Reproducible Research using Containers (Apptainer/ Singularity)

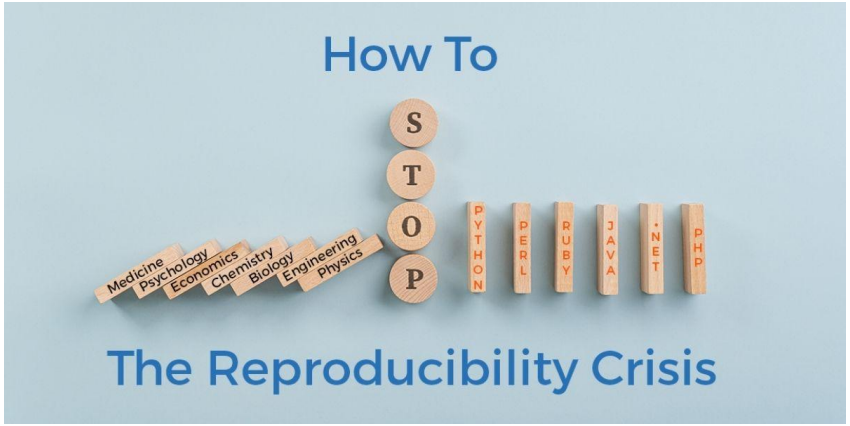
Mir Junaid

January 31, 2023

Contents

Reproducibility in Science

- More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments
- 31% think that failure to reproduce published results means that the result is probably wrong but most say that they still trust the published literature
- Containers is a way forward for computational reproducibility



Introduction to Containerization

Introduction to Containerization

- Fast-paced development of computational tools has enabled tremendous scientific progress in recent years
- However, this rapid surge of technological capability also comes at a cost
- It leads to an increase in the complexity of software environments and potential compatibility issues across systems
- Advanced workflows in processing or analysis often require specific software versions and operating systems to run smoothly
- Discrepancies across machines and researchers can prevent/delay reproducibility and efficient collaboration
- As a result, scientific teams are increasingly relying on containers to implement robust, dependable research ecosystems

Introduction to Containerization

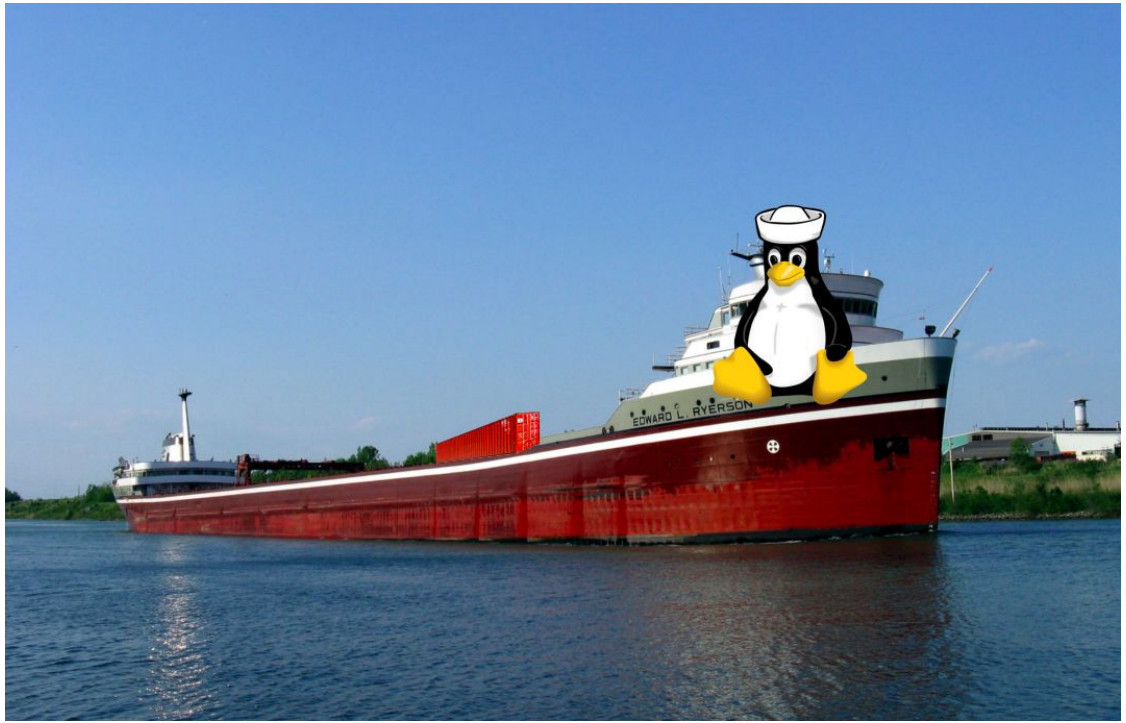
- Originally popularized in software engineering, containers have become common in scientific projects, particularly in large collaborative efforts
- Containers store the software and all of its dependencies (including a minimal operating system) in a single image so that there is nothing to install and when it comes time to run the software
- Everything "just works"
- This makes the software both shareable and portable while ensuring reproducibility
- Containerization allows applications to be “written once and run everywhere”
- In a nutshell, containers are encapsulations of system environments

Introduction to Containerization

- Container technologies have been designed for the enterprise computing

Containers for HPC

- Our use case is the opposite of enterprise computing



Containers for HPC

- Scientists are like Pirates, pillaging for resources instead of booty!
- We want to run our jobs. We want to get results.
- When we find available resources, we need to ensure application and environment compatibility
- This is where containers can be a perfect fit
- But as I mentioned, our use-case and needs are different from enterprise…

What about Docker?

- Docker is the most well known and utilized container platform
- Designed primarily for network and micro-service virtualization
- Facilitates creating, maintaining and distributing container images
- Containers are kinda reproducible
- Easy to install, well documented, standardized

For these reasons, it appears to be the solution.

So why not just keep using about Docker?

- The good news
  - You can! It works great for local and private resources
  - You can use it to develop and share your work with others using Docker-Hub
- The bad news
  - If you ever need to scale beyond your local resources, it maybe a dead end path
  - Docker, and other enterprise focused containers, are not designed for, efficient or even compatible with traditional HPC
  - No HPC centers allow it
- Docker images are not secure because they provide a means to gain root access to the system they are running on

Apptainer/Singularity Containers

- Designed from necessity, Apptainer/Singularity is an alternative to Docker that is both secure and designed for HPC
- Singularity started as an open-source project at Lawrence Berkeley National Laboratory in 2015
- First public release in April 2016
- Created for and by the people who need and use it
    - Scientists, HPC Engineers, Linux Developers
- Tighter integration with other scientific apps (SLURM, MPI, etc.)
- Singularity/Apptainer is compatible with all Docker images and it can be used with GPUs and MPI applications
- Integration with other container technologies

Apptainer/Singularity: Design Goals

- Single file based container images
  - Facilitates distribution, archiving, and sharing
  - Very efficient for parallel file systems
- No system, architectural or workflow changes necessary to integrate on HPC
- Limits user's privileges (inside user == outsider user)
- No root owned container daemon
- Simple integration with resource managers, Infiniband, GPUs, MPI, file system, and supports multiple architectures (x86_64, PPC, ARM, etc.)

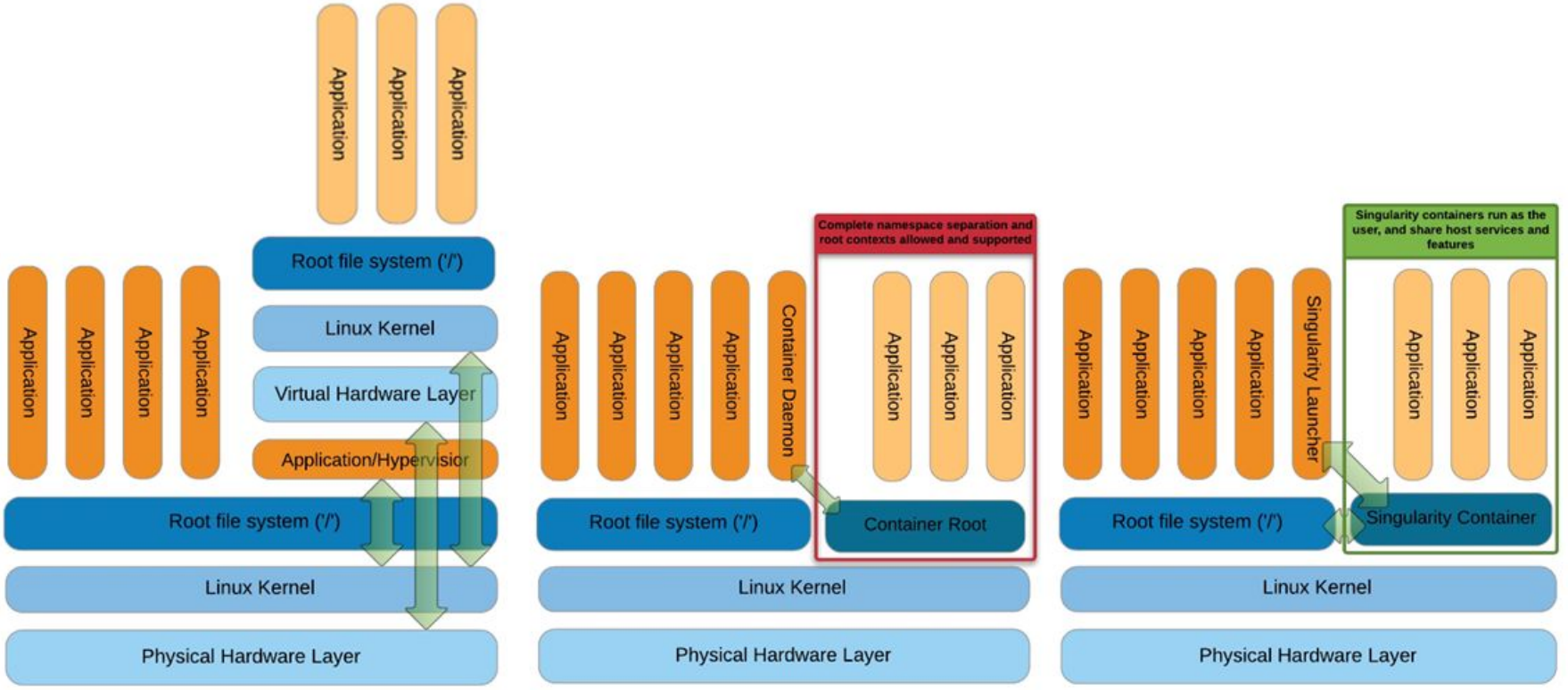
Apptainer/Singularity: Access and Privilege

- User contexts are always maintained when the container is launched
- When launched by a particular user, the programs inside will be running as that user
- Any escalation pathways inside a container are blocked
- Thus, if you want to be root inside the container, you must be the root outside the container!

Popular Container Registries

- Docker Hub
- NVIDIA GPU Cloud
- Singularity Cloud Library
- Singularity Hub
- Quay.io
- **BioContainers**
- IBM PowerAI (Traverse only)
- AMD InfinityHub (AMD GPUs)

VM vs. General Container vs. HPC Container



Apptainers on GLiCID

- On GLiCID, you just have to load the Apptainer module

```
$ module load guix          $ guix install squashfs-tools
$ module load apptainer/1.1.6
or
$ cd /opt/software/glicid/containers/apptainer/bin
```

The Apptainer Command Line Interface

- Apptainer provides a CLI to interact with the containers
- You can search, build, or run containers in a single line
- To check the version of the Apptainer or Singularity you are using

```
$ module load apptainer
$ apptainer --version
apptainer version 1.2.2
```

- You can check the available options and subcommands using

```
$ apptainer --help
```

Downloading Images

- Downloading an image from the Container Library is pretty straightforward and the image is stored locally as .sif file (Singularity Image Format)

```
$ apptainer pull docker://alpine
$ apptainer pull docker://python (Try this one if you don’t have Python)
$ Apptainer> python
```

- Apptainer is also compatible with Docker images

Running Containers

- Initializing a shell and exiting it

```
$ apptainer shell docker://alpine
$ Apptainer>
$ Apptainer> id
uid=1001(jmir) gid=1001(jmir) groups=1001(jmir)
$ Apptainer> exit
```

Running Containers

- The command exec starts the container from a specified image and executes a command inside it

```
$ apptainer exec docker://alpine cat /etc/os-release
```

TP 1: Fun with Containers

```
$ module load apptainer
$ apptainer --version
$ appatiner --help
$ apptainer pull docker://alpine
$ apptainer shell docker://alpine
$ Apptainer > whoami
$ Apptainer > id
$ Apptainer > ls
$ Apptainer > cat /etc/os-release
$ Apptainer > exit
```

Try these commands after exiting the container and know the difference

Bind Paths and Mounts

- When Apptainer swaps the host operating system for the one inside your container, the host file becomes inaccessible.
- However, you may want to read and write files on the host system from within the container.
- Apptainer allows to map directories on your host system to directories within your container using **bind mounts**
- This allows you to read and write data on the host system with ease
- To enable this functionality, Apptainer will bind directories back into the container via two primary methods:
  - system-defined bind paths
  - user-defined bind paths

Bind Paths and Mounts

- System-defined bind paths
  - System admin has the ability to define what bind paths will be included automatically inside each container
  - Some bind paths are automatically derived (eg., a user's home directory) and some are statically defined (e.g., bind paths in Apptainer configuration file).
  - In the default configuration, the system default bind points are $HOME, /sys:/sys, /proc:/proc, /tmp:/tmp, /var/tmp:/var/tmp, /etc/resolv.conf:/etc/resolv.conf, /etc/passwd:/etc/passwd, and $PWD.
  - Here the first path before **:** is the path from the host and the second path is the path in the container
  - You can disable the system bind paths using --no-mount flag.
  - For example, if admin has configured apptainer.conf to always mount /data, you can disable this with

    ```
    $ apptainer run --no-mount /data mycontainer.sif
    ```

  - To disable all bind path entries set in apptainer.conf, use --no-mount bind-paths

    ```
    $ apptainer run --no-mount bind-paths mycontainer.sif
    ```

Bind Paths and Mounts

- User-defined bind paths
  - Unless system admin has disabled user control of binds, you will be able to request your own bind paths within your container.
  - The Apptainer action commands (run, exec, and shell) will accept the --bind/-B command-line option to specify bind paths
  - Here's an example of using --bind option and binding /scratch on the host to the container

```
$ apptainer shell --bind /scratch/nautilus/users/user_name mycontainer.sif
```

Building Containers

- Apptainer Definition File (or “def file”) is like a set of blueprints explaining how to build a custom container
- It includes
  - specifics about the base OS to build or the base container to start from
  - software to install
  - environment variables to set at runtime
  - files to add from the host system, and container metadata
- Apptainer Definition file is divided into two parts, Header and Sections

Building Containers

- Header
  - It describes the core operating system to build within the container
  - Configure the base operating system features needed within the container
  - Specify the Linux distribution, the specific version, and the packages that must be part of the core install (borrowed from the host system).
- Sections
  - Each section is defined by a % character followed by the name of the particular section
  - All sections are optional, and a def file may contain more than one instance of a given section

Building Containers

- The following recipe shows how to build and run a hello-world container

Step 1. Open a text editor

```
$ nano hello-world.def
```

Step 2. Include the following script in the hello-world.def file to define the environment

```
BootStrap: docker
From: ubuntu:20.04
%runscript
echo "Hello World"
# Print Hello World when the image is loaded
```

- BootStrap: docker indicates that apptainer will use the docker protocol to retrieve the base OS to start the image
- From: ubuntu:20.04 is given to apptainer to start from a specific image/OS in docker Hub
- Any content within the %runscript will be written to a file that is executed when one runs the apptainer image
- The echo "Hello World" command will print the Hello World on the terminal
- Finally the # hash is used to include the comments within the definition file

Building Containers

Step 3. Build the image

```
$ apptainer build hello-world.sif hello-world.def
```

Step 4. Run the image

```
$ ./hello-world.sif
```

TP 2: Build from Scratch

- Create a Hello-World Container
- Use the definition file to create a container
- Interact with container

Miniconda3 on GLiCID Cluster

- Let’s create a Apptainer container based on the specified Docker image
- Setting up an environment with Miniconda and additional configurations,
- and running a Python script when the container is executed

Miniconda3: Definition File

```
Bootstrap: docker
From: ubuntu:22.04

%help
  This container provides a Python script and research data. To run the script:

    $ apptainer run myimage.sif  # or ./myimage.sif

  The script is found in /ml-container/scripts and the data is found in /ml-container/data.

%labels
  AUTHOR_NAME Junaid Mir
  AUTHOR_EMAIL junaid.mir@ec-nantes.fr
  VERSION 1.0

%environment
  export PATH=/opt/miniconda3/bin:${PATH}
  # set system locale
  export LC_ALL='C'

%post -c /bin/bash
  apt-get -y update && apt-get -y upgrade
  apt-get -y install wget

  INSTALL_SCRIPT=Miniconda3-py38_4.9.2-Linux-x86_64.sh
  wget https://repo.anaconda.com/miniconda/${INSTALL_SCRIPT}
  bash ${INSTALL_SCRIPT} -b -p /opt/miniconda3
  rm ${INSTALL_SCRIPT}
  /opt/miniconda3/bin/conda install pandas -y

  # cleanup
  apt-get -y autoremove --purge
  apt-get -y clean

%runscript
  python /home/jmir@ec-nantes.fr/ml-container/scripts/myscript.py

%test
  /opt/miniconda3/bin/python --version
```

Miniconda3: Definition File

- Let's break down the different sections of the definition file

1. Bootstrap: docker

   - Specifies that the container should be built using a Docker image as the base.

2. From: ubuntu:22.04

   - Specifies the base Docker image to use, in this case, Ubuntu 22.04.

3. %help

   - This section provides information on how to use the container. Here, it gives instructions on running the Python script inside the container.

4. %labels

   - These are metadata labels for the container, providing information such as the author's name and email, and the version of the container.

Miniconda3: Definition File

5. %environment

- This section sets environment variables within the container. It adds the Miniconda3 binary path to the `PATH` variable and sets the system locale.

6. %post -c /bin/bash

- This is a script that runs during the container build process. It updates the package manager, installs `wget`, downloads and installs Miniconda3, installs the pandas package using conda, and then performs cleanup.

7. %runscript

- This specifies the command that will be executed when the container is run. In this case, it runs a Python script located at `/home/jmir@ec-nantes.fr/ml-container/scripts/myscript.py`.

8. %test

- This section provides a test command to check if the container is working correctly. It checks the version of Python installed in the container.

Slurm Script

```
#!/bin/bash
#SBATCH --job-name=myjob          # Name for your job
#SBATCH --comment="Run My Job"    # Comment for your job
#SBATCH --time=0-00:05:00         # Time limit
#SBATCH --nodes=1                 # How many nodes to run on
#SBATCH --ntasks=2                # How many tasks per node
#SBATCH --cpus-per-task=2         # Number of CPUs per task
#SBATCH --mem-per-cpu=10g         # Memory per CPU
#SBATCH --qos=short               # priority/quality of service

hostname                          # Run the command hostname

cd /home/jmir@ec-nantes.fr/ml-container
module purge
module load apptainer/1.1.6
apptainer --version
./myimage.sif                     # run the container
```

TP 3: dl-container

- Create a Miniconda Container
- Create a Slurm script
- Submit the Job
- Monitor the job
- Check the results

```
$ apptainer shell --bind /scratch/nautilus/users/username myimage.sif
$ python myscript.py
```

Thank you

Any Questions?