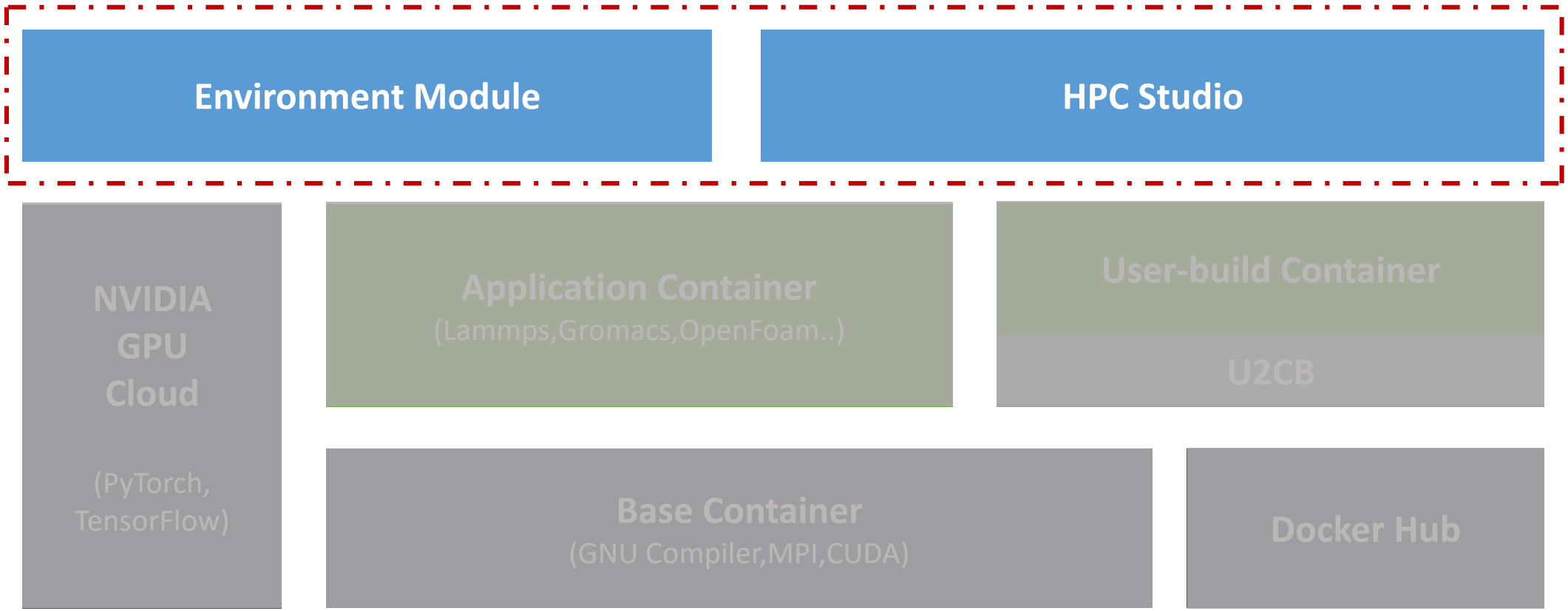
二、支持交互式和define文件进行容器构建

三、使用fakeroot特性保障了U2CB Server的安全



针对问题三，我们提供为用户提供便捷的接口

问题三：如何扩展容器的服务范围，让不了解容器的用户享受到容器带来的好处?



解决方案的第一部分：使用Module封装

Module是超算中心最常用的软件管理方式，也是用户最熟悉的应用使用方式

```
$ module load gromacs
$ declare -f gmx_mpi
gmx_mpi () {
        singularity run --nv /path/gromacs/2020-ngc.sif gmx_mpi $@
}
```

在使用Module封装后，

1.用户可以使用熟悉的module命令，即可享受经过优化的高性能应用镜像。

2.对用户来说，无需更改任何脚本或流程，学习成本为0。

解决方案的第二部分：与HPC Studio整合

GUI Application (Qt, OpenGL)

GNU Octave

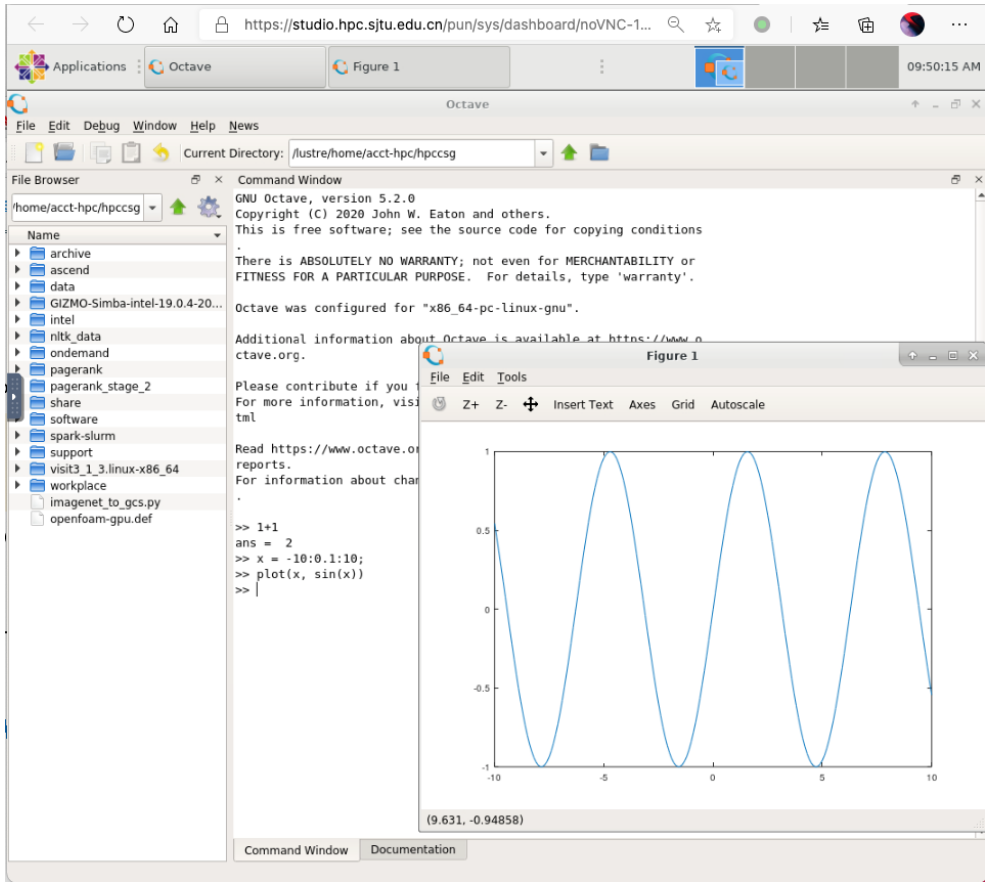
OVITO

ParaView

VISIT

GUI Application

网址：https://studio.hpc.sjtu.edu.cn/
基于OOD开发的可视化计算平台



解决方案的第二部分： 与HPC Studio整合

Web Application

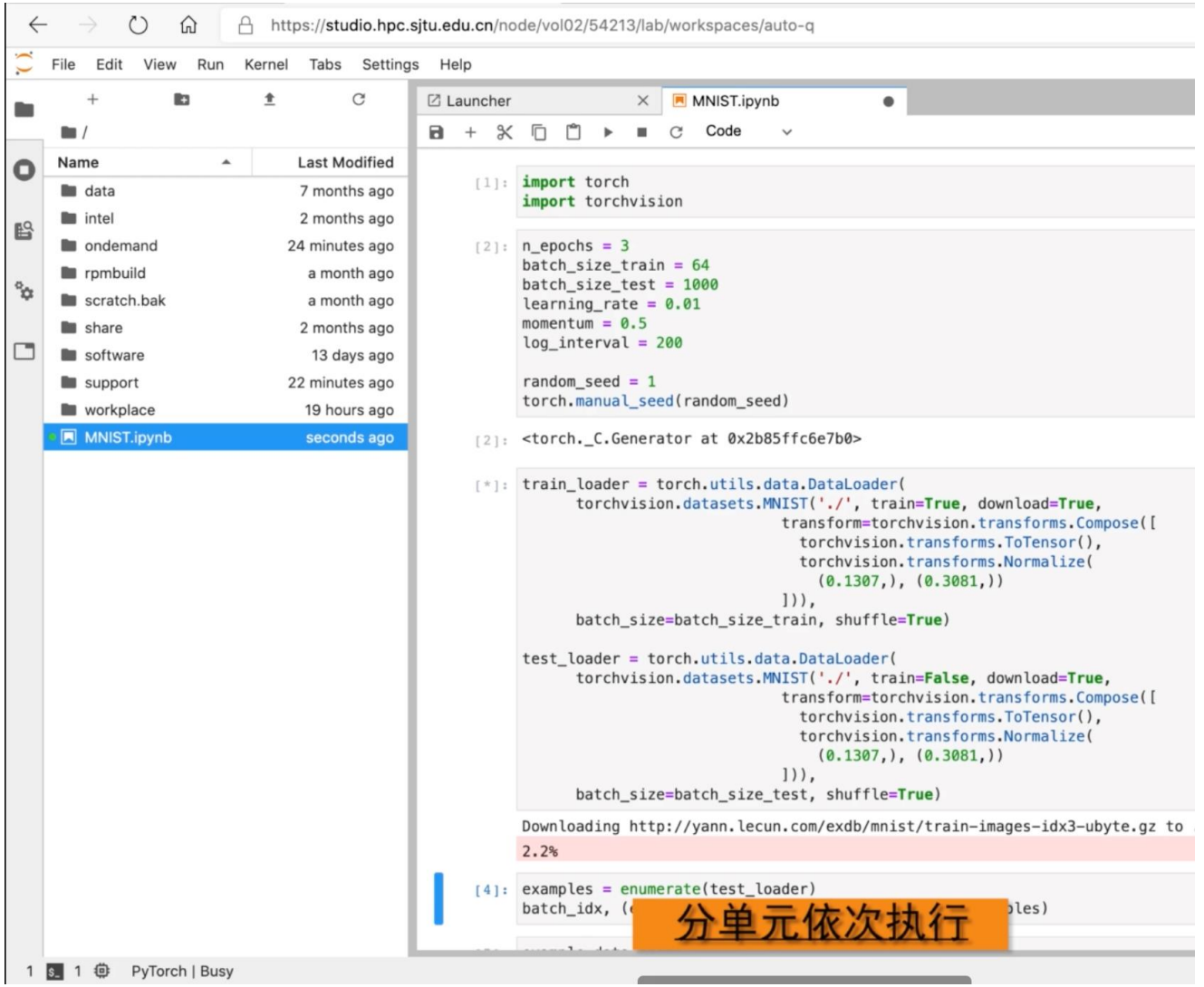
jupyter

RStudio®

VS Code

TensorBoard

Web Application



目录

案例一：封装及使用分子动力学软件的应用镜像

- 作业提交脚本**无需任何修改**
- 提供平台最优化的性能

```
#!/bin/bash
#SBATCH --job-name=gmx_test
#SBATCH --partition=cpu
#SBATCH –n 80
#SBATCH --ntasks-per-node=40

module load gromcas/2020-cpu

srun --mpi=pmi2 gmx_mpi mdrun -resethway -noconfout -nsteps 4000 –v -ntomp 6
```

Gromacs

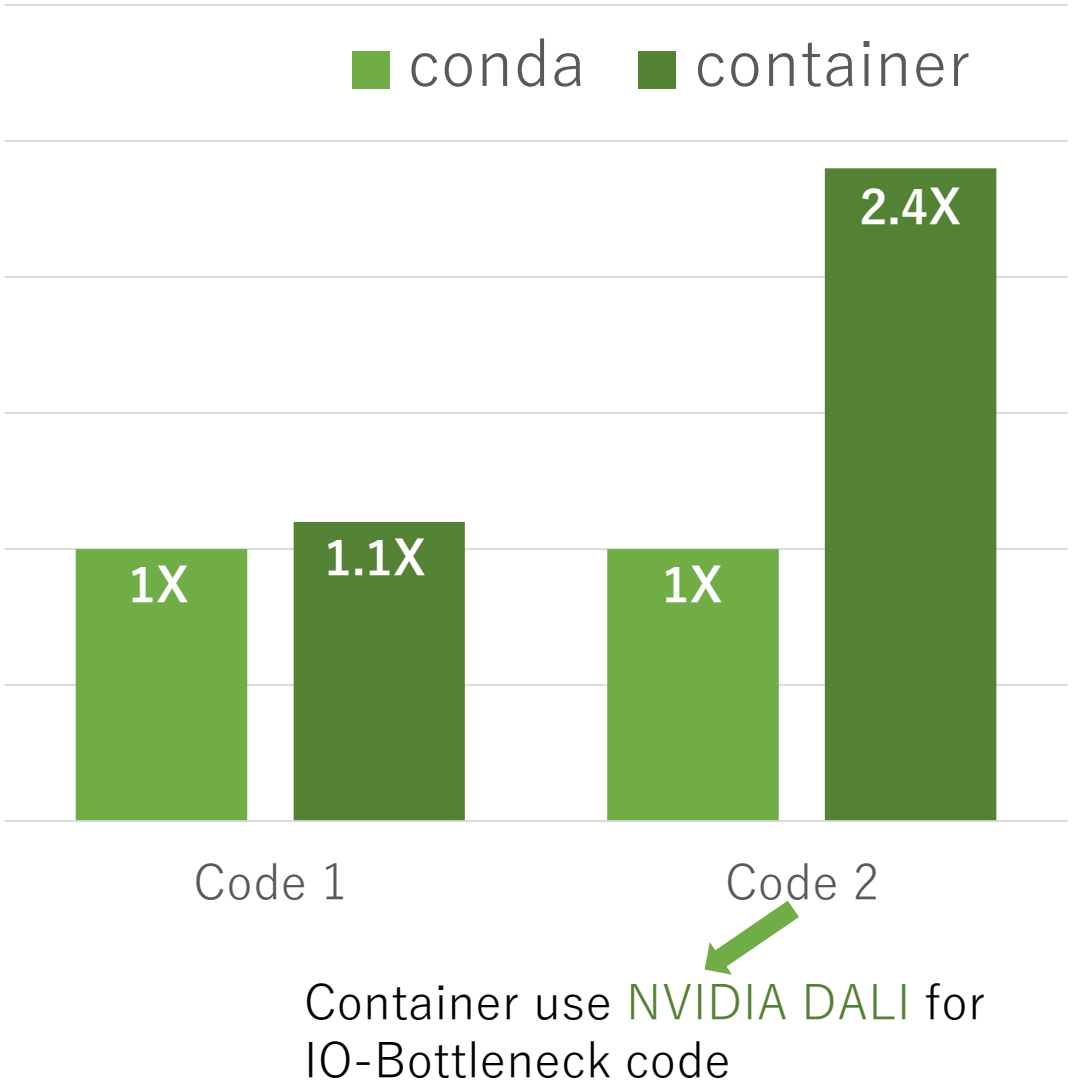
```
#!/bin/bash
#SBATCH --job-name=lmp_test
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:2

module load lammps/2020-dgx-kokkos

srun --mpi=pmi2 lmp -k on g 2 t 12 -sf kk -pk kokkos comm device -in in.eam
```

Lammps

案例二：封装及使用PyTorch的应用镜像

- 作业提交脚本**无需任何修改**
- 针对conda安装版本有明显性能优势

```
#!/bin/bash
#SBATCH --job-name=pytorch_test
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:2

module load pytorch/1.6.0

python -c 'import torch; \            print(torch.__version__); \
                print(torch.zeros(10,10).cuda().shape)'
```

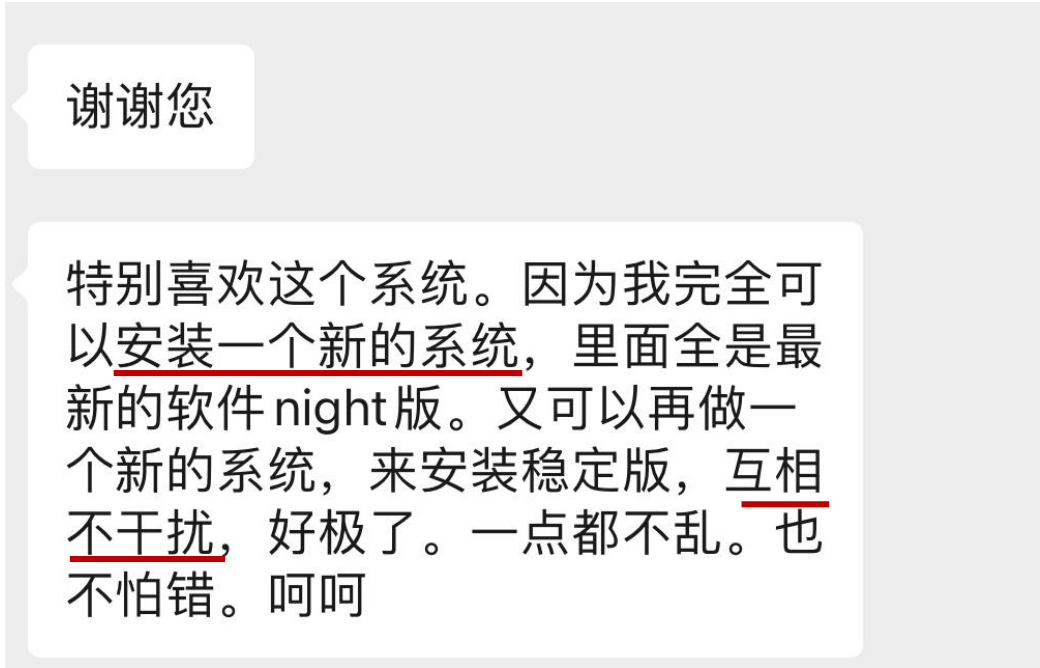


案例三：支持复杂的生信软件

生命科学学院 汪志军团队

软件名称：Collaborative Computational Project for Electron cryo-Microscopy

软件简介：The CCP-EM software suite is a package containing tools for cryo-EM, including: Relion3, Coot, LocScale 等**15个应用**

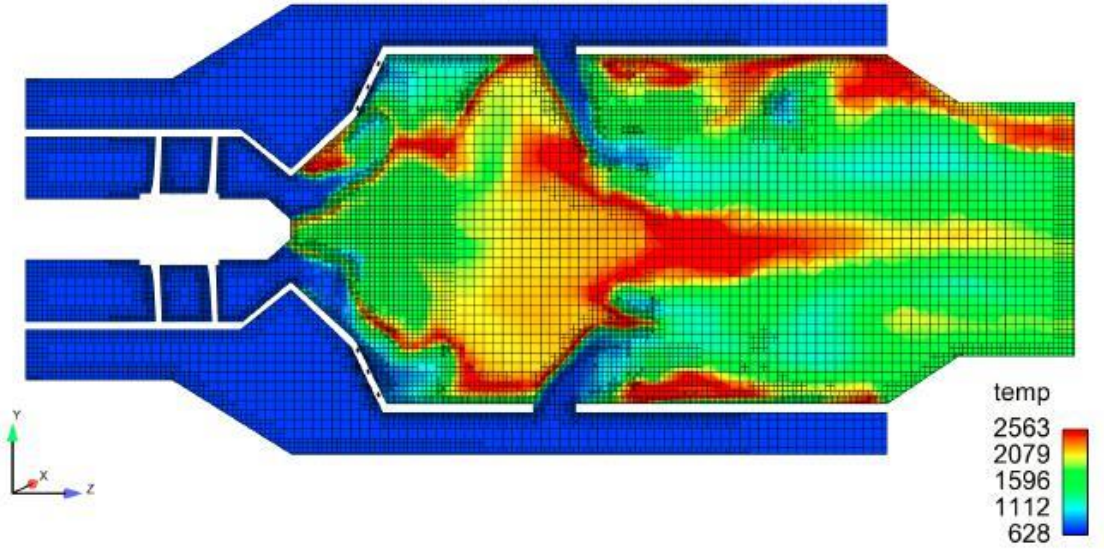


案例四：燃烧仿真计算平台

密西根学院 陈谦斌团队

项目名称：固体燃料燃烧仿真

项目简介：针对燃料燃烧的过程，通过理论建模，数值模拟和试验验证，建立一种基于多尺度的高精度仿真工具，实现对燃料燃速和燃烧产物特性的准确预示，为燃料配方调试，过载两相流的模拟，以及燃烧稳定性的评估奠定坚实基础。



2天将实验室基于Ubuntu的开发环境直接迁移到PI 2.0

基础镜像：保证了在PI 2.0上的性能

U2CB：自定义安装的依赖或软件

交大使用容器技术支持的主流软件

GROMACS
FAST. FLEXIBLE. FREE.

QUANTUMESPRESSO

TensorFlow

RStudio

RELION

GNU Octave

PyTorch

OVITO

LAMMPS

Amber

MindSpore

VISIT

OpenFOAM

VASP

RAPIDS

igv

ParaView

Scilab

VS Code

目录

总结

- 我们的容器体系很好的解决了在复杂的硬件异构上搭建高性能软件堆栈的挑战
- 使用非特权用户容器构建方案，用户可以方便的在集群上构建和使用容器
- 提供用户交互接口，让不了解容器的用户使用容器，极大扩展了容器的服务范围
- 并且在实践中证明了：
  - 用户：更灵活高效的应用安装和使用，提高科研效率
  - 系统管理员：将繁综复杂的应用环境和系统环境解耦，提高系统稳定性和运维效率

GitHub: https://github.com/SJTU-HPC/hpc-app-container

https://github.com/SJTU-HPC/container-modules