

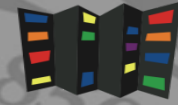
Introduction to Containers

Martin Čuma

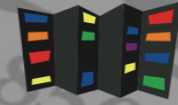
Center for High Performance Computing

University of Utah

m.cuma@utah.edu



- Why do we want to use containers?
- Containers basics
- Run a pre-made container
- Build and deploy a container
- Containers for complex software



1. Download the talk slides

<http://home.chpc.utah.edu/~mcuma/chpc/Containers20s.pdf>

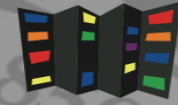
<https://tinyurl.com/yd2xtv5d>

2. Using FastX or Putty, ssh to any CHPC Linux machine, e.g.

```
$ ssh uxxxxxx@frisco.chpc.utah.edu
```

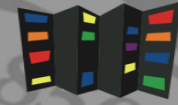
3. Load the Singularity and modules

```
$ module load singularity
```

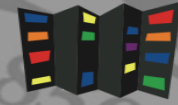


1. Create a GitHub account if you don't have one
<https://github.com/join>
{Remember your username and password!}
2. Go to <https://cloud.sylabs.io/home>
click Remote Builder, then click Sign in to Sylabs and then Sign in with GitHub,
using your GitHub account
3. Go to <https://cloud.sylabs.io/builder>
click on your user name (upper right corner), select Access Tokens, write token
name, click Create a New Access Token, and copy it
4. In the terminal on frisco, run the following command and paste the token when
prompted:

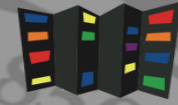
```
singularity remote login
```



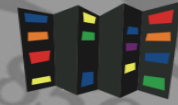
Why to use containers?



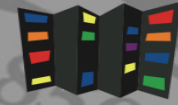
- Some programs require complex software environments
 - OS type and versions
 - Drivers
 - Compiler type and versions
 - Software dependencies
 - glibc, stdlibc++ versions
 - Other libraries and executables
 - Python/R/MATLAB versions
 - Python/R libraries



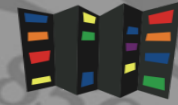
- Research outputs include software and data
- Software reproducibility
 - Software repositories (svn, git)
 - Good but often software has dependencies
- Data reproducibility
 - Data as publication supplementary info, centralized repositories (NCBI), ...
 - Disconnected from the production environment
- Package data AND code AND compute environment in one file



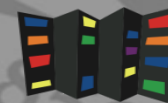
- Develop a program / pipeline locally, run globally
- Scale to parallel resources
 - Run many times
 - Use local or national HPC resources
- Automate the process
 - Container/software building and deployment
 - Parallel pipeline



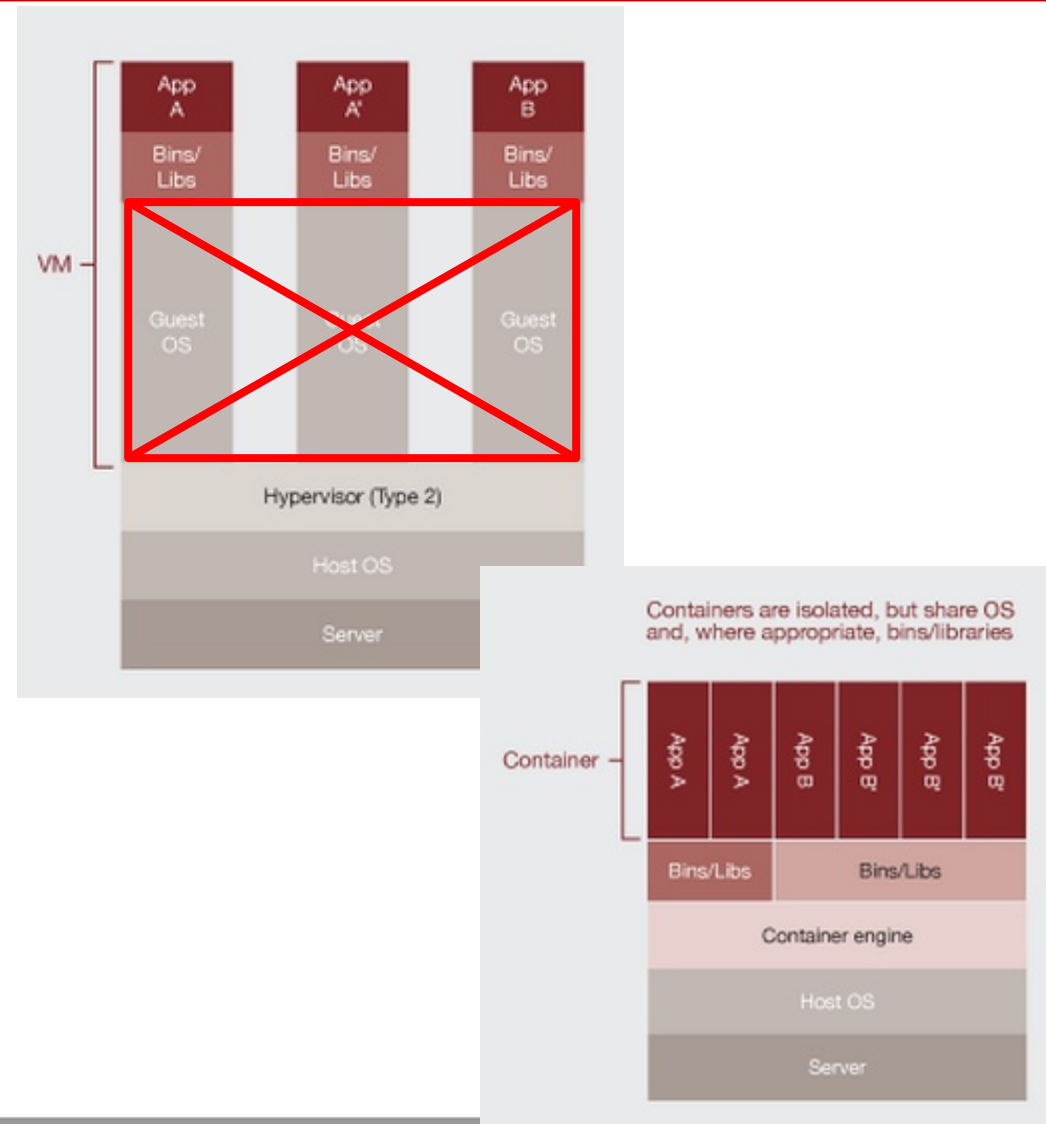
- Old applications built on old Linux versions can run on newer Linux host, and vice versa
- May be able to run Windows programs on Linux

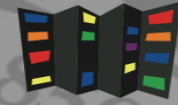


Container basics



- Hardware virtualization
 - Running multiple OSes on the same hardware
 - VMWare, VirtualBox
- OS level virtualization
 - run isolated OS instance (guest) under a server OS (host)
 - Also called containers; user defined software stack (UDSS)
 - Docker, Singularity

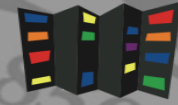




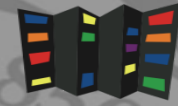
- Isolate computing environments
 - And allow for regenerating computing environments
- Guest OS running over host OS
 - Guest's OS can be different than host's
 - Low level operations (kernel, network, I/O) run through the host
- From user standpoint guest OS behaves like standard OS

- Docker
 - Well established
 - Has docker hub for container sharing
 - Problematic with HPC
- Singularity
 - Designed for HPC, user friendly
 - Support for MPI, GPUs
- Charliecloud; Shifter, udocker
 - Also HPC designed, built on top of Docker
 - Simple but less user friendly



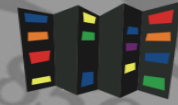


- OS vendor tools
 - RedHat Podman, Buildah, Skopeo - new with RHEL 8
- Other Linux based container solutions
 - runC, LXC
- Orchestration tools
 - use containers to spin up servers
 - Kubernetes, Docker Compose

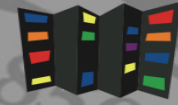


- Integrate with traditional HPC
 - Same user inside and outside of the container
 - Same file systems (home, scratch), environment
 - Can integrate with existing software (CHPC sys branch)
- Portable and sharable
 - A container is a file
 - It can be built on one OS and run on another
- Only Linux support right now
- Possible security issues due to the use of setUID executables
 - Hacker can exploit potential flaws in setUID programs to gain root
 - <https://sylabs.io/guides/3.8/user-guide/security.html>

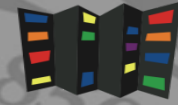




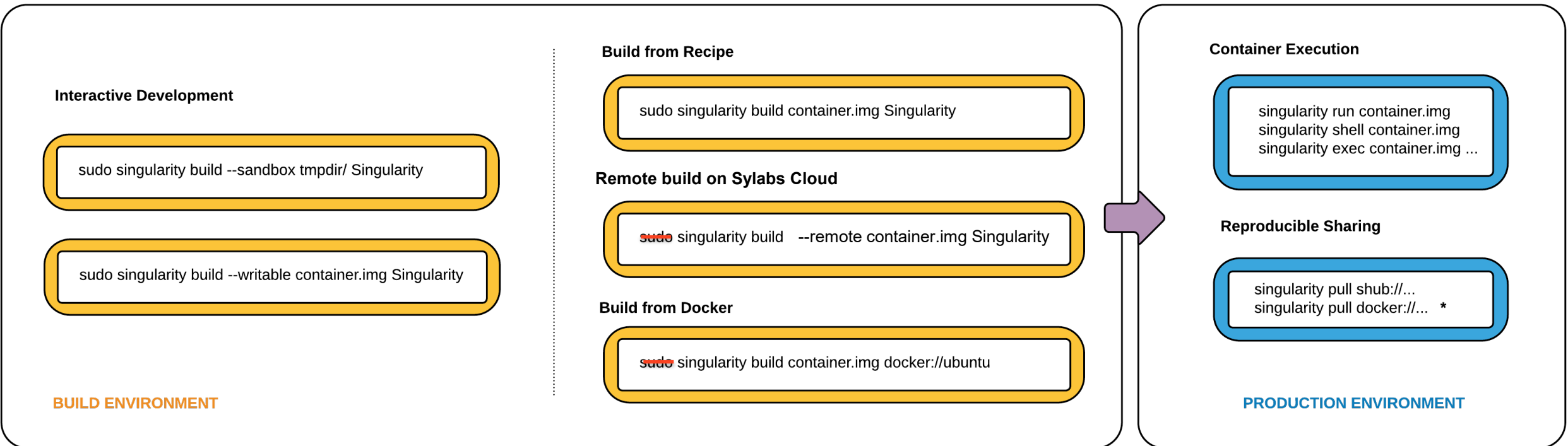
- Containers need privilege escalation to run
 - Give sudo
 - Run root owned daemon process (Docker)
 - Use setUID programs (programs which parts can run in privileged mode) (Singularity now, udocker)
 - User namespaces – new Linux kernel feature to further isolate users (Charliecloud, limited Singularity, Docker)
 - Linux capability set – fine grained privilege isolation (Singularity future)
- In HPC environment
 - setUID if you have some trust in your users, user namespaces if you don't (and have newer Linux distribution – e.g. CentOS \geq 7.4)



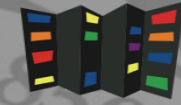
- Uses user namespaces for isolation
 - More secure
 - Limited to CentOS 7 and other recent Linux distributions (not supported in older CentOS or other Linux releases)
- Uses Docker containers
 - Needs Docker to build containers
 - Extracts and repackages Docker containers
- Singularity has an `–usersns` option to force User namespace
 - But capabilities limited to directory (sandbox) based containers



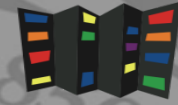
- A lightweight runtime for Docker containers
- No actual root escalation
 - Only does chroot to mount base OS to a different path
 - Uses different mechanisms to achieve this
 - The default, PRoot, uses ptrace system call
- Another option to run DockerHub containers
- Allows to change container files without privileges
- We've seen problems with stale file handles in the past



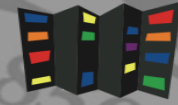
* Docker construction from layers not guaranteed to replicate between pulls



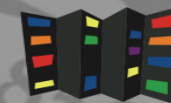
Run a pre-made container



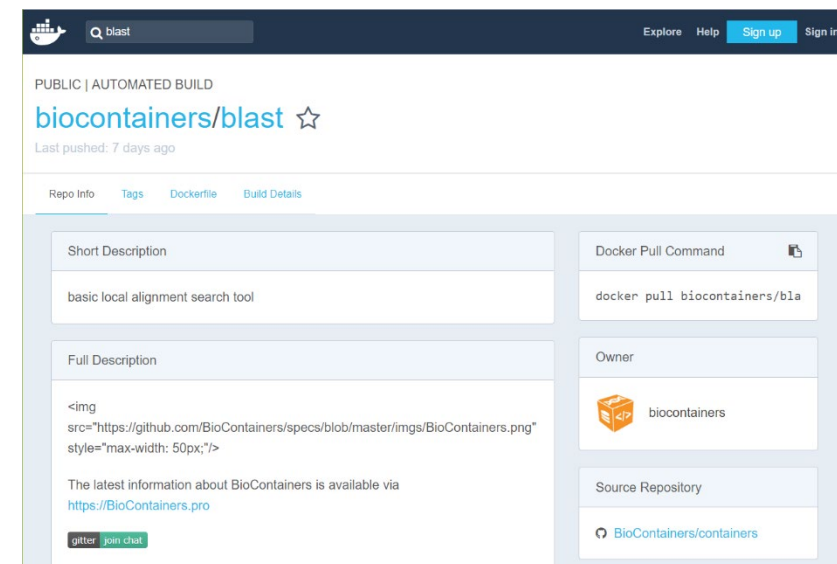
- Building a container requires a root, or sudo
 - You can do that on your own machine
 - You can do that in the cloud (e.g. Sylabs Cloud, Docker Hub).
 - You can't do that at CHPC clusters
 - > build your containers locally, or on container hubs
- You can run a container as an user
 - You can run your own containers at CHPC
 - You can run CHPC provided containers at CHPC
 - You can run containers obtained on the internet at CHPC



- Containers are run in user space (no root required)
- An appropriate environment module has to be loaded
 - Singularity, udocker, Charliecloud
- User inside of the container
 - Current user
 - If specified, other user, including root
 - Root inside container is contained = can't be root outside
- Containers can't be modified by non-root user
 - One small exception with udocker



- Most containers reside in registries
 - Content delivery and storage systems for container images
- Docker Hub is the most common registry
 - <https://hub.docker.com>
 - Contains many channels, e.g. Biocontainers (<http://biocontainers.pro/>)
- There are a few other registries
- Sylabs (Singularity) also has a hub (library)
 - We'll use it to build a container



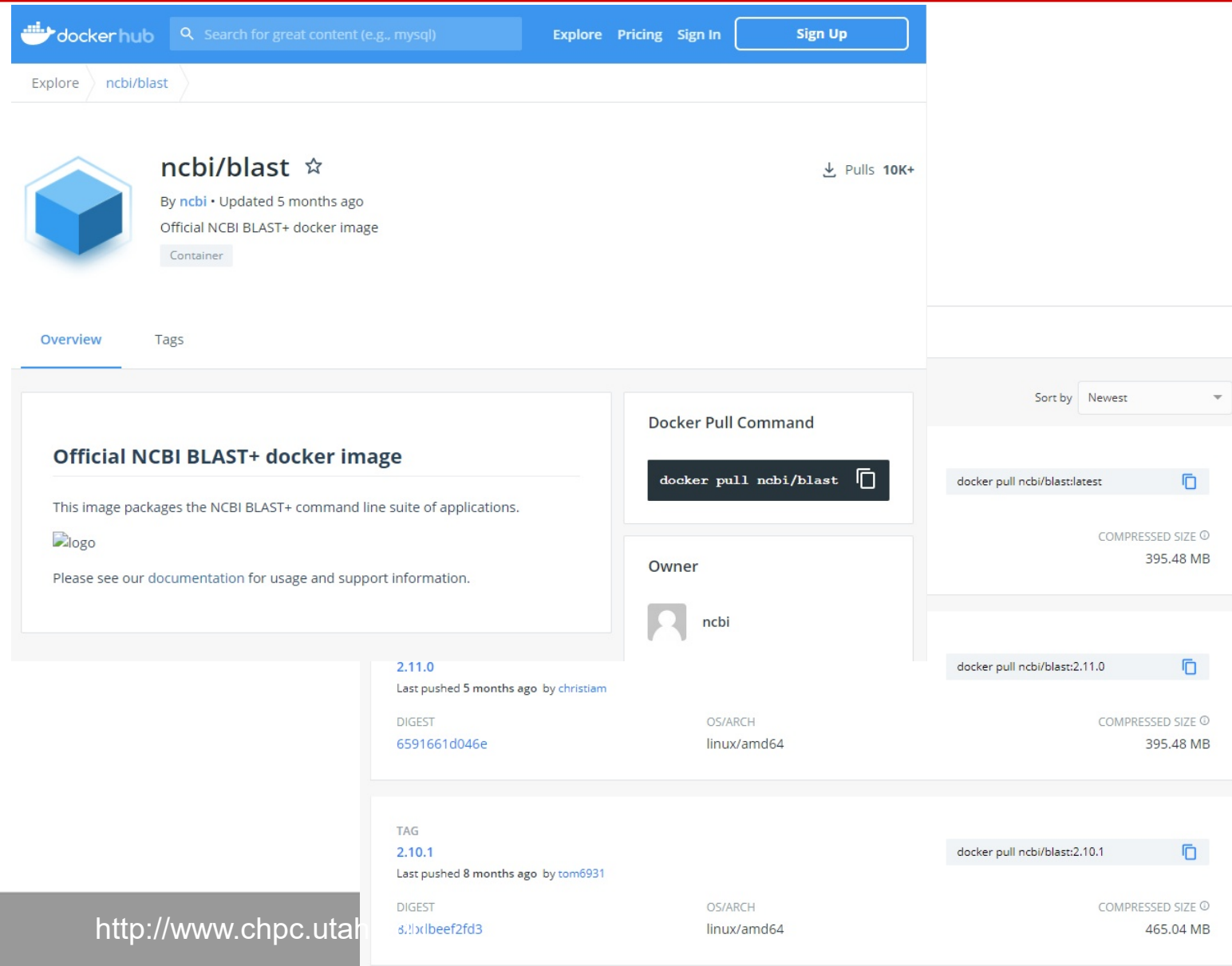
- Google or hub.docker.com search

- E.g. “blast docker”

- Use singularity (Sylabs library) search

```
$ singularity search lammers
Found 4 containers for 'lammers'
      library://lammers/default/lammers_development
```

```
      Tags: centos7
centos8 fedora30_mingw
fedora32_mingw ubuntu16.04
ubuntu18.04 ubuntu18.04_amd_rocm
ubuntu18.04_amd_rocm_cuda
ubuntu18.04_intel_openc1
ubuntu18.04_nvidia ubuntu20.04
```



The screenshot shows the Docker Hub page for the `ncbi/blast` container image. The page includes a search bar, navigation links, and a list of tags. The main content area displays the image's description, pull command, owner information, and a list of tags with their respective digests and compressed sizes.

ncbi/blast ☆ ↓ Pulls 10K+

By `ncbi` • Updated 5 months ago
Official NCBI BLAST+ docker image
Container

Overview Tags

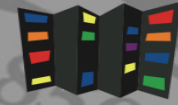
Official NCBI BLAST+ docker image

This image packages the NCBI BLAST+ command line suite of applications.

`docker pull ncbi/blast`

Owner
`ncbi`

TAG	DIGEST	OS/ARCH	COMPRESSED SIZE
2.11.0	6591661d046e	linux/amd64	395.48 MB
2.10.1	s.!!xlbeef2fd3	linux/amd64	465.04 MB



- pull will download the container and create sif file
- build does the same but names sif file and has more options
- run/exec will download if needed and run

```
$ singularity pull docker://ubuntu:latest
```

```
- Singularity pulls the Docker layers to ~/.singularity, but puts sif file to local directory
```

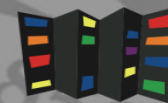
```
$ singularity build my_ubuntu.sif docker://ubuntu:latest
```

```
$ ls
```

```
my_ubuntu.sif    ubuntu_latest.sif
```

```
$ singularity exec ubuntu_latest.sif /bin/bash
```

```
$ singularity shell ubuntu_latest.sif
```



- Let's create a Python script

```
echo 'print("hello world from the outside")' > myscript.py
```

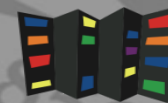
- Now run this script using the system's Python

```
python ./myscript.py
```

- Now run the script in the DockerHub python container

```
singularity exec docker://python python ./myscript.py
```

...Conclusion: Scripts and data can be kept inside or outside the container. In some instances (e.g., large datasets or scripts that will change frequently) it is easier to containerize the software and keep everything else outside.



- Let's create a Python script

```
echo 'print("hello world from the outside")' > myscript.py
```

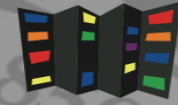
- Now run this script using the system's Python

```
python ./myscript.py
```

- Now run the script in the DockerHub python container

```
singularity exec docker://python python ./myscript.py
```

...Conclusion: Scripts and data can be kept inside or outside the container. In some instances (e.g., large datasets or scripts that will change frequently) it is easier to containerize the software and keep everything else outside.



- **Binding mount points (singularity < 3.0)**

```
$ export SINGULARITY_BINDPATH="/tmp"  
$ singularity shell -B /scratch/local/$USER:/tmp  
ubuntu_python.img
```

- **Specifying shell**

```
$ export SINGULARITY_SHELL=/bin/bash  
$ singularity shell -s /bin/bash ubuntu_python.img
```

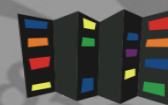
- **More specialized topics – ask us**

- Using environment modules from the host
- Using GPUs, MPI over InfiniBand

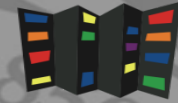


- Need to bring in the Nvidia driver stack
 - --nv runtime flag brings the drivers from the host
 - Still need to have a compatible CUDA installed in the container (older than or the same as the driver)
 - The present NVIDIA Driver at CHPC is 450.51, which is compatible with CUDA 11.0 or lower.
 - This means you can build a container by bootstrapping a base NVIDIA CUDA image (11.0 or lower) from: <https://hub.docker.com/r/nvidia/cuda/>
- On a GPU node, can e.g. execute:

```
singularity exec --nv docker://tensorflow/tensorflow:latest-gpu python -c "import tensorflow as tf; tf.config.list_physical_devices()"
```



- Need to bring the IB stack in the container
 - Some people bring the needed IB libraries from the host
 - For Ubuntu we prefer to install the Ubuntu stack
 - <https://github.com/CHPC-UofU/Singularity-ubuntu-mpi>
- MPI
 - Build inside the container with IB, or use CHPC's modules
 - Prefer MPICH and derivatives, OpenMPI is very picky with versions
 - If using OS stock MPI, then make sure to LD_PRELOAD or LD_LIBRARY_PATH ABI compatible libmpi.so with InfiniBand
 - <https://github.com/CHPC-UofU/Singularity-meep-mpi>



- Many Linux programs are binary compatible between distros
 - Most installed binaries are (Intel, PGI tools, DDT, ...)
- No need to install these in the container – use our NFS mounted software stack through Lmod
 - Need to have separate Lmod installation for Ubuntu due to some files having different location
- In the container
 - Install Lmod dependencies
 - Modify /etc/bash.bashrc to source our Lmod

<https://github.com/CHPC-UofU/Singularity-ubuntu-python/blob/master/Singularity>

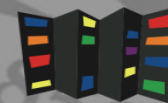


- General HPC
 - Nvidia Containers - <https://catalog.ngc.nvidia.com/containers>
 - E4S – Extreme-scale Scientific Software Stack – <https://e4s.io>
 - 50 programs, incl. GPU, multiple Linux Distros
 - Many other projects
- Interpreted languages
 - R - Rocker - <https://www.rocker-project.org/>
 - Python – official DockerHub - https://hub.docker.com/_/python

- Follow our documentation:
 - <https://www.chpc.utah.edu/documentation/software/singularity.php#exd>
- Pull container
 - NGC's LAMMPS: <https://catalog.ngc.nvidia.com/orgs/hpc/containers/lammps>

```
$ mkdir ~/containers; cd containers; ml singularity  
$ singularity pull lammps.sif docker://nvcr.io/hpc/lammps:29Sep2021
```
- Explore the container to find the program files

```
$ singularity shell lammps.sif  
$ which lmp
```

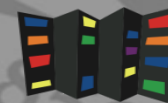


- Get a module file:

```
$ mkdir $HOME/MyModules/lammps
$ cd $HOME/MyModules/lammps
$ cp /uufs/chpc.utah.edu/sys/modulefiles/templates/container-template.lua
29Sep2021.lua
$ singularity build lammps.sif docker://nvcr.io/hpc/lammps:29Sep2021
```

- Modify the module file

```
-- path to the container sif file
local CONTAINER="/uufs/chpc.utah.edu/common/home/u0101881/containers/lammps.sif"
-- text array of commands to alias from the container
local COMMANDS =
{"usr/local/lammps/sm70/bin/lmp", "usr/local/lammps/sm70/bin/hpcbind"}
-- set to true if the container requires GPU(s)
local GPU = true
```

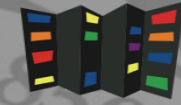


- Use module file:

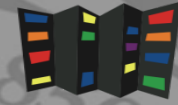
```
$ module use $HOME/MyModules/lammps
$ module load lammps/29Sep2021
$ which lmp
$ mkdir -p $HOME/lammps/data; cd $HOME/lammps/data
$ wget https://lammps.org/inputs/in.lj.txt
$ lmp -in in.lj.txt
```

(to run on GPU)

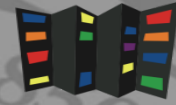
```
$ lmp -k on g 1 -sf kk -pk kokkos cuda/aware on neigh full comm device
binsize 2.8 -var x 8 -var y 8 -var z 8 -in in.lj.txt
```



Building Singularity containers



1. Create a GitHub account if you don't have one
<https://github.com/join>
{Remember your username and password!}
2. Go to <https://cloud.sylabs.io/home>
click Remote Builder, then click Sign in to Sylabs and then Sign in with GitHub,
using your GitHub account
3. Go to <https://cloud.sylabs.io/builder>
click on your user name (upper right corner), select Access Tokens, write token
name, click Create a New Access Token, and copy it
4. In the terminal on frisco, run the following command and paste the token when
prompted:
`singularity remote login`



- On **any system** with Singularity, even without administrative privilege, you can retrieve and use containers:
- Download a container from Docker Hub

```
singularity pull docker://some_image
```

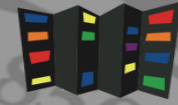
```
singularity build mycont.sif docker://some_image
```
- Run a container

```
singularity run mycont.sif
```
- Execute a specific program within a container

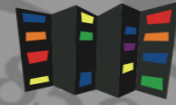
```
singularity exec mycont.sif python mysript.py
```
- “Shell” into a container to use or look around

```
singularity shell mycont.sif
```
- Inspect an image

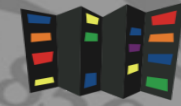
```
singularity inspect --runscript mycont.sif
```



- Complex software dependencies
 - Especially Python and R packages
 - bioBakery – intricate dependencies of Python and R which did not build on CentOS
 - SEQLinkage – instructions to build on Ubuntu using its packages
- Quick deployment
 - Some Linux distros provide program packages while others don't
 - paraview-python on Ubuntu via apt-get
- Deploying your own code or pipeline
- Modify or add onto an existing container



- Start with a the basic container (e.g. ubuntu:latest from Docker)
- Shell into the container
 - Install additional needed programs
 - If they have dependencies, install the dependencies – google for the OS provided packages first and install with apt-get/yum if possible
 - Put the commands in the `%post` scriptlet
- Build the container again
 - Now with the additional commands in the `%post`
 - If something fails, fix it, build container again
- Iterate until all needed programs are installed



- Build a container on a system on which you have administrative privilege (e.g., your laptop, singularity.chpc.utah.edu).
 - **Pros:** You can interactively develop the container.
 - **Cons:** Requires many GB of disk space, requires administrative privilege, must keep software up-to-date, container transfer speeds can be slow depending on personal network connection.
- Build a container on Sylabs Cloud
 - **Pros:** Essentially zero disk space required on your system, doesn't require administrative privilege, no software upgrades needed, easy to retrieve from anywhere, typically faster transfers from Sylabs Cloud to desired endpoint, interactive container development works.
 - **Cons:** Need to set up access to Sylabs Cloud, slower interactivity in the container development.

- Create a writeable container

```
$ singularity build --sandbox mycont.sif ubuntu16.def
```

- This creates a container directory called `mycont.sif`

- If additional installation is needed after the build

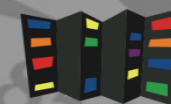
- Shell into the container and do the install manually

```
$ sudo singularity shell -w -s /bin/bash mycont.img
```

- Execute what's needed, modify container definition file, repeat

- When done, create a production container

```
$ sudo singularity build ubuntu16.sif ubuntu16.def
```



- Defines how the container is bootstrapped
 - Header – defines the core OS to bootstrap
 - Sections – scriptlets that perform additional tasks
- Header
 - Docker based (faster installation)

```
BootStrap: docker
```

```
From: ubuntu:latest
```

- Linux distro based

```
BootStrap: debootstrap
```

```
OSVersion: xenial
```

```
MirrorURL: http://us.archive.ubuntu.com/ubuntu/
```

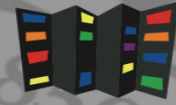
```
Bootstrap:docker
From:ubuntu:latest

%labels
MAINTAINER Andy M

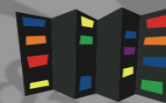
%environment
HELLO_BASE=/code
export HELLO_BASE

%runscript
echo "This is run when you run the image!"
exec /bin/bash /code/hello.sh "$@"

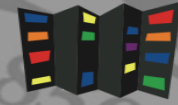
%post
echo "This section is performed after you bootstrap to build the image."
mkdir -p /code
apt-get install -y vim
echo "echo Hello World" >> /code/hello.sh
chmod u+x /code/hello.sh
```



- `%setup` Runs on the host
 - Install host based files (e.g. GPU drivers)
- `%post` Runs in the container
 - Install additional packages, configure, etc
- `%runscript` Defines what happens when container is run
 - Execution commands
- `%test` Runs tests after the container is built
 - Basic testing



- `%environment` Definition of environment variables
 - `%files` Files to copy into the container
 - `%labels` Container metadata
 - `%help` What displays during `singularity help` command
-
- More details at https://sylabs.io/guides/3.8/user-guide/definition_files.html



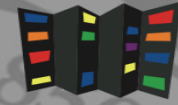
1. Have a Sylabs Cloud account and its token linked to your CHPC account
2. Log in to Frisco: `ssh uxxxxxx@frisco1.chpc.utah.edu`
3. Create a recipe file for your container, name it “Singularity”, e.g.

```
$ nano Singularity
Bootstrap: docker
From: alpine:3.9
%post
apk update; apk upgrade; apk add bash
```

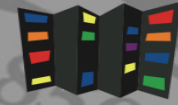
To exit and save type [ctrl-x], then “y”, then [enter].
4. Initialize Singularity and use `-remote` option to build

```
$ ml singularity
$ singularity build --remote alpine.sif Singularity
```
5. Container will be built in Sylabs Cloud and sif file brought back
6. Verify that the container is available by opening shell in it

```
$ singularity shell alpine.sif
```

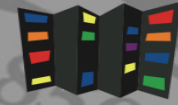


Troubleshooting and Caveats

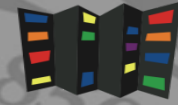


- Container problems are often linked with how the container “sees” the host system. Common issues:
 - The container doesn’t have a bind point to a directory you need to read from / write to
 - The container will “see” python libraries installed in your home directory (and the same is true for R and other packages). If this happens, set the PYTHONPATH environment variable in your job script so that it points to the container paths first.

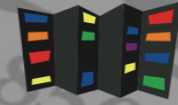
```
export PYTHONPATH=<path-to-container-libs>:$PYTHONPATH
```
 - or use the `--cleanenv` option
- To diagnose the issues noted above, as well as others, “shelling in” to the container is a great way to see what’s going on inside.
- Also, look in the `singularity.conf` file for system settings (can’t modify).



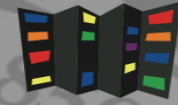
- Failures during container pulls that are attributed (in the error messages) to `*tar.gz` files are often due to corrupt `tar.gz` files that are downloaded while the image is being built from layers. Removing the offending `tar.gz` file will often solve the problem. Look for them in the `~/singularity` directory
- When building ubuntu containers, failures during `%post` stage of container builds from a recipe file can often be remedied by starting the `%post` section with the command “`apt-get update`”. As a best practice, make sure you insert this line at the beginning of the `%post` section in all recipe files for ubuntu containers.



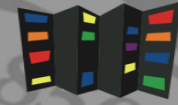
- Overlays are additional images that are "laid" on top of existing images, enabling the user to modify a container environment without modifying the actual container. Useful because:
 - Overlay images enable users to modify a container environment even if they don't have root access (though changes disappear after session)
 - Root users can permanently modify overlay images without modifying the underlying image.
 - Overlays are a likely way to customize images for different HPC environments without changing the underlying images.
 - More on overlays:
https://www.sylabs.io/guides/3.6/user-guide/persistent_overlays.html



- You've built your container on your laptop. It is 3 Gigabytes. Now you want to move it to CHPC to take advantage of the HPC resources. What's the best way?
- Containers are files, so you can transfer them to CHPC just as you would a file:
 - Command line utilities (scp, sftp)
 - Globus or rclone (recommended)
 - <https://www.chpc.utah.edu/documentation/software/rclone.php>
 - <https://www.chpc.utah.edu/documentation/software/globus.php>
 - For more on data transfers to/from CHPC:
 - https://www.chpc.utah.edu/documentation/data_services.php



- <http://sylabs.io>
- <http://cloud.sylabs.io>
- <https://www.chpc.utah.edu/documentation/software/containers.php>
- <https://github.com/CHPC-UofU>



Questions?