Software Portability for HTC

Christina Koch

Slides adapted from Andrew Owen, Rachel Lombardi

OSG User School 2026

Goals for this session

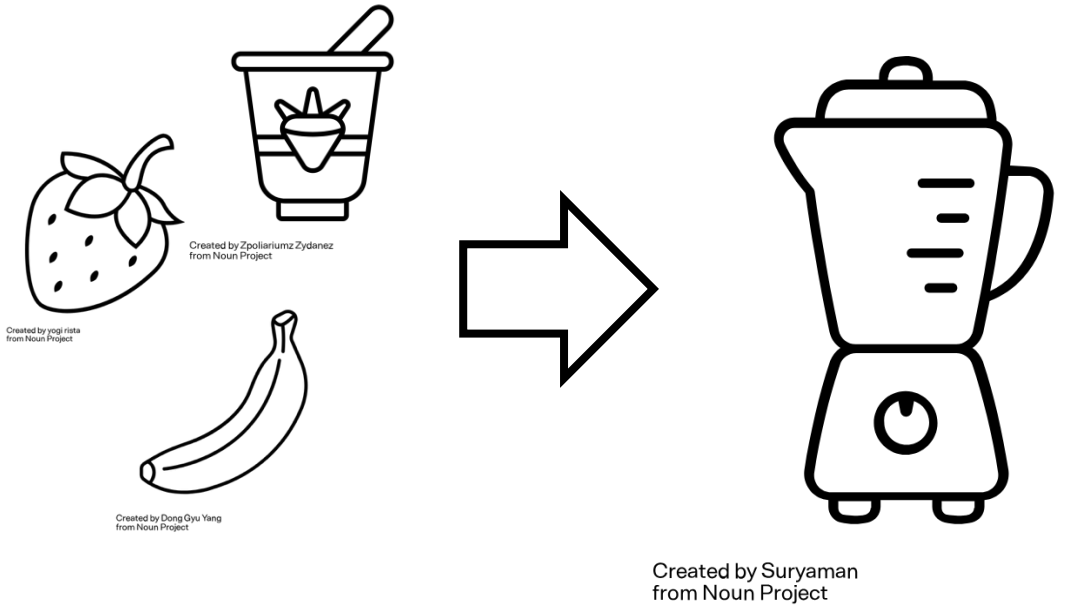
- Describe what it means to make software “portable.”
- Describe what software (and data) is used by your research work
- Use a Docker or apptainer container in a job
- Build a conda, python, or R based container with packages


Imagine that we asked you to
make a smoothie…

A computing smoothie

You need ingredients (fruit, yogurt, etc.) and equipment (a blender).

In computing for research, we need ingredients (data/scripts) and equipment (software) to generate results.



Data, scripts

Software

Software requirements

- Specifically you need a kitchen that (a) has a blender (b) that you can find.
- Just like you need your software files in a known location.

Working on your computer

Is like making food in your own kitchen

- You know what is there.
  - All the software you need is already installed.
- You have the right versions.
- You know where everything is (mostly).
- You have full control.
  - You can add new programs when and where you want.

OSPool: Other People's Computers

Working on someone else's computer

Like trying to cook in someone else's kitchen.

- What's already there?
  - Is R installed? Or Python? What about the packages you need?
- If the software you need is installed, do you know where it is or how to access it?
- Is it the right version?
- Are you allowed to change whatever you want?

Software portability

- Imagine going camping or backpacking – what do you need to do to cook anywhere?
- Similarly: what would it take to drop into any computer and be able to run your software there?
- This is what it means to make software portable.

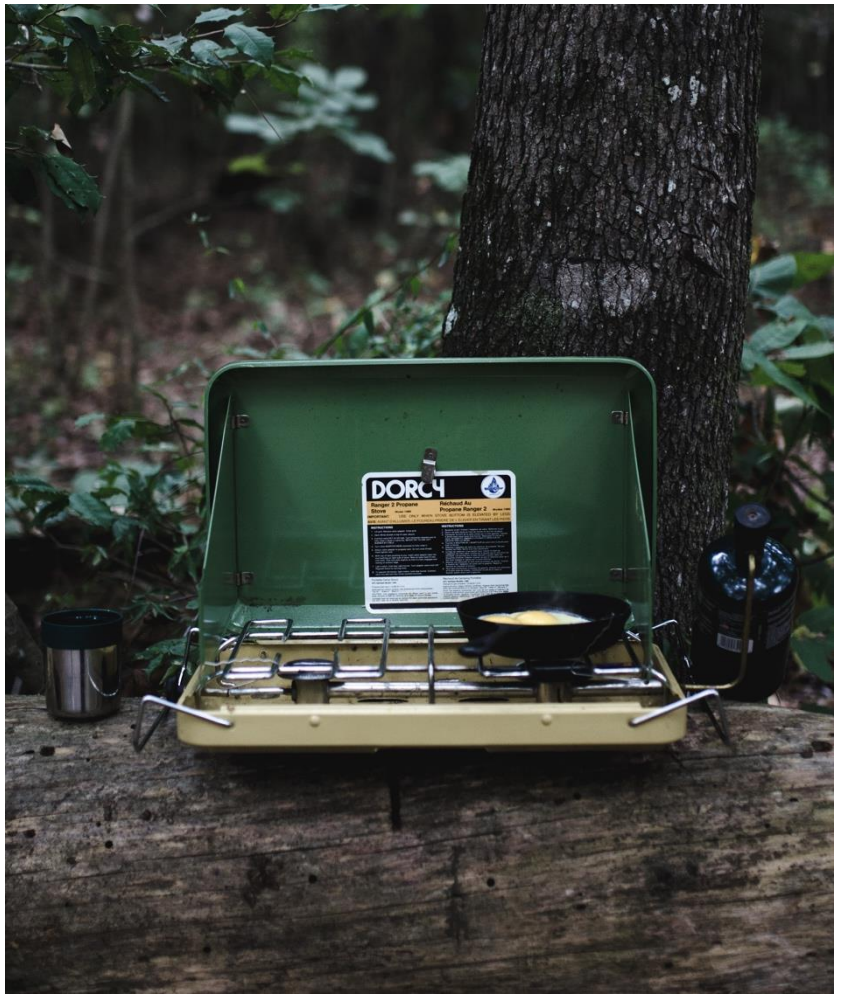


Photo by andrew welch on Unsplash

Enter containers

Photo by PunkToad on Flickr, CC-BY

Using a container is like bringing along a whole kitchen, with all the specific equipment we need.

Why use containers?

Containers provide

- portability
  - have ALL your files
  - in the right places
- user control – install software like on your computer
- ease of use – add one line to your submit file

Using containers in jobs

For today:

```
container_image = cowsay.sif

...usual submit options...
```

That's it!

Using containers in jobs

Well… all other times:

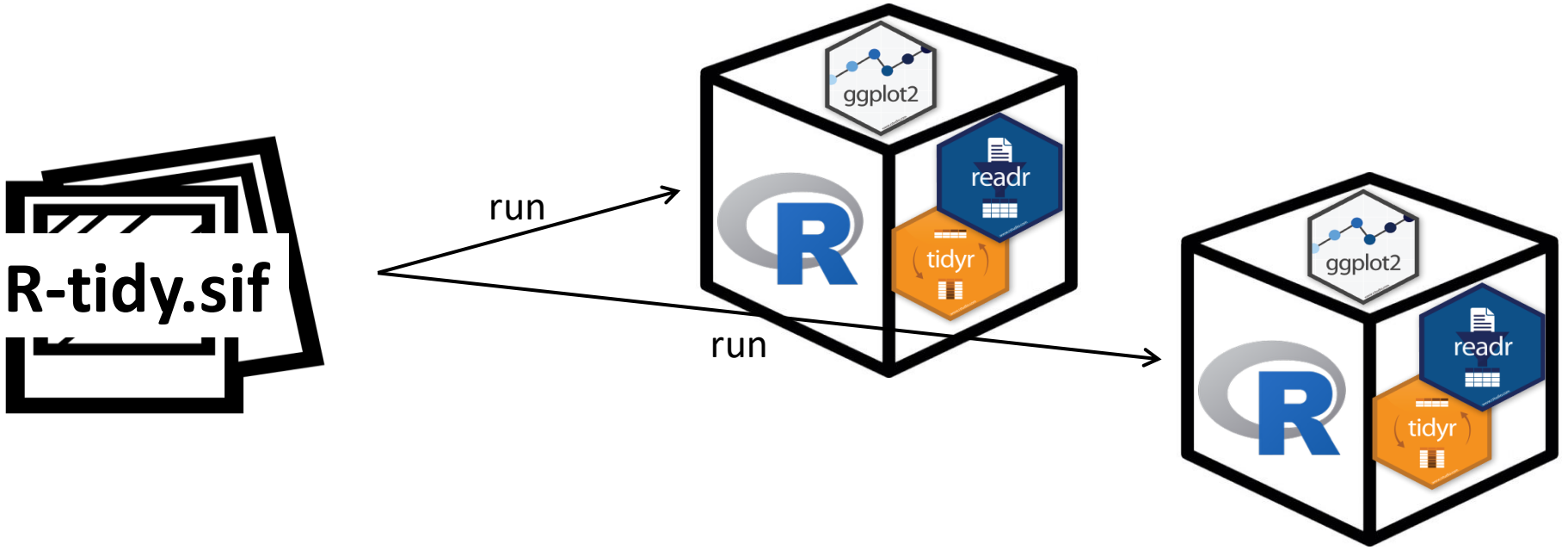
```
container_image = osdf:///ospool/ap40/username/cowsay.sif

...usual submit options...
```

More on OSDF tomorrow!

Containers

Containers are a tool for capturing an entire “environment” (software, libraries, operating system) into an “image” that can be run and used as the environment for a job



Credit: polaroid photos by Nick Bluth from the Noun Project

Words for containers

container = the actively running ~thing~ that can run commands

container image = is the set of files that can be used to create the container

"Launching" or "running" a container is the process of using the "container image" to create the running "container"

"Building" is the process of creating the "container image" (and often uses a pre-existing container image as the basis for the new one)

Technology for containers

https://apptainer.org/

https://www.docker.com/

For beginners and those who only want to run the container on the OSPool

For advanced users and those who want to deploy containers to many places

Technology for containers

https://apptainer.org/

+ Open source software
+ Does not require root to install
+ Beginner friendly syntax
- Fewer features
- No distribution system
- Harder to run on your laptop

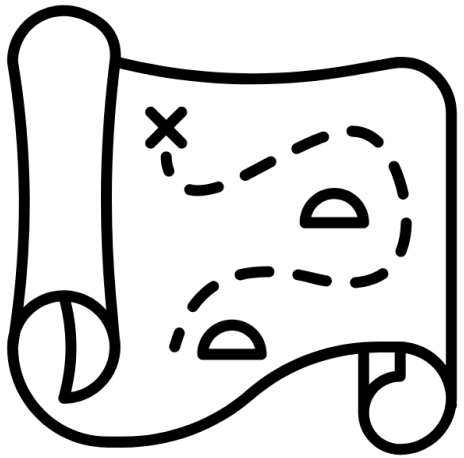
https://www.docker.com/

+ Modern app interface (Docker Desktop), can be run on your laptop
+ Widely used distribution system (DockerHub, analogous to GitHub)
- Non-intuitive syntax
- Requires root to install
- Commercial software

Demo

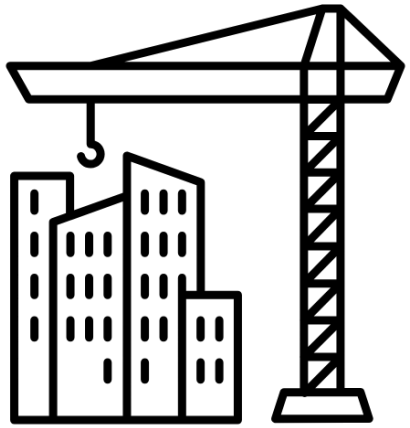
Getting started with containers

Find an existing container



Created by Azland Studio
from Noun Project

Build your own container



Created by Ferdian Mauladi Riziq
from Noun Project

Find existing containers

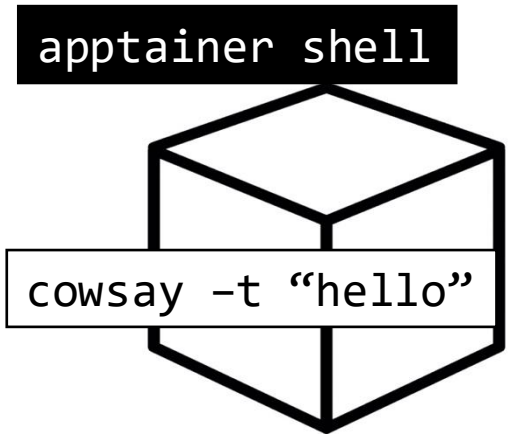
- Review your software's web page / documentation
- Go to common container repositories and search for tools. Opt for containers that are:
  - Regularly updated
  - Owned by official community organizations
- (See Exercise 2.2)

Effort estimates for building containers

- How hard was it to install this software initially?
- (How hard would it be to install on a different computer?)
- Building a container should be about the same amount of challenge/effort, not a lot more.

How it works

Create a definition file	**Build** a container image	**Test** the container image	**Deploy** in jobs!

`Bootstrap: docker`
`From: python:3.13`
 	
`%post`
`python3 -m pip install cowsay`
			`container_image = cowsay.sif`

Apptainer definition file

Create a file `container.def` with the following contents:

```
Bootstrap: docker
From: python:3.13

%post
    python3 -m pip install cowsay
```

Tells Apptainer to use the DockerHub "python" container with the tag of "3.13"*

The following commands should be used to modify the container

Install the "cowsay" Python package

*In the case of the "python" container, the "tag" equals the version of python inside.

Choosing a base container

Search for your software + "container".

- Most big-name softwares have official containers online

Once identified, find the container "address", usually represented in the form of a `docker pull` command:

```
docker pull user/repository:tag
```

Do not use the `latest` tag - choose an explicit one

Installation commands in %post

- **Level 1**
  - Use common packages like those from PyPi or CRAN; build environments with conda. → Exercise 2.3
- **Level 2**
  - Use the Linux built in package manager (`apt` for Debian/Ubuntu, `yum` or `dnf` for Red Hat family). → Exercise 2.4
- **Level 3**
  - Run a series of installation commands that downloads or clones files, unzips files, runs an installation script or compiles code. Set environment variables as needed. → Exercise 2.5

Brief side quest

- A software program is a **FILE**.
- Linux expects software files to be in certain default **PLACES**.
- **Level 1 + 2 installation**
  - The package manager downloads the correct files and puts them in the right place.
- **Level 3 installation**
  - YOU have to make sure the files are downloaded, compatible with your version of Linux/your hardware, and you have to know where they are.

Demo

Installation commands in %post

- **Level 1**
  - Use common packages like those from PyPi or CRAN; build environments with conda. → Exercise 2.3
- **Level 2**
  - Use the Linux built in package manager (`apt` for Debian/Ubuntu, `yum` or `dnf` for Red Hat family). → Exercise 2.4
- **Level 3**
  - Run a series of installation commands that downloads or clones files, unzips files, runs an installation script or compiles code. Set environment variables as needed. → Exercise 2.5

90% of software installs

What to install and where

"Common" tools/libraries/packages are rarely included in "official" containers; be sure to add what you need/want

Dynamic items (scripts, input data) should not be included in the container

Recommend that user software is installed in the `/opt` folder

Build the container image

Build the container image file using the provided definition file

```
apptainer build container.sif container.def
```

desired filename

definition file

Apptainer build output

Output details what Apptainer is doing:

1. Download the base container image
2. Launch the base container image
3. Execute the commands from the `%post` section
4. Create the new container image file

When finished, should see a message like

```
INFO: Build complete: container.sif
```

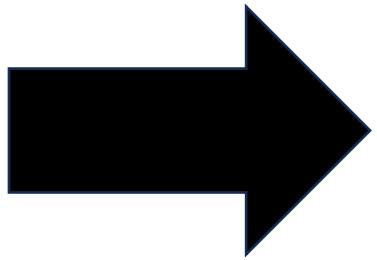
Test the Apptainer image

Test the `container.sif` container by launching it with

```
apptainer shell -e container.sif
```

If you can get the "help" or "version" text for your desired program, should be good to go.

Test the Apptainer image



```
Apptainer> cowsay "hello"
 _______
| Hello |
 =======
        \
         \
            ^__^
           (oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||
```

```
$ apptainer shell cowsay.sif
```

Test container image

Differences between local & cluster use

With HTCondor,

containers are always run ***as the user*** and ***never as root***

But…

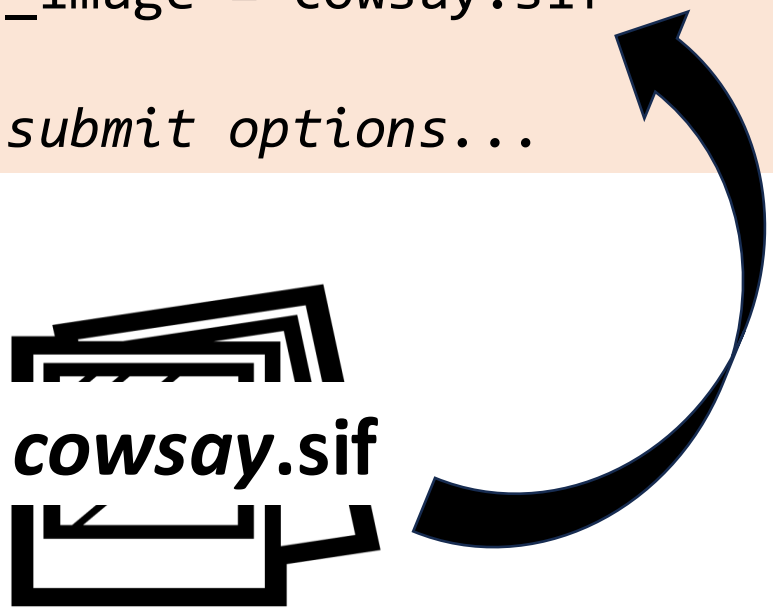
official containers often assume you are running ***as root***!

When testing a container, ***test as the user***.

Use the Apptainer image

```
container_image = cowsay.sif

...usual submit options...
```



Back to our cooking analogy…

Photo by Derrick Mercer on Flickr, CC-BY-SA

It is possible to use a backpacking approach (bring along files) instead of a portable kitchen (container).

Bring along software files

We always need a "compiled" file that is compatible with the version of Linux used by the execution point. Some options:

- Download pre-compiled software files
- Compile software yourself
  - Generate a single binary file
  - Create an installation with multiple binary files contained in a single folder
- If you think your use case may benefit from this approach (instead of a container) – talk to School staff.

Software installation pep talk 

Software installation can be hard…and you can do it!

- Make sure to read the instructions for installing your software, especially the requirements
- The last error message you see may not be the same as the first error message that caused it..
- Test, test, test!

Work Time

1. Go through the introductory exercises
2. Then, choose an approach for *your* software and try to find or make a portable version for OSPool jobs.
3. Ask for help!

Additional guides are at portal.osg-htc.org/documentation

Acknowledgements

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

Appendix: Container and Compiling Tips

Best Practices in Using Containers on OSG

- Don’t use the **latest** tag in images
- Use **version number**/specific names in the images
- Test images with **apptainer shell**
- **Unique** image name eliminates the risk of running a job using previous versions due to stashing.

Where to install software?

- Do not use $HOME, /root or /srv
  - Container will run as some user we do not know yet, so $HOME is not known and will be mounted over
  - /root is not available to unprivileged users
  - /srv is used a job cwd in many cases
- /opt or /usr/local are good choices

Common issues

Building Docker on a Mac (with ARM)
→ Make sure to include option `--platform linux/amd64`

Custom command not found
→ Manually add install location to your `PATH` at start of execution

Cannot create directory at `/.cache`
→ Manually set `HOME` or other environment variables at start of execution

Cannot write file at `/app/data`, etc.
→ Change program to use relative path for execution, not absolute

Finding files in the command line

Put the software location in the PATH variable

- The PATH variable is a list of folders on your computer.
- When you type a command in the terminal, behind the scenes it checks each folder in the path for those files.
- You can add folders to the PATH

Give a relative/absolute path to the software location

- You can also type *exactly* where the software files or your scripts are.
  - **Relative**: starts with ./ or ../
  - **Absolute**: starts with /

Learn how to build/use containers

- OSG User Documentation:
  https://portal.osg-htc.org/documentation/

- Videos:
  - https://portal.osg-htc.org/documentation/support_and_training/training/materials/
  - https://www.youtube.com/watch?v=awSLTflAIJ8