On the Hellgate Cluster, we utilize Apptainer, an OCI-compliant containerization platform, enabling it to work well with Docker and other container images.

Containers offer a flexible, efficient and secure way to run applications on Hellgate. This enables us to easily and quickly create environments for research programs which may not be able to run on Hellgate’s native environment.

More information can be found at [Apptainer’s Documentation](https://apptainer.org/docs/user/main/introduction.html).

# Definition Files

An Apptainer Definition File (or “def file” for short) is a set of blueprints explaining how to build a custom container. It includes specifics about the base OS to build with or a container to start from, software to install, environment variables, files to add from the host system, and container metadata. These are usually given a file extension of .def to signify it is a definition file.

## Example of a Definition File

We create an Apptainer definition file named definition.def:

nano definition.def

We include these lines of code:

### HEADER ###

Bootstrap: docker

From: ubuntu:20.04

### SECTIONS ###

%environment

## Define environmental variables at runtime, such as paths or variables

%post

## Execute during build time (After OS install)

apt-get update

apt-get upgrade -y

Definition files are divided into two parts, Header and Section.

* Header: defines the core OS to build.
* Sections: Optional, executed at build time to further configure the container.

## Building Apptainer Container

Building an Apptainer container involves using the apptainer build command, a versatile tool for creating containers. You can use it to download and assemble containers from external OCI-compliant resources (like Docker Hub) or creating a container from scratch with a definition file. Notice the container name comes first, then the definition file name:

apptainer build container.sif definition.def

Apptainer Images are usually given an .sif file extension to signify it is an Apptainer Image.

More information can be found at [Apptainer’s Definition Documentation](https://apptainer.org/docs/user/latest/definition_files.html).

## Finding Definition of Existing Container

Apptainer inspect shows labels, environment variables, apps and scripts associated with the container image.

apptainer inspect --deffile container.sif

# Sandbox

Sandboxing is used to provide a read-write container within a directory. It is useful for the development of a container by allowing for interactive changes and testing. Once development and testing are finished, it can be converted into an Apptainer image.

More information can be found at [Apptainer’s Sandbox Page](https://apptainer.org/docs/user/latest/build_a_container.html#creating-writable-sandbox-directories).

## To create an Apptainer Sandbox Directory

Creating an Apptainer sandbox is the same as building an apptainer container, with the added option of --sandbox.

apptainer build --sandbox sandbox definition.def

This will create a directory in your current working directory and can be used like a container.

## Making persistent changes

The apptainer shell --writable command is used to start an interactive shell within a writable Apptainer container, often referred to as a sandbox. This allows you to easily modify the container’s filesystem, install new software, and make other changes as if you were working within a regular Linux environment.

apptainer shell --writable sandbox

You may find yourself having issues with insufficient permissions. To get around this Apptainer offers an option of --fakeroot to give users escalated privileges in the container.

apptainer shell --fakeroot --writable sandbox

## Converting sandbox into a container

To convert an Apptainer sandbox into a .sif file, you can use the apptainer build command. This process will create a compressed, read-only Singularity Image File (SIF) from your writable sandbox directory.

apptainer build container.sif sandbox

It is recommended to create a .def file to document what was configured on the container and allow for easier reproducibility.

## Removing sandbox

When removing a sandbox, you may notice you do not have the right permissions to remove some files. We can change the permissions on the files to allow for removal.

chmod -R u\_rwX sandbox

rm -R sandbox

# Using Apptainer Containers

There are a couple of ways to use an Apptainer Container:

## Shell

The apptainer shell command is used to start an interactive shell within an Apptainer container. This is particularly useful for exploring the container’s environment and debugging. When you run this command, you are placed into the container’s environment, where you can execute commands as if you were inside a Linux environment.

apptainer shell container.sif

To exit the shell, you can simply type exit or press Ctrl+D.

## Exec

The apptainer exec command allows you to execute a specific command within an Apptainer container. This is useful when you want to run a single command or script inside the container environment without launching an interactive shell.

apptainer exec container.sif echo "hello"

## Run

The apptainer run command is used to launch an Apptainer container and execute a runscript if one is defined for that container. (%runscript in .def)

apptainer run container.sif

## Binding Directories

The --bind option in Apptainer is used to bind host system directories to directories within the container. This allows you to access files from your host system inside the container for processing.

apptainer shell --bind /mnt/beegfs/projects:/mnt/beegfs/hellgate/home/$USER container.sif

# Installing Software

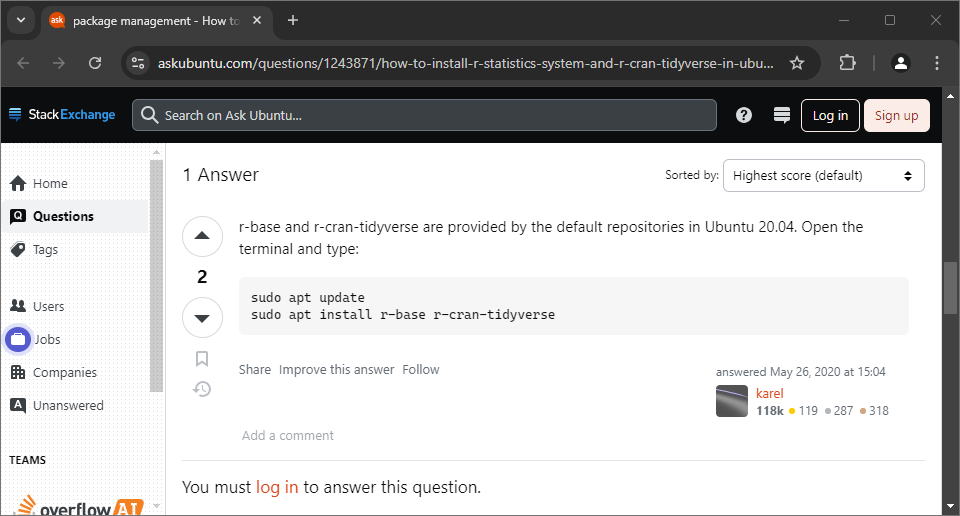
Here are some examples of using Apptainer.

## From Package Manager

We’ll demonstrate how to install software using the package manager within an Apptainer container. Specifically, we’ll install R along with the tidyverse package on an Ubuntu container.

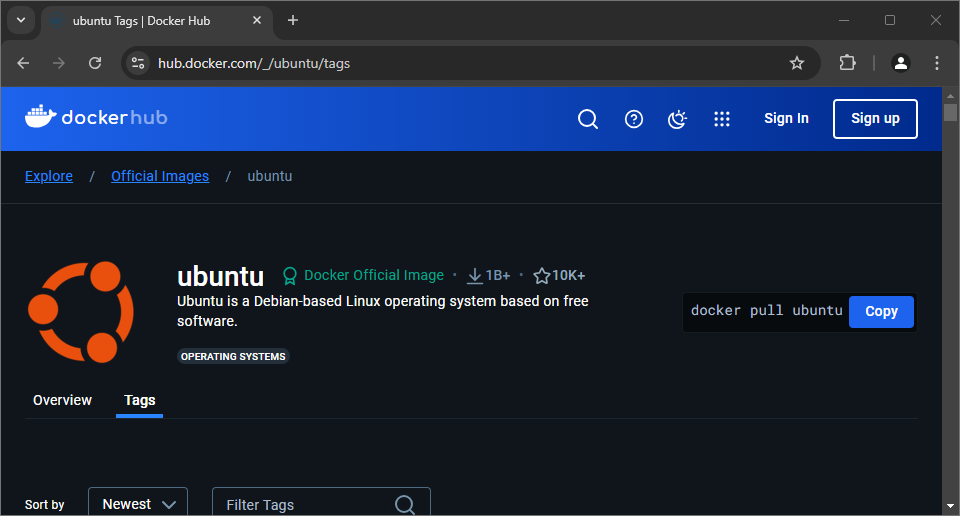
### Research

After some investigation, we discovered that tidyverse is available as an R package for Ubuntu. This allows us to leverage Ubuntu’s package manager for installation, simplifying the process significantly. The command apt install r-cran-tidyverse will handle the installation, as r-cran-tidyverse depends on r-base, which will be automatically included.

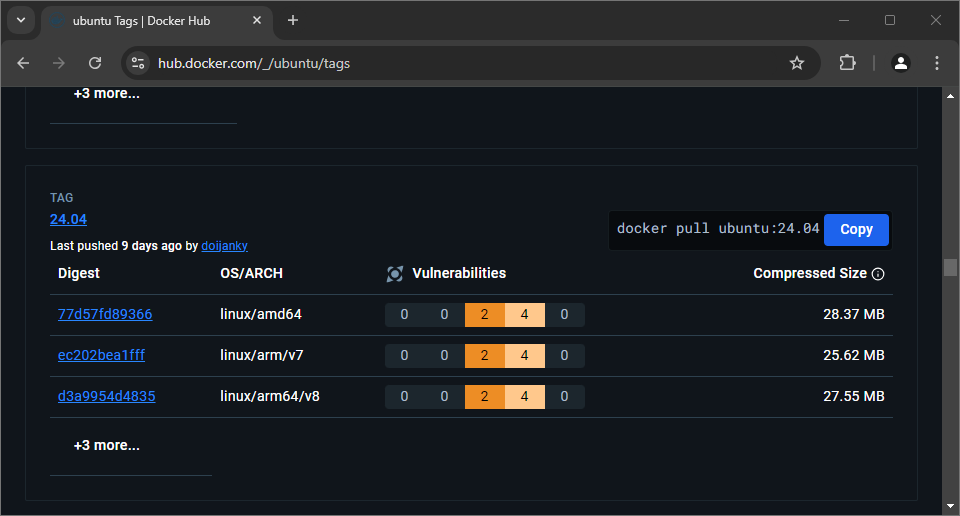


### Creating an Apptainer .def file:

To create a container, we need a .def file for Apptainer to start with. We choose Docker as the Bootstrap method since there is a wide variety of premade containers available. After exploring DockerHub, we found that Ubuntu offers an official container.



We can browse the Tags tab to find the desired version of Ubuntu. In this case, we chose 24.04, one of the newest versions. To pull the image using Docker, we would use ‘docker pull ubuntu:24.04’ in Apptainer, we just need the pull tag of ‘ubuntu:24.04’.



With this information, we can create a basic Apptainer container based on a Docker container. We’ll specify to Apptainer that we are using a Docker in the Bootstrap and the Ubuntu image we would like to use. The .def file should look like this:

Bootstrap: docker

From: ubuntu:24.04

Next, we need to run commands in the %post section of the .def file. To ensure the container’s package manager can find r-cran-tidyverse, we will update it with ‘apt-get update’ then apply the updates using ‘apt-get upgrade’.

Bootstrap: docker

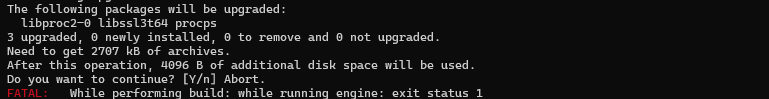
From: ubuntu:24.04

%post

apt-get update

apt-get upgrade

When building this, we find that Apptainer does not handle commands requiring user input well.



The error indicates that apt-get needs us to verify the download. We can bypass this by adding the -y option to automatically say we would like to. This should allow the build to succeed.

Bootstrap: docker

From: ubuntu:24.04

%post

apt-get update

apt-get upgrade -y

Now we have an Ubuntu container, but it does not have tidyverse installed. To install it, we can add apt install r-cran-tidyverse -y in the %post section, again using -y to bypass the verification prompt.

Bootstrap: docker

From: ubuntu:24.04

%post

apt-get update

apt-get install r-cran-tidyverse -y

We encounter an issue with a package asking us to choose our time zone, breaking the build. To bypass the interactive prompt, we can set DEBIAN\_FRONTEND=noninteractive before installing r-cran-tidyverse.

Bootstrap: docker

From: ubuntu:24.04

%post

apt-get update

DEBIAN\_FRONTEND=noninteractive apt-get install r-cran-tidyverse -y

After building the container, we should verify R is working by shelling into the container and using some R scripts to ensure tidyverse is functioning correctly.

apptainer shell container.sif

### Using Apptainer sandbox:

Although sandboxing is not recommended due to potential issues with imported environmental variables and reproducibility, it allows us to create a base container that we can shell into and configure, preserving the changes we make.

The .def (tidyverse.def):

Bootstrap: docker

From: ubuntu:24.04

We now can build the basic ubuntu container as a sandbox:

apptainer build --sandbox container\_sandbox definition.def

We will include -w (writable) and -f (fakeroot) when shelling into it so we can alter the files:

apptainer shell -wf container

Within the shell, we can now run our commands to install r-cran-tidyverse:

apt-get update -y

apt-get install r-cran-tidyverse -y

Once those have finished, we can exit the container with ‘exit’ and build the sandbox into a smaller container.

apptainer build container.sif container\_sandbox

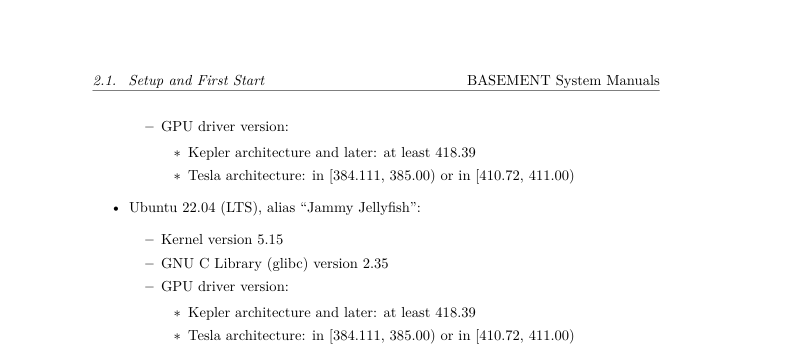
After building the container, verify its functionality by running an R script that utilizes tidyverse.

## From Binaries

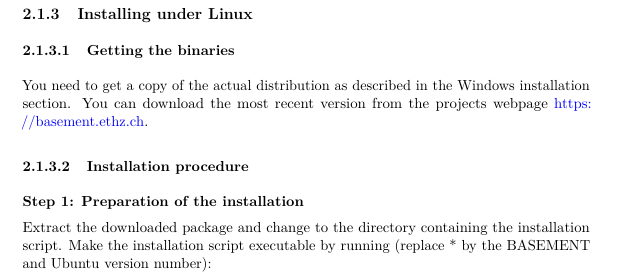
Installing software from binaries (install files) is similar to doing so on a regular system. In this example, we’ll install Basement from ETH Zürich in a container.

### Research:

Upon reviewing the manuals, we find that the software depends on a specific Ubuntu version. Therefore we’ll use an Ubuntu 22.04 container.

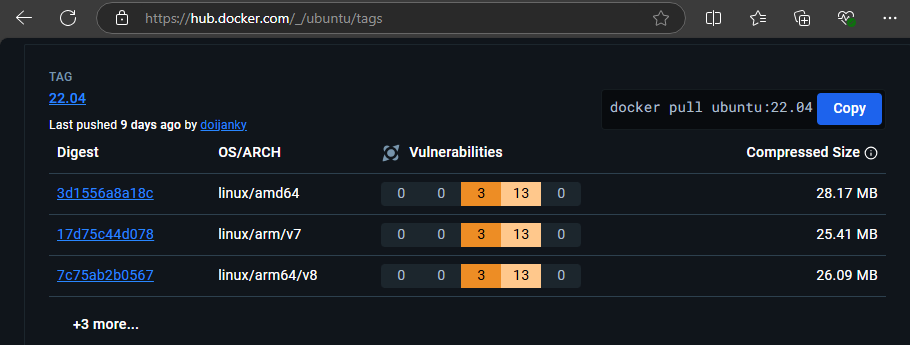


The software is downloadable from the website and will be a .tar.gz file.



### Creating an Apptainer .def file:

We can search through Ubuntu’s Docker Hub Tags page to find the appropriate Ubuntu image.



The base of our .def file should be:

Bootstrap: docker

From: ubuntu:22.04

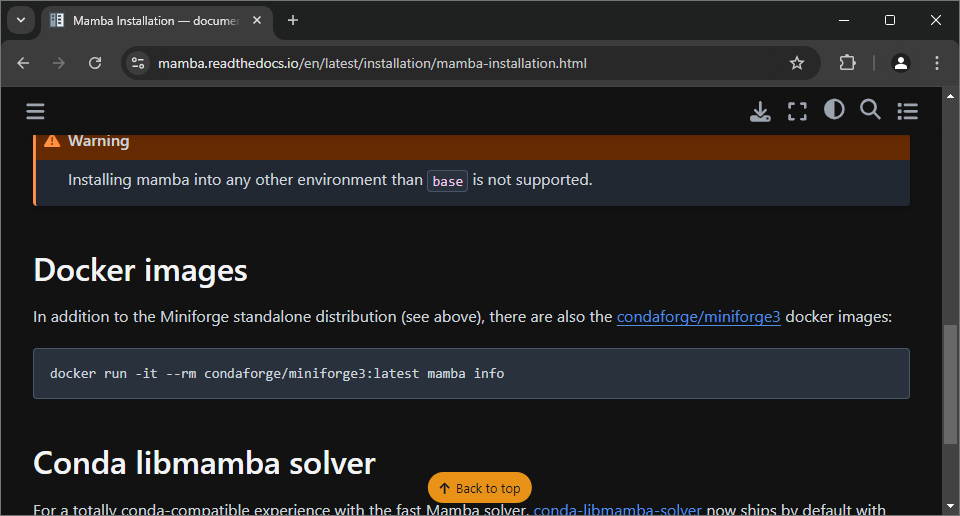
We can then either create a sandbox and manually run the install commands or copy the instructions into the .def file with some tweak to build the container and let Apptainer handle it for us.

## From Conda/Mamba (Not Recommended)

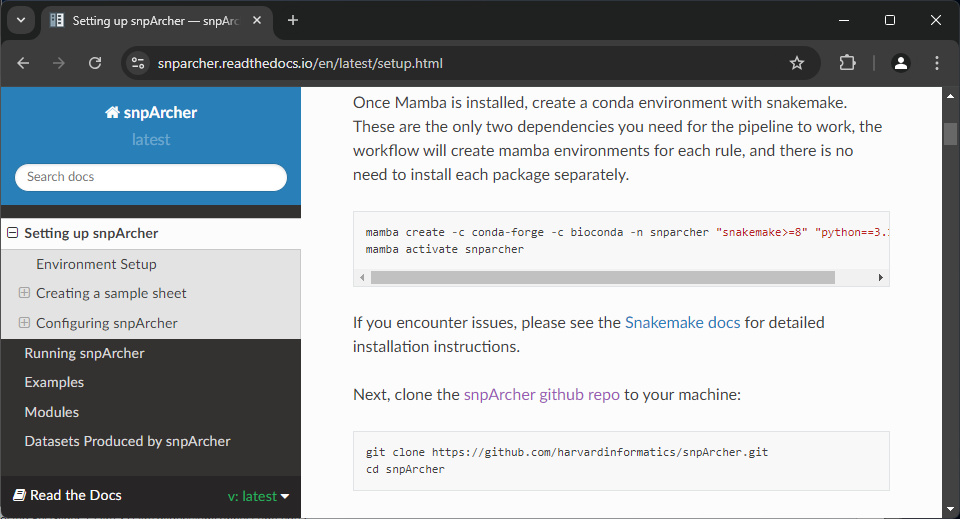
While Conda and Mamba serve similar purposes to Apptainer, they can encounter permission issues on HPC systems. To address this, we can create a container with Conda/Mamba installed, ensuring the necessary permissions for the program. In this example, we’ll create a snparcher environment inside the container.

### Research

First, we need to explore container options. According to Mamba’s documentation, the recommended installation method is through Miniforge, which conda-forge provides a Docker image (condaforge/miniforge3:latest).



Next, we look into installing snparcher, which is done using a Mamba command.



### Creating the .def File:

The base .def file should look like:

Bootstrap: docker

From: condaforge/miniforge3:latest

Following snparcher’s installation documentation, we add the command to the %post section:

Bootstrap: docker

From: condaforge/miniforge3:latest

%post

apt-get update

mamba create -c conda-forge -c bioconda -n snparcher "snakemake>=8" "python==3.11.4"

Upon shelling into the container, we find that Mamba is installed but not initialized, preventing environment activation.



We can initialize it, but upon exiting and returning, we need to initialize Mamba again.

After extensive searching and trial and error, it appears that by adding the initialization command to the .def file, the container will load the environment correctly and function as intended.

Bootstrap: docker

From: condaforge/miniforge3:24.7.1-0

%environment

source /opt/conda/etc/profile.d/conda.sh

conda activate snparcher

%post

apt-get update -y

mamba create -c conda-forge -c bioconda -n snparcher "snakemake>=8" "python==3.11.4"

echo "source /opt/conda/etc/profile.d/conda.sh" >> ~/.bashrc

echo "conda activate snparcher" >> ~/.bashrc

# Integrating Apptainer with Slurm

To run the necessary programs and gain more resources, we can use Slurm, either through sbatch or srun.

## Sbatch

By utilizing sbatch, we can create a reproducible resource environment for Apptainer. The main challenge with combining sbatch and Apptainer is that the user cannot interact with the program during execution. Therefore, it’s crucial to ensure the program can complete the workflow without any user input.

### Single Command:

If the workflow requires only a single command, Apptainer exec might be sufficient. This command creates the container, runs the specified command, and then terminates the container. For example, running an R script inside Apptainer would look like:

apptainer exec container.sif Rscript script.r

Using Sbatch, we can allocate more resources than normal. We can simply add the command to an Sbatch script:

#!/bin/bash

#SBATCH --job-name=cpu\_job

##SBATCH --mail-type=END,FAIL # Mail events (NONE, BEGIN, END, FAIL, ALL)

##SBATCH --mail-user= # Where to send mail

#SBATCH --partition='cpu(all)'

#SBATCH --nodes=1 # Number of nodes

#SBATCH --ntasks=20 # Number of simultaneous processes

#SBATCH --cpus-per-task=1 # Number of cores per process

#SBATCH --mem=100G # Memory (RAM)

#SBATCH --time=0-8

#SBATCH --output=out.txt

#SBATCH --error=err.txt

apptainer exec container.sif Rscript script.r

### Persisting Containers:

For workflows requiring the container to persist, such as running services. Apptainer instance allows the container to idle, note that instances will wait for each command to finish before the next command can be executed and must be terminated.

There will likely be issues with accessing services, as HPC systems are restricted.

Here we create an Apptainer instance:

apptainer instance start container.sif example\_instance

We can see what instances are running with:

apptainer instance list

We can then send commands to the instance:

apptainer exec instance://example\_instance echo 'Hello!'

apptainer exec instance://example\_instance echo 'Hello again!'

And so on. Once finished, we will terminate the instance:

apptainer instance stop example\_instance

Here is what a Sbatch script would look like using Apptainer instances:

#!/bin/bash

#SBATCH --job-name=cpu\_job

##SBATCH --mail-type=END,FAIL # Mail events (NONE, BEGIN, END, FAIL, ALL)

##SBATCH --mail-user= # Where to send mail

#SBATCH --partition='cpu(all)'

#SBATCH --nodes=1 # Number of nodes

#SBATCH --ntasks=20 # Number of simultaneous processes

#SBATCH --cpus-per-task=1 # Number of cores per process

#SBATCH --mem=100G # Memory (RAM)

#SBATCH --time=0-8

#SBATCH --output=out.txt

#SBATCH --error=err.txt

apptainer instance start container.sif example\_instance

apptainer exec instance://example\_instance echo 'Command 1'

apptainer exec instance://example\_instance echo 'Command 2'

apptainer instance stop example\_instance