Finding, Downloading, and Applying Software on CURC Resources

January 9, 2025

John Reiland

Research Computing
UNIVERSITY OF COLORADO BOULDER

Learning Objectives

- Learn about different methods to install and use software on Alpine

Session Overview

- The Module System (Lmod)
- Building Software on CURC Systems
- Simplifying Source Installations with Spack
- Virtual Environments with Anaconda
- Containerization with Apptainer
- Requesting Software Installations

The Module System

Lmod

In most cases, a supercomputer has far more software installed than the average user will ever use.

- Users may need different versions of the same software, which in general cannot be installed nor used in parallel on the same system.
- The requirements for one package may adversely affect another package or even be mutually exclusive.

The Module System

HPC centers manage this complexity with **environment module systems.**

CURC uses the Lmod system.

The Module System

Setting up for today’s session.

1. Log in to the CURC HPC system
2. Get on an Alpine compute node

```
$ module avail
$ acompile --help
$ acompile --time=2:00:00
$ module avail
```

Note: Login nodes do not have the full software stack!

The Module System

Live Demo: Loading and unloading software modules will dynamically change the software environment on the cluster.

```
$ module purge
$ module load intel/2022.1.2
$ module avail
$ module load impi
$ module avail
```

```
$ module purge
$ module load gcc
$ module avail
$ module load openmpi
$ module avail
```

The Module System

Live Demo: Loading and unloading software modules will set (and reset) important environment variables for you.

```
$ module purge
$ module load intel
$ module load hdf5
$ module display hdf5
$ env | grep HDF5
```

```
$ module purge
$ module load gcc
$ module load hdf5
$ module display hdf5
$ env | grep HDF5
```

The Module System

Live Demo: Useful Lmod commands

```
module spider                       # list all available modules
module avail                        # list modules available to you
module load <package/version>       # load a module into your env
module purge                        # unload all modules
module list                         # list currently loaded modules
module display <package>            # display module info/help
module spider <package>             # view info for all versions
module spider <package/version>     # view info for specific version
```

The Module System

Lmod

Points to note about CURC-managed modules:

- CURC does not update system modules; we do fresh installs of new versions and change the default when that is appropriate
- Sometimes when a module is outdated or problematic we will remove it from the software stack

Take home: pay attention to what software modules you are loading, as this may be important for reproducibility!

Building Software on CURC Systems

- Definitions
  - **Building-** a generic term describing the overall installation process that includes compiling
  - **Compiling**- the process of converting source code to an executable
  - **Linking**- the process of combining pieces of code and data into a single file that can be loaded into memory and executed
  - **Installing**- the process of preparing, configuring, and moving files to make software usable on a system

Building Software on CURC Systems

- There are numerous ways to install software on CURC systems
  - Grab pre-compiled binaries
  - Within virtual environments (using Conda, Miniconda, or Mamba)
  - Using containers (Apptainer)
  - From source code
  - Using a package manager for HPC systems (Spack)

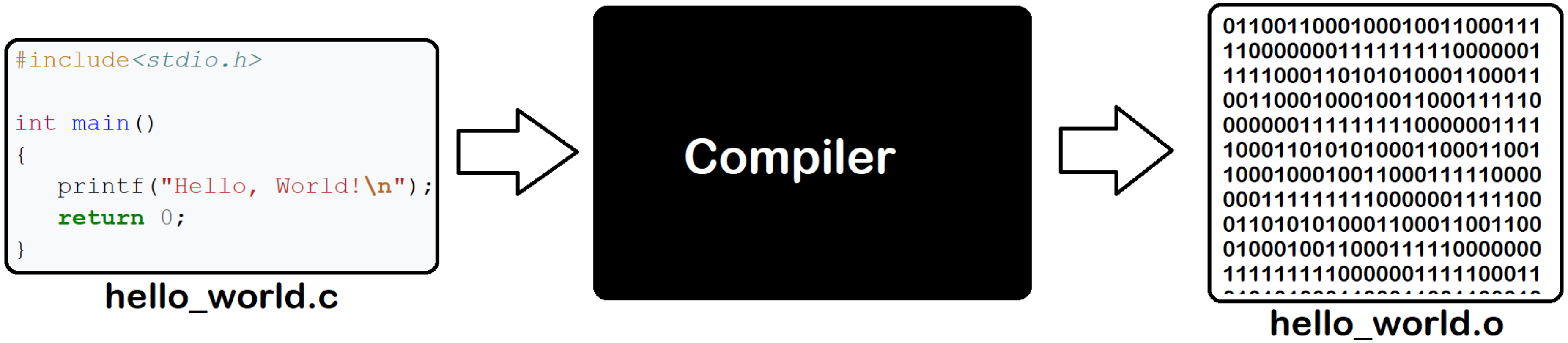
Building Software on CURC Systems

Why compile a research application manually from source code?

1. If it is not distributed as a pre-compiled binary, by any package manager, and is not easily containerized.
2. Compiling from source on the cluster will greatly improve performance.

Building Software on CURC Systems

Compilers are programs that convert code written in high level programming languages (like C/C++ or Fortran) to executable binary files.



```c
#include<stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

hello_world.c

```
0110011000100010011000111
1100000001111111110000001
1111000110101010001100011
0011000100010011000111110
0000001111111110000001111
1000110101010001100011001
1000100010011000111110000
0001111111110000001111100
0110101010001100011001100
0100010011000111110000000
1111111110000001111100011
```

hello_world.o

Building Software on CURC Systems

Build systems automate the process of compiling and linking.

1. GNU Build System
   - Used if your application includes instructions to run `./bootstrap`, `./autogen.sh`, `./configure` or `make` (the latter without a preceding `cmake`)
   - `make` is available in `/usr/bin`; Autotools available as a module

```
$ ./configure --prefix=/projects/$USER/software/bin
$ make
$ make install
```

Building Software on CURC Systems

Build systems automate the process of compiling and linking.

2. Cmake
    - Used if your application includes a `cmake` step
    - `module avail cmake`

```
$ cmake .. -DCMAKE_INSTALL_PREFIX=$INSTALLDIR \ --
      DCMAKE_CXX_COMPILER=g++ -DREGRESSIONTEST_DOWNLOAD=ON
$ make -j 8
$ make install
```

Building Software on CURC Systems

Conventions and best practices

- You will need to adapt installations for **local** or **user installations** (look for these terms in the software's docs)
- Don't install software in `/home/$USER` (too small) or scratch (purged every 90 days); `/projects/$USER/software` is the way to go!
- Keep your software installations organized by using a consistent file structure and naming convention
- Load the compiler first, MPI implementation second, and third-party libraries last

Building Software on CURC Systems

Conventions and best practices

- Don’t install executables to the source directory
  - Use `cmake -DCMAKE_INSTALL_PREFIX`, `./configure --prefix`
- The newest version of a compiler might not be compatible with your application. Read the package documentation and don’t be afraid to try different compilers and compiler versions
- Read our ‘Compiling and Linking’ documentation https://curc.readthedocs.io/en/latest/compute/compiling.html

Building Software on CURC Systems

Conventions and best practices

- Make life easier for yourself by adding executables to PATH and any directories with libraries that your application links to LD_LIBRARY_PATH

```
$ export PATH=/projects/$USER/software/phyloflash/bin:$PATH

$ echo $PATH

$ export
      LD_LIBRARY_PATH=/curc/sw/hdf5/1.10.1/impi/17.3/intel/17.4/lib
:$LD_LIBRARY_PATH

$ echo $LD_LIBRARY_PATH
```

Want to go the extra mile?

Try our Hands-on exercise #1 provided in EXERCISES.pdf.
This is not required for the micro-credential.

Objectives:

1) Explore CURC compilers and compiler environment variables.
2) Perform a simple source installation.

Estimated time to complete: 15 minutes

Simplifying Installations with Spack

How can we simplify source installations?

- **Package Managers** – Tools that automate installing, maintaining, and configuring software and any dependencies
- **Environments** – A collection of resources that are available in a self-contained 'bubble'

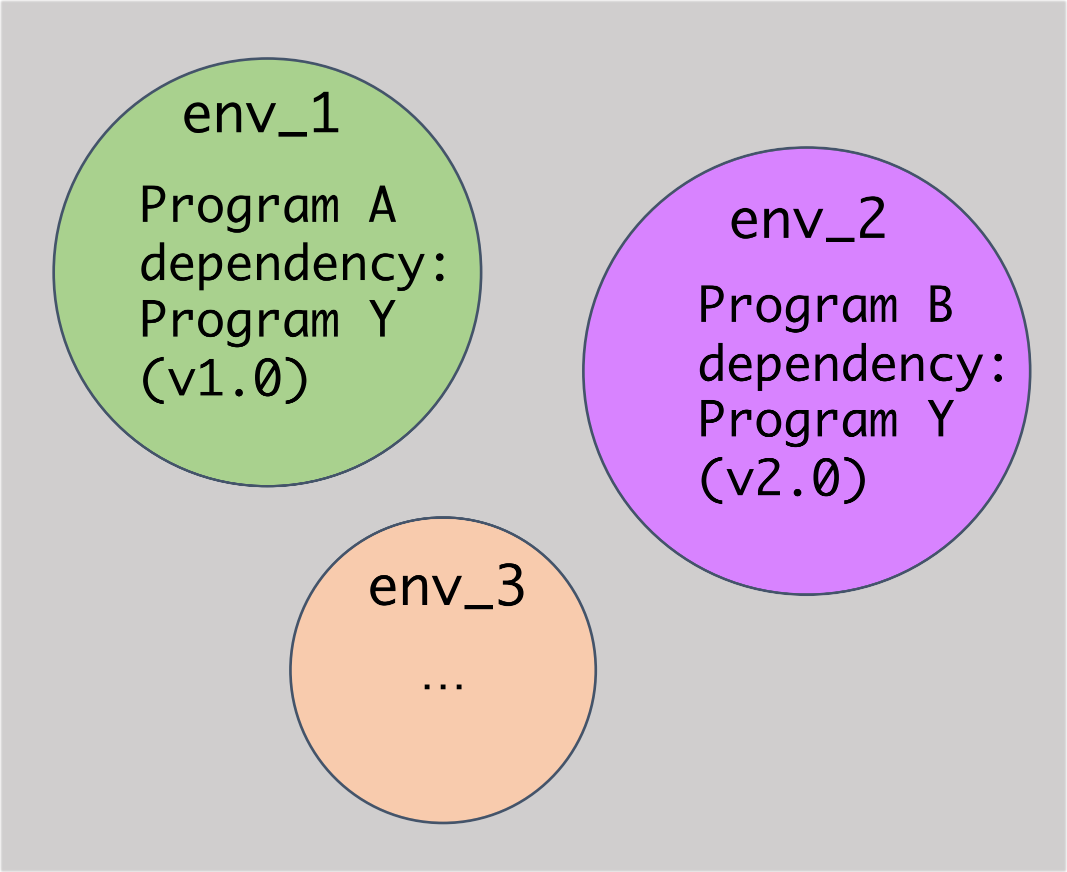
Simplifying Installations with Spack

Think of virtual environments as self-contained bubbles.

env_1 contains all the dependencies of ‘Program A’.

env_2 contains all the dependencies of ‘Program B’.

The environments do not interact.



env_1
Program A dependency: Program Y (v1.0)
env_2
Program B dependency: Program Y (v2.0)
env_3
…

Simplifying Installations with Spack

Your workflow requires two programs, ‘Program A’ and ‘Program B’.

- ‘Program A’ depends on ‘Program Y’ **v1.0**
- ‘Program B’ depends on ‘Program Y’ **v2.0**

What do you do?!

Simplifying Installations with Spack

Environments are created and programs are installed in a few simple steps

```
$ module purge
$ module load spack/0.20.1
$ spack env create my_first_env
$ spacktivate my_first_env
$ spack install --add samtools
```

Warning: Don’t install packages outside of an environment!

Simplifying Installations with Spack

- Packages are installed within **activated** environments

using `spack install`

```
$ spack install --add samtools      #install default samtools
$ spack install --add samtools@1.9  #install specific version
```

Simplifying Installations with Spack

- Spack installations can be slow but will progress more quickly with more cores.
    - Spack builds all packages in parallel. The default parallelism is equal to the number of cores available to the process, up to 16.

Simplifying Installations with Spack

- Useful spack commands

```
spack env list                      # list all your environments
spack remove <env>                  # remove an environment
spack uninstall <packagename>       # remove package
spack env status                    # check which env you’re in
spack info <packagename>            # prints detailed package info
spack find                          # show installed packages
despacktivate                       # deactivate environment
spack spec <packagename>            # list packages plan
```

Simplifying Installations with Spack

- Useful spack file paths

```
# root of the spack install tree
/projects/$USER/software/spack


# location of package executables - these are symbolically linked to
      the installation tree subdirectory
/projects/$USER/spack/environments/<env>/.spack-env/view/bin


# location of spack config file
/home/$USER/.spack/config.yaml
```

Want to go the extra mile?

Try our Hands-on exercise #2 provided in EXERCISES.pdf. This is not required for the micro-credential.

Objectives:
1) Create a Spack environment
2) Install `fastqc` in your Spack environment

Estimated time to complete: 20 minutes
Documentation:
https://curc.readthedocs.io/en/latest/software/spack.html

Virtual Environments with CONDA

Conda is a package (software) management system

- Installs, runs, and updates packages *and their dependencies*
- Creates, saves, loads, and switches between virtual environments
- Created for Python programs, but can package and distribute software for any language

Virtual Environments with CONDA

For our system, please be sure to add a .condarc file in your home directory.

- Prevents package installs from going to your home directory

```
$ cd ~
$ nano .condarc

Add the following content:

pkgs_dirs:
  - /projects/$USER/.conda_pkgs
envs_dirs:
  - /projects/$USER/software/anaconda/envs
```

Virtual Environments with CONDA

Environments are created and programs are installed in a few simple steps

```
$ module load anaconda
$ conda create -n my_first_env python==3.10
$ conda activate my_first_env
$ python
```

Warning: Don’t install packages in your base environment!

Virtual Environments with CONDA

Packages are installed within **activated** environments

using `conda install` (preferred method, when available)

```
$ conda install pandas            #install latest pandas
$ conda install pandas==0.20.3  #install specific version of pandas
```

Virtual Environments with CONDA

Packages are installed within **activated** environments

using `pip install` (if you must)

```
$ pip install --no-cache-dir pandas        #install latest pandas
```

Warning: `--no-cache-dir` is crucial on CURC systems!

Virtual Environments with CONDA

Useful conda commands

```
conda env list                     # list all environments
conda list                         # list packages in active env
conda env remove -n <envname>      # remove an environment
conda config --show channels       # view configured channels
conda deactivate                   # deactivate environment
conda create --name <clonedenv> /  # clone an environment
      --clone <envtoclone>
```

Virtual Environments with CONDA

Useful conda file paths

```
# location of python libraries
/projects/$USER/software/<env>/lib/python3.10/site-packages

# location of package executables
/projects/$USER/software/<env>/bin

# location of .condarc file
/home/$USER/.condarc
```

Want to go the extra mile?

Try our Hands-on exercise #3 provided in EXERCISES.pdf. This is not required for the micro-credential.

Objectives:

1) Configure your `.condarc` file
2) Create a conda environment and install samtools
3) Activate the environment and run samtools.

Estimated time to complete: 15 minutes

Documentation:

https://curc.readthedocs.io/en/latest/software/python.html

Containerization with APPTAINER

Containers are portable virtualizations of an operating system, software, libraries, data, and/or workflows

- Pros
  - Portability**-** containers can run on any system equipped with its specified container manager
  - Reproducibility- because containers are instances of prebuilt isolated software, the software will always execute the same every time
- Cons
  - Steeper learning curve than conda
  - Can be difficult to troubleshoot issues
  - Building containers can be tricky for multi-node MPI applications

Containerization with APPTAINER

CURC offers Apptainer (formerly Singularity) as container management software

- Apptainer comes pre-installed on all Alpine nodes, so no need to load any specific software modules!

Many common research applications have already been containerized and can be pulled from container repositories (such as Docker Hub).

Containerization with APPTAINER

Useful `Apptainer` commands:

```
apptainer exec          #Execute a command to your container
apptainer run           #Run your image as an executable
apptainer build         #Build a container
apptainer pull          #pull an image from hub
apptainer inspect       #See labels/environment vars, run scripts
apptainer shell         #Access the command line of your container
```

Containerization with APPTAINER

A container has its own file system and so needs help “seeing” files outside the container (on the host system). If not done in the `.def` file, this can be accomplished at runtime with bind mounting.

```
# bind mount a directory

apptainer run -B /source/directory:/target/directory sample-image.sif
```

On CURC systems, a running container automatically bind mounts these paths: `/home/$USER, $PWD.` *`Note that other locations will need to be manually mounted.`*

Want to go the extra mile?

- Try our Hands-on exercise #4 provided in EXERCISES.pdf. This is not required for the micro-credential.

Objectives:

1) Become familiar with basic Apptainer commands.

2) Pull an image from a pre-built container, then run the program from the container.

Estimated time to complete: 20 minutes

Documentation: https://curc.readthedocs.io/en/latest/software/Containerization.html

Requesting Software Installations

- Is the software already installed on the cluster? https://curc.readthedocs.io/en/latest/clusters/alpine/software.html
- Have you considered its utility and complexity?
  - Are you the only user of this software?
  - How complex or difficult is this software to install?
- Have you tried installing the package on your own?
- Software request form: https://www.colorado.edu/rc/userservices/software-request

Thank you!