Refocusing on Research with Singularity/Apptainer

Ben Cardoen

Medical Image Analysis Research Group

Simon Fraser University

Complexity

- Essential
  - Open research problems: $\exists f : y' = f(x)$ s.t. $||y, y'|| < \epsilon$
- Accidental
  - Compute resources
  - Dependency management
  - Syntax/semantics
  - Reproducibility

Minimizing Accidental Complexity

- Solve a problem once (DRY)
    - If you need to do it 2+ times, automate
    - If it can't be automated, it's a waste of time
- Use tools that grant you full control/freedom
- Principle of least surprise
- Runs anywhere

Reproducible research $\rightarrow$ you get to focus on essential complexity (the stuff you're here for)

You compete with other researchers for resources.

In other words, you want to avoid:

- “But it works on my laptop”
- “Dear X, when trying to make your code work, …”
- Requesting 4xGPUs on a cluster, waiting for 2 days in the queue, only to have the job crash because numpy compiled versions mismatch
- Having to write a 3-page README on how to make your code work
- Your code works on Mondays, but otherwise it doesn’t. Or maybe it does.
- “Dear helpdesk, I need X and it works on my workstation but not in your cluster”
- Reviewer 2: “Can you try x=2?”
    - We tried, and failed, and also now x=1 doesn’t work anymore in Python 3.7+.

Singularity

An environment that offers

- Create it once, run $+\infty$ semantics
- 100% freedom (you're root)
- Read only (so no surprises)
    - If you don't change it, it never changes.
- 1-1 compatible with Docker / Open Container Image (OCI)
- Does the heavy lifting for you
- Easily automated
- Instructions to reproduce are `./myimage.sif --args`

Creating Containers

You create using recipes (simple text files)

```
singularity build myimage.sif myrecipe.def
```

You may need sudo to do this

If it works in a bash script/command line, it's a recipe

Creating Containers

Why not reuse what others have built ?

```
singularity pull image.sif docker://nvcr.io/nvidia/pytorch:22.08-py3
```

NVidia has an entire library of docker images

Singularity runs 1-1 with Docker

Your own recipe

```
Bootstrap: docker             ## Source: docker, shub, yum debootstrap, localimage, ...
From: fedora:35               ## Tag + version
%files                        ## If you need to include data/code
    localdir/localfile    containerdir/containerfile

%post                         ## Your instructions to tweak
    dnf install -y wget openssh-clients git g++
    cd /opt && git clone https://github.com/<you>/yourcode
    chmod u+x /opt/yourcode/installstuff.sh

%environment
    export LC_ALL=C

%runscript
    /opt/yourcode/runstuff.sh "$@" # Pass CLI args to script

```

Running containers

Executing commands

```
singularity exec myimage.sif python -c 'import torch'
```

Executing predefined scripts

```
singularity run myimage.sif
```

or shorter

```
./myimage.sif
```

Interactive use

You can open a shell inside the container

```
singularity shell myimage.sif
Singularity>
Singulartiy> python
>>>import torch
```

Interactively building/debugging

Changing the recipe line by line and rebuilding is boring and time consuming.

```
mkdir mydir
singularity build --sandbox mydir/ myrecipe.def
singularity --shell --writeable mydir/           # Container = folders
Singularity>
```

Fix and rebuild

```
dnf install python3 <CTRL-D>
singularity build myimage.sif mydir/
```

Ideally, you copy the fixes to your recipe, don’t share modified containers.

What if I only changed the `%environment`, do I need to rebuild it all?

```
singularity build --section environment ...
```

Debugging builds

My recipe dies at line 15 of %post and a rebuild = 15mins

1.Create a recipe minus line 15 `baseline.sif`

```
singularity build baseline.sif baseline.def
```

2. Create an interative container

```
mkdir test && singularity build --sandbox test
singularity shell --writable test
Singularity> Fix line 15
```

3. Fix your definition file (but not from scratch)

```
# Fixed.def
Bootstrap localimage
From      baseline.sif
%post
    line15 fixed
```

```
singularity build fixed.sif fixed.def
```

Writing in read only containers?

Sometimes you just need write access, for example, debugging, logging, history, …

Use writable overlays

```
singularity image.create overlay.img
singularity shell --overlay overlay.img container.img
```

Any changes you write are saved in `overlay.img`.

```
singularity shell container.img
## All is forgotten
```

Symlinking can be a workaround

```
%post

...

rm -rf /opt/mymodule/logs

 # Code will try to write to this

ln -s /tmp /opt/mymodule/logs # This WILL leak potentially private info
```

>=3.9

```
singularity overlay create --help
```

Mounting

By default Singularity accesses your `$HOME` only. Grant it more access by mounting

```
singularity shell --bind /localscratch2:/localspace myimage.simg
```

If the source and target are the same (name)

```
singularity shell -B /project myimage.simg
singularity shell -B /project:/project myimage.simg
```

```
[--bind|-B] source1[,source2]:target
```

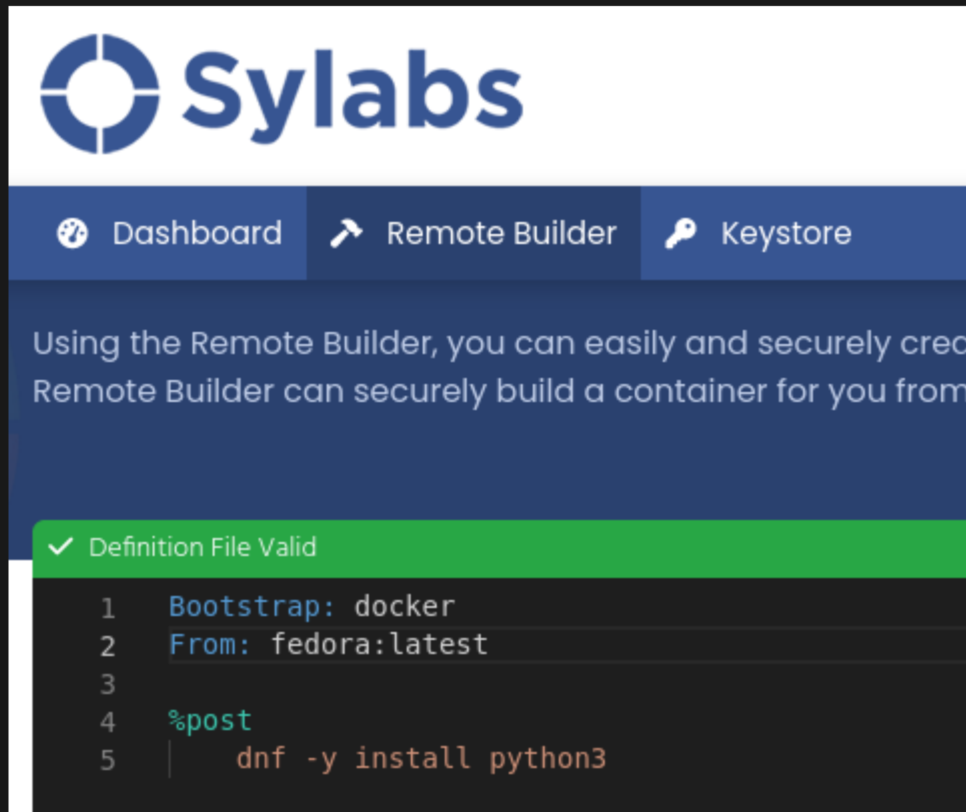
Using a runscript ENVs are easier

```
export SINGULARITY_BINDPATH="source:target"
```

If overlay is configured `target` does not need to exist, otherwise it needs to be an empty directory

Automation

You can automate building at Sylabs (free) https://cloud.sylabs.io/




```
singularity remote login
singularity build --remote ...
```

Github Actions/CircleCI can do this for you as well (if you need more resources)
https://github.com/singularityhub/circle-ci-sregistry

Environments

Your `~/.bashrc`, `module load X`, `conda activate` and other running systems pollute your environment in ways you may not want to pollute to your container.

Running with clean environment

```
singularity <cmd> -e myimage.sif
```

Note that this also unsets $USER, so ymmv.

Checking what a container defines

```
singularity inspect -e myimage.sif
```

Or

```
singularity shell -e myimage.sif
printenv
```

Note: $SLURM_{X} variables are passed with your env. If you set -e, then you'll likely lose them in the container.

Apps

What if you want to run multiple applications with your environment?

```
Bootstrap: docker
From: ubuntu
...
%apprun app1
    exec echo "One"

%appinstall foo
   exec /opt/configure1.sh

%apprun app2
    exec echo "Two"

%appinstall foo
    exec /opt/configure2.sh
```

Using apps

```
singularity run --app app1
```

(Multi) GPU

```
singularity <cmd> --nv <image>
```

If you need to control which GPUs are visible

```
export SINGULARITYENV_CUDA_VISIBLE_DEVICES=0
```

For newest versions works directly with NV Container layers

```
--nvccli
```

Encryption & Signing

Encryption

When your image / definition file is hosted on insecure storage

```
singularity build --passphrase encrypted.sif encrypted.def
singularity run --passphrase encrypted.sif encrypted.def
```

Signing

To prevent MITM attacks you can verify images (and sign them)

```
singularity verify [-all] image.sif
```

Generating keys

```
singularity key newpair # Gen new PEM keys
```

Finding keys

```
singularity key search thisuser
```

Signing keys

```
singularity sign [-all] myimage.sif
```

Bringing it all together – Cedar example

```
salloc --mem=32GB --account=X --cpus-per-task=8 --time=3:00:00 --gres=gpu:1
module purge
module load cuda
module load singularity
if [[ "$SLURM_TMPDIR" ]]; then export STMP=$SLURM_TMPDIR; else export STMP="/scratch/$USER"; fi
mkdir -p $STMP/singularity/{cache,tmp}
export SINGULARITY_CACHEDIR="$STMP/cache"
export SINGULARITY_TMPDIR="$STMP/tmp"
singularity pull image.sif docker://nvcr.io/nvidia/pytorch:22.08-py3
singularity exec --nv -B /scratch image.sif python -c 'import torch'
```

Notes

- Do not pull/build on login nodes
- Don’t pull inside compute jobs, pull once, then keep it local
- The default singularity cache is $HOME, always override this
- 8 cores: Singularity will (de)compress heavily using 8-9 cores, so give it what it needs

Fakeroot

```
singularity build image.sif image.def
FATAL:   You must be the root user, however you can use --remote or --fakeroot to build from a Singulari
```

Instead try

```
singularity build --fakeroot image.sif image.def
```

Checking programmatically if this is allowed to work:

```
cat /etc/subuid | grep $USER
```

Should list something like

```
<you>:100000:65536
```

See https://apptainer.org/admin-docs/master/user_namespace.html

Fakeroot remaps user and group ids so you (normal user) are mapped to root in the container. This needs explicit support on the host and configuration.

This will only work in restricted scenarios

Troubleshooting

Exclude stale files

```
--disable-cache
```

Fix pull errors from Docker

```
--docker-login
```

Allow overwriting existing images

```
--force
```

When permissions go haywire

```
--fix-perms # = chmod rwX****** for all content
```

I get a /tmp error and run out of space but I have enough space

```
mkdir -p $STMP/singularity/{cache,tmp}
export SINGULARITY_CACHEDIR="$STMP/cache"
export SINGULARITY_TMPDIR="$STMP/tmp"
```

When you want 100% isolation

```
--contain # Restricts access to filesystem
```

Used with

```
--workdir # working directory to be used for /tmp,/var/tmp and $HOME
```

Singularity configuration details

```
vi /etc/singularity/singularity.conf
```

ENV vars

```
SINGULARITY_DISABLE_CACHE=yes
SINGULARITY_CACHEDIR=. # layers, docker, shub cache
SINGULARITY_PULLFOLDE=. # Pulled images go here
SINGULARITY_LOCALCACHEDIR= # Non persistent (runtime) cache
```

Ask Singularity

```
singularity cache [list, clean]
```

Stacking Layers

Reuse in layers what you built earlier

Base container base.sif: NVidia PyTorch

Lab environment: add VTK, GCC

```
```bash!
BootStrap: docker
From: nvcio:pytorch
%post
    dnf install -y vtk-devel gcc
```
```

Project 1 needs Matlab runtime

```
```bash!
BootStrap: shub
From: mylabimage:latest # Torch + vtk + gcc

%post
    wget matlab-runtime.tgz && tar -xf
```
```

Project 2 needs 1 + Julia

```
```bash!
BootStrap: shub
From: mylabimage_matlab:latest # Torch + vtk + gcc + matla
%post
    dnf -y install julia
```
```

Remember: Don’t repeat yourself, automate, share

Use case: reproducibility can gain you time as well

Singularity gives you 100% control, so you you can specialize/optimize.

Example pipeline with Julia (1 cell = 2000x2000x70x3)

- Without Singularity: 100s/cell
- With Singularity: 10s/cell
- With Singularity optimized: 1s/cell

Your container is 100% free for you to optimize:

Precompile your code ahead of time (Numba), install tuned versions of libraries, strip libraries, use static images

The container is 1 compressed image $\rightarrow$ fastest IO

Example common workflows

Development/Prototyping

```
- base image (stable deps, torch, np)
- writeable overlay for moving deps
- -B mycode:mycode for mounted code
```

Deployment to Cedar/Solar/Cluster

```
base + dependencies + code in image
image automated with CircleCI/Travis/SingHub/Docker/...
```

Monolith

```
Image +
    - app "train"
    - app "inference"
    - app "relatedwork"
    - thesis
    - ...
```

Best practices

- **Stick to definition files as final product**
  - **Mutable containers are necessary, but temporary**
- **Stick to versions:**
  - **fedora:35 vs fedora:latest = safe vs roulette**
  - `{dnf|apt} -y update` is also a fun way to lose a weekend
  - Linux Kernel does not break userspace
  - User space breaks userspace *all the time*
- **Build incrementally**
  - **Don’t have a 3000 line definition file**

Resources

Singularity docs

https://docs.sylabs.io/guides/3.5/user-guide/fakeroot.html

Apptainer

https://apptainer.org/

DRI Singularity docs

https://docs.alliancecan.ca/wiki/Singularity

NVidia example on multigpu

https://developer.nvidia.com/blog/how-to-run-ngc-deep-learning-containers-with-singularity/

Slurm+Cuda example

https://github.com/bencardoen/singularity_slurm_cuda

Setting up Conda + Singularity

https://github.com/ds4dm/singularity-conda