Installing and Using Software on FASRC Cluster

Plamen Krastev, PhD
Harvard - Research Computing

Overview

- Software environment basics and software modules
- Installing `Python`, `R` and `Perl` packages
- Using precompiled (numeric, I/O, etc) software libraries
- Installing software yourself / `Spack`
- `Singularity` basics
- RC VDI portal (GUI apps, such as MATLAB, RStudio Server, Jupyter Notebook, Remote Desktop, and more)
- Upcoming OS upgrade (Rocky 8) and how this will affect SW support

Software on Cannon

- Operating System: CentOS 7
- 1,000+ scientific applications
  - https://portal.rc.fas.harvard.edu/p3/build-reports
  - Numeric Libraries: GSL, MKL, OpenBLAS, Eigen, Armadillo, etc.
  - Compilers: GCC, Intel, PGI
  - MPI: OpenMPI, Mvapich, Intel MPI
  - Scripting: Python, R, Perl, Ruby
  - Computing Environments: Matlab, Mathematica, Stata, SAS
  - GPU Computing: CUDA, OpenACC
  - Visualization: ParaView, VisIt
  - ML/DL: Tensorflow, Pytorch, MxNET
  - Performance Tools: TotalView, TAU Intel VTune, and others
- LMOD environment modules (HeLMOD, EasyBuild)
- VDI Portal for interactive computing (GUI apps)
- Containers: Singularity

Environment basics

When you login, Unix executes certain steps for your interactive sessions

- Startup ("dot") files are read
- Command prompts are set up
- Aliases expanded

Startup files set up default values for your environment

- `/etc/profile`
- `.bash_profile | .bash_login | .profile`
- `.bashrc`

The only things that really need to be in `.bash_profile` are

- environment variables and
- their exports and commands
- these aren't definitions but actually run or produce output when you log in

Option and alias definitions should go into the environment file `.bashrc`

Environment basics - .bash_profile

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

export PATH=$PATH:$HOME/bin
```

Environment basics - .bashrc

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions

# Settings
alias ls='ls --color=auto'

# RC Tools
export PATH=/n/sw/rc/bin:$PATH
```

LMOD Module System

LMOD: ENVIRONMENTAL MODULES SYSTEM

https://www.tacc.utexas.edu/research-development/tacc-projects/lmod

- Environment Modules provide a convenient way to dynamically modify the user's environment through module files (Lua-based scripting files)
- Software module files define various environment variables, such as `PATH`, `LD_LIBRARY_PATH`, `LIBRARY_PATH`, `CPATH`, `FPATH`, etc., so that executables, header files and required libraries can be found by the specific software application
- Software can be "loaded" and "unloaded" dynamically
- Hierarchies (incremental module loading/unloading) prevent software conflicts
- Various software versions can coexist
- Harvard modified LMOD software modules system: HeLMOD (Harvard Extensions for Lmod deployment) - https://github.com/fasrc/helmod
- EasyBuild (software build and installation framework) - https://easybuild.readthedocs.io/en/latest/

Software Modules

- About modules
- Software modules basics

```
module load gcc/9.3.0-fasrc01        # Load Compiler
module load openmpi/4.0.2-fasrc01    # Load MPI library
module load netcdf/4.7.3-fasrc05     # Load NetCDF Library

module list                          # List loaded modules
module purge                         # Unload all modules
module display netcdf/4.7.3-fasrc05 # Display module environment

module avail netcdf                  # Show available netcdf modules
```

- Finding modules
  - User portal - https://portal.rc.fas.harvard.edu/p3/build-reports/
  - Use the `module-query` command

```
module-query netcdf                 # Find available NetCDF modules
module-query netcdf/4.7.3-fasrc05 # List more information including how to load it
```

https://docs.rc.fas.harvard.edu/kb/software/

Modules: How do they work? (1)

```
[pkrastev@builds02 ~]$ module load gcc/9.3.0-fasrc01
[pkrastev@builds02 ~]$ which gcc
/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/bin/gcc
[pkrastev@holygpu2c0711 ~]$ gcc --version
gcc (GCC) 9.3.0
[pkrastev@builds02 ~]$ ls -l
/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01
total 2504
drwxr-xr-x 2 root root    614 Jun 11 17:43 bin
-rw-r--r-- 1 root root 625908 Mar 12 07:08 ChangeLog
-rw-r--r-- 1 root root  18002 Mar 12 07:07 COPYING
drwxr-xr-x 3 root root     21 Jun 11 17:41 include
drwxr-xr-x 2 root root    356 Jun 11 17:41 INSTALL
drwxr-xr-x 2 root root   1988 Jun 11 17:42 lib
drwxr-xr-x 3 root root   2489 Jun 11 17:42 lib64
drwxr-xr-x 3 root root     21 Jun 11 17:42 libexec
-rw-r--r-- 1 root root   2838 Jun 11 17:26 modulefile.lua
-rw-r--r-- 1 root root 926990 Mar 12 07:08 NEWS
-rw-r--r-- 1 root root   1026 Mar 12 07:07 README
drwxr-xr-x 6 root root     94 Jun 11 17:43 share
```

Modules: How do they work? (2)

```
[pkrastev@builds02 ~]$ module display gcc/9.3.0-fasrc01
…
/n/helmod/modulefiles/centos7/Core/gcc/9.3.0-fasrc01.lua:
…
whatis("Name: gcc")
whatis("Version: 9.3.0-fasrc01")
whatis("Description: the GNU Compiler Collection version 9.3.0")
setenv("CC","gcc")
setenv("CXX","g++")
setenv("FC","gfortran")
setenv("F77","gfortran")
setenv("GCC_HOME","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01")
prepend_path("PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/bin")
prepend_path("CPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/9.3.0/include")
prepend_path("CPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/9.3.0/install-tools/include")
prepend_path("CPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/9.3.0/plugin/include")
prepend_path("CPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/include")
prepend_path("FPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/9.3.0/include")
prepend_path("FPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/9.3.0/install-tools/include")
prepend_path("FPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/9.3.0/plugin/include")
prepend_path("FPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/include")
prepend_path("INFOPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/share/info")
prepend_path("LD_LIBRARY_PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib")
prepend_path("LIBRARY_PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib")
prepend_path("LD_LIBRARY_PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64")
prepend_path("LIBRARY_PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64")
prepend_path("MANPATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/share/man")
prepend_path("PKG_CONFIG_PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib/pkgconfig")
prepend_path("PKG_CONFIG_PATH","/n/helmod/apps/centos7/Core/gcc/9.3.0-fasrc01/lib64/pkgconfig")
prepend_path("MODULEPATH","/n/helmod/modulefiles/centos7/Comp/gcc/9.3.0-fasrc01")
setenv("FASRCSW_COMP_NAME","gcc")
setenv("FASRCSW_COMP_VERSION","9.3.0")
setenv("FASRCSW_COMP_RELEASE","fasrc01")
family("Comp")
```

Modules: Hierarchies (1)

Use of groupings is important for proper working of scientific applications.

- Applications built with specific compiler flavor/version need to be linked with libraries compiled with the same compiler flavor and version
- **Message Passing Interface (MPI)** library allows for communication between tasks on a distributed memory computers with many processors
- Parallel applications and libraries must be built with a matching MPI library and compiler

Instead of using a flat namespace, we can use module hierarchies:

- Simple technique because once users chose a compiler and MPI implementation, they can only load modules that match that compiler and MPI implementation
- FASRC follow's TACC's convention:

```
# MODULEPATH_ROOT=/n/helmod/modulefiles/centos7/
${MODULEPATH_ROOT}/{Core,Comp,MPI, CUDA}
```

Modules: Hierarchies (2)

```
[pkrastev@builds02 ~]$ module-query hdf5
------------------------------------------------------------------------------------------------
  HDF5
------------------------------------------------------------------------------------------------
    Description:
      HDF5 is a data model, library, and file format for storing and managing data. It supports an
unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and
complex data.
      HDF5 is portable and is extensible, allowing applications to evolve in their use of HDF5. The HDF5
Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data
in the
      HDF5 format.  HDF5 is used as a basis for many other file formats, including NetCDF.

    Versions:
      HeLmod CentOS 7
            hdf5/1.8.21-fasrc12.....................
            ...
            hdf5/1.6.7-fasrc01.....................
            hdf5/1.8.12-fasrc08..................... Added c++ bindings
            hdf5/1.8.12-fasrc09..................... Compiler-specific build
            ...
            hdf5/1.10.7-fasrc01..................... Core module for CentOS 7
            hdf5/1.10.7-fasrc02.....................
            hdf5/1.10.7-fasrc03.....................
            hdf5/1.12.1-fasrc01.....................

      Easy Build
            HDF5/1.10.1-intel-2017b.................
            HDF5/1.10.2-foss-2018b..................
            HDF5/1.10.2-intel-2017b.................

    To find detailed information about a module, search the full name.

      module-query HDF5/1.10.2-intel-2017b
…
```

Modules: Hierarchies (3)

```
[pkrastev@builds02 ~]$ module-query hdf5/1.12.1-fasrc01

------------------------------------------------------------------------------
  HDF5 : hdf5/1.12.1-fasrc01
------------------------------------------------------------------------------
    Build flavor: HeLmod CentOS 7
    Description:
      HDF5 is a data model, library, and file format for storing and managing data. It
supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O
and for high volume and complex data.
      HDF5 is portable and is extensible, allowing applications to evolve in their use of
HDF5. The HDF5 Technology suite includes tools and applications for managing, manipulating,
viewing, and analyzing data in the
      HDF5 format.  HDF5 is used as a basis for many other file formats, including NetCDF.

This module can be loaded as follows:
      module load gcc/10.2.0-fasrc01 openmpi/4.1.1-fasrc01 hdf5/1.12.1-fasrc01
      module load intel/21.2.0-fasrc01 openmpi/4.1.1-fasrc01 hdf5/1.12.1-fasrc01
```

Python Programs (1)

For Python we recommend:

- Load one of the available Python modules, e.g., `python/3.9.12-fasrc01`
- Use the Anaconda environment for customizing python packages
- Multiple custom environments can be set up for home or lab folders (e.g., development or production code). Check conda options for "non-standard" locations
- https://docs.rc.fas.harvard.edu/kb/python/

```
# Load module
module load python/3.9.12-fasrc01
# Create local python environment in ~/.conda/envs/ENV_NAME
conda create -n ENV_NAME --clone="$PYTHON_HOME"
# Use the new environment
source activate ENV_NAME
# Install a new package named MY_PACKAGE
conda install MY_PACKAGE
# If the package is not available with conda use pip
pip install MY_PACKAGE
# If you have problems updating a package first remove it
conda remove MY_PACKAGE
```

Python Programs (2)

For faster creation of `conda` environments and installing / uninstalling packages, we recommend using mamba in place of `conda`:

- Mamba is a tool to manage `conda` environments
- Mamba uses the same commands and configuration options as `conda`
- You can swap almost all commands between `conda` & `mamba`
- https://github.com/fasrc/User_Codes/blob/master/Languages/Python/Mamba.md

```
# Load the Mamba module
module load Mamba/4.14.0-0

# Create local conda environment in ~/.conda/envs/ENV_NAME
mamba create -n ENV_NAME PACKAGE_LIST

# Use the new environment
source activate ENV_NAME

# Install a new package named MY_PACKAGE
mamba install MY_PACKAGE
```

Python Programs (3)

For optimal performance it is recommended to use the `--prefix` option to create / relocate your conda environments to your LAB space, e.g.,

`/n/holylabs/LABS/<PI_LAB>/Lab`

or

`/n/holylabs/LABS/<PI_LAB>/Users/${USER}`

```
# Load the Mamba module
module load Mamba/4.14.0-0

# Create a conda environment in LAB space
mamba install -y --prefix=/n/holylabs/LABS/<PI_LAB>/Lab/sw/<ENV_NAME> PACKAGE_LIST

# Activate the conda environment
source activate /n/holylabs/LABS/<PI_LAB>/Lab/sw/<ENV_NAME>
```

R Programs (1)

- When loading R from the LMOD software module system, 100s of common packages have already been installed.
- Use the `R_LIBS_USER` environment variable to specify location of customized R package installations:
- https://docs.rc.fas.harvard.edu/kb/r-packages/

```
# Load R module, e.g.,
module load R/4.2.0-fasrc01

# Set R_LIBS_USER to your location for R packages, e.g.,
export R_LIBS_USER=$HOME/apps/R:$R_LIBS_USER

# Start R
R

# Inside R, install the desired package, e.g.,
> install.packages("Rcpp")
```

R Programs (2)

- Some R packages, such as `rgdal`, `rstan`, etc, may require specific compiling options.
- This would require unsetting the environment variable `R_LIBS_SITE` and set `R_LIBS_USER`.
- In addition, you may need to create configuration file in:

```
${HOME}/.R/Makevars
```

- You can find examples in:
  - https://github.com/fasrc/User_Codes/blob/master/Languages/R/rgdal.md
  - https://github.com/fasrc/User_Codes/blob/master/Languages/R/rstan.md

Perl Programs

```
# Load perl modules & perl-modules, e.g.,
module load perl/5.26.1-fasrc01
module load  perl-modules/5.26.1-fasrc01

# Create space for local perl packages, e.g.,
mkdir -p $HOME/apps/src
mkdir -p $HOME/apps/perl/lib/perl5

# Set up local perl environment (this can go to your .bashrc file)
export LOCALPERL="$HOME/apps/perl"
export PERL5LIB="$LOCALPERL:$LOCALPERL/lib/perl5:$PERL5LIB"
export PERL_MM_OPT="INSTALL_BASE=$LOCALPERL"
export PERL_MB_OPT="--install_base $LOCALPERL"
export PATH="$LOCALPERL/bin:$PATH"

# Download the package you need to install and unpack it, e.g.,
cd $HOME/apps/src
wget http://search.cpan.org/CPAN/authors/id/T/TW/TWYLIE/modules/FASTAParse-0.0.3.tar.gz
tar xzvf FASTAParse-0.0.3.tar.gz

# Makefile.PL build
cd FASTAParse-0.0.3
perl Makefile.PL
make
make install
```

https://docs.rc.fas.harvard.edu/kb/perl/

Using Software Libraries (1)

- Software libraries allow you to use precompiled functions and routines in your applications.
- Many are already installed on the cluster, e.g., `GSL`, `BLAS`, `LAPACK`, `NetCDF`, `HDF5`, `FFTW`, `MKL`, `BOOST`, etc., and are available as software modules.
- Software libraries could also be a part of the OS and are typically located in `/lib` and `/lib64`
- Linking to specific libraries can be done by setting `-l` and `-L` flags, e.g.,

```
# Load required software modules, e.g.,
module load gsl/2.7-fasrc01

# Compile and link the application, e.g.,
gcc -o gsl_int_test.x gsl_int_test.c -O2 -lm -lgsl -lgslcblas
```

https://github.com/fasrc/User_Codes/tree/master/Libraries

Using Software Libraries (2)

```
#=============================================================
# Make file for gsl_int_test.c
#=============================================================
CFLAGS   = -c -O2
COMPILER = gcc
PRO      = gsl_int_test
OBJECTS  = gsl_int_test.o

LINK_GSL = -lm -lgsl -lgslcblas

${PRO}.x : $(OBJECTS)
        $(COMPILER) -o ${PRO}.x $(OBJECTS) $(LINK_GSL)

%.o : %.c
        $(COMPILER) $(CFLAGS) $(<F)

clean :
        rm -rf *.o *.x *.mod
```

https://github.com/fasrc/User_Codes/tree/master/Libraries

Singularity

Singularity provides a container runtime and an ecosystem for managing images that is suitable for multi-tenant systems and HPC environments.

Important aspects:

- No need to have elevated privileges at runtime, although root privileges are needed to build the images.
- Each application will have its own container
- Containers are not fully isolated (e.g., host network is available)
- Users have the same *uid* and *gid* when running an application
- Containers can be executed from local image files, or pulling images from a `docker` registry, a singularity hub or from `sylab` libraries https://cloud.sylabs.io

For basic usage refer to:

- https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers
- https://docs.sylabs.io/guides/3.11/user-guide

Installing Software - ***Singularity*** (1)

Example - Running TensorFlow on a CPU node:

```
# --- Start an interactive session ---
[login-node  ]$ salloc -p test --mem=4G -N 1 -t 60
# --- cd to your SCRATCH folder ---
[compute-node]$ cd $SCRATCH/your_lab/your_user/
# --- Pull the latest TF version from the Docker registry ---
[compute-node]$ singularity pull --name tf27_cpu.simg   \
> docker://tensorflow/tensorflow:latest
# --- Launch Python and print the TF version ---
[compute-node]$ singularity exec tf211_cpu.simg python
… (omitted output)
>>> import tensorflow as tf
>>> print(tf.__version__)
2.11.0
# --- Get examples from keras.io ---
[compute-node]$ git clone https://github.com/keras-team/keras-io.git
# --- Execute the code ---
[compute-node]$ singularity exec tf211_cpu.simg python  \
./keras-io/examples/vision/mnist_convnet.py
… (omitted output)
Test loss: 0.026334384456276894
Test accuracy: 0.9904999732971191
```

Installing Software - ***Singularity*** (2)

Running tensorflow on a GPU node:

```
# --- Start an interactive session on a partition with GPUs, e.g.,
[login-node  ]$ salloc -p gpu_test --gres=gpu:1 --mem=4G -N 1 -t 60
# --- cd to your SCRATCH folder ---
[compute-node]$ cd $SCRATCH/your_lab/your_user/
# --- Pull the latest TF GPU version from the Docker registry ---
[compute-node]$ singularity pull --name tf211_gpu.simg   \
> docker://tensorflow/tensorflow:latest-gpu
# --- Get examples from keras.io ---
[compute-node]$ git clone https://github.com/keras-team/keras-io.git
# --- Execute the code ---
[compute-node]$ singularity exec --nv tf211_gpu.simg python  \
./keras-io/examples/vision/mnist_convnet.py
… (omitted output)
Test loss: 0.02494816295802593 2
Test accuracy: 0.9915000200271606
```

Installing Software - ***Singularity*** (3)

Example: pulling images from repositories

Preparation (start an interactive session and cd to $SCRATCH directory):

```
[login-node  ]$ salloc -p gpu_test --gres=gpu:1 --mem=4G -N 1 -t 60
[compute-node]$ cd $SCRATCH/your_lab/your_user/
```

Pulling from Docker:

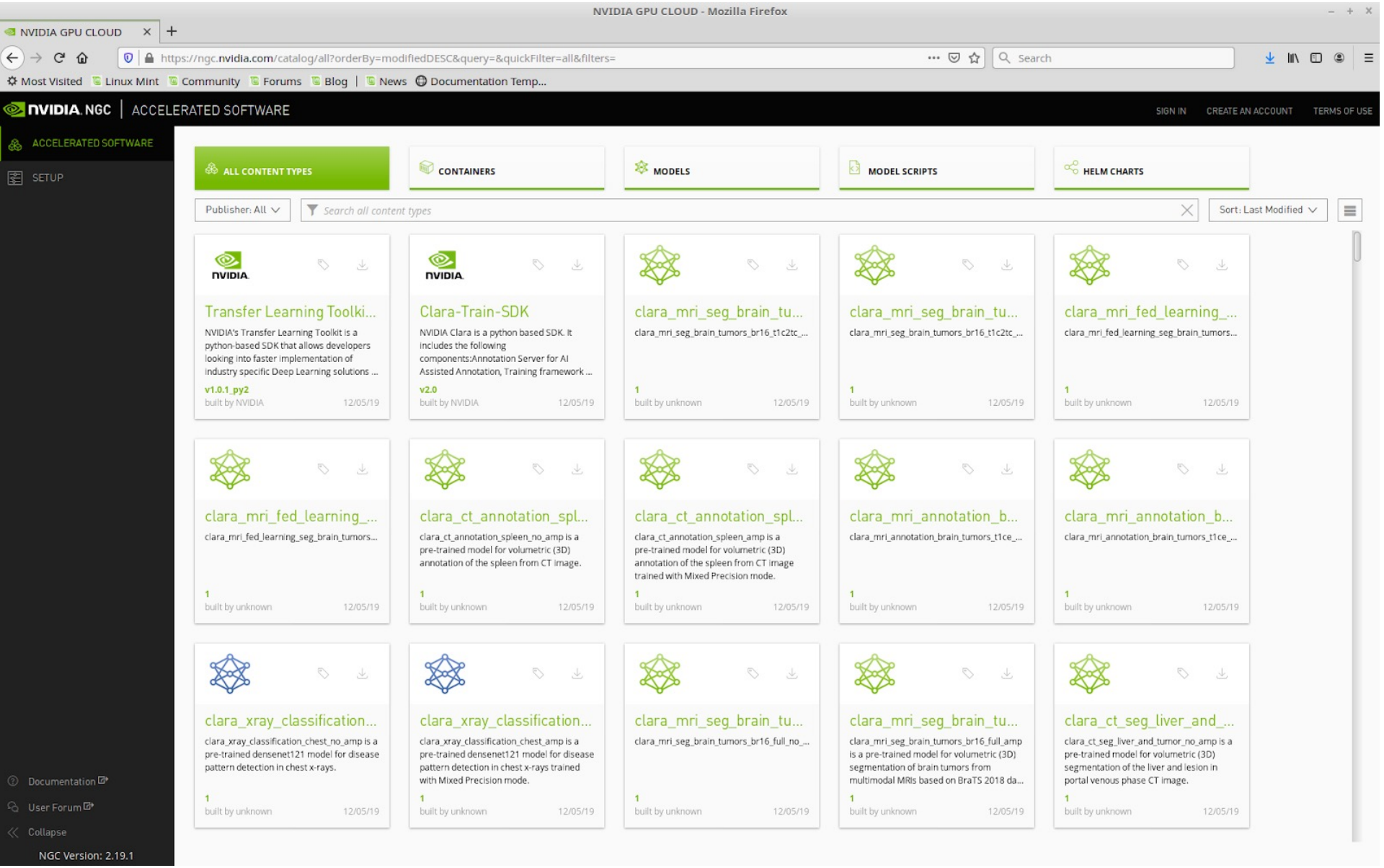
```
[compute-node]$ singularity pull docker://tensorflow/tensorflow:latest
```

Pulling from sylab / library -- https://cloud.sylabs.io/library

```
[compute-node]$ $ singularity pull library://library/default/ubuntu:21.04
```

Pulling from NVIDIA's NGC registry - https://catalog.ngc.nvidia.com

```
[compute-node]$ singularity pull docker://nvcr.io/nvidia/tensorflow:23.02-tf2-py3
[compute-node]$ singularity exec tensorflow_23.02-tf2-py3.sif python
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> print(tf.__version__)
2.11.0
```


NVIDIA GPU CLOUD - Mozilla Firefox
NVIDIA GPU CLOUD
https://ngc.nvidia.com/catalog/all?orderBy=modifiedDESC&query=&quickFilter=all&filters=
Search
Most Visited
Linux Mint
Community
Forums
Blog
News
Documentation Temp...
NVIDIA NGC | ACCELERATED SOFTWARE
SIGN IN
CREATE AN ACCOUNT
TERMS OF USE
ACCELERATED SOFTWARE
SETUP
ALL CONTENT TYPES
CONTAINERS
MODELS
MODEL SCRIPTS
HELM CHARTS
Publisher: All
Search all content types
Sort: Last Modified
Transfer Learning Toolki...
NVIDIA's Transfer Learning Toolkit is a python-based SDK that allows developers looking into faster implementation of industry specific Deep Learning solutions ...
v1.0.1_py2
built by NVIDIA
12/05/19
Clara-Train-SDK
NVIDIA Clara is a python based SDK. It includes the following components:Annotation Server for AI Assisted Annotation, Training framework ...
v2.0
built by NVIDIA
12/05/19
clara_mri_seg_brain_tu...
clara_mri_seg_brain_tumors_br16_t1c2tc_...
1
built by unknown
12/05/19
clara_mri_seg_brain_tu...
clara_mri_seg_brain_tumors_br16_t1c2tc_...
1
built by unknown
12/05/19
clara_mri_fed_learning_...
clara_mri_fed_learning_seg_brain_tumors...
1
built by unknown
12/05/19
clara_mri_fed_learning_...
clara_mri_fed_learning_seg_brain_tumors...
1
built by unknown
12/05/19
clara_ct_annotation_spl...
clara_ct_annotation_spleen_no_amp is a pre-trained model for volumetric (3D) annotation of the spleen from CT image.
1
built by unknown
12/05/19
clara_ct_annotation_spl...
clara_ct_annotation_spleen_amp is a pre-trained model for volumetric (3D) annotation of the spleen from CT image trained with Mixed Precision mode.
1
built by unknown
12/05/19
clara_mri_annotation_b...
clara_mri_annotation_brain_tumors_t1ce_...
1
built by unknown
12/05/19
clara_mri_annotation_b...
clara_mri_annotation_brain_tumors_t1ce_...
1
built by unknown
12/05/19
clara_xray_classification...
clara_xray_classification_chest_no_amp is a pre-trained densenet121 model for disease pattern detection in chest x-rays.
1
built by unknown
12/05/19
clara_xray_classification...
clara_xray_classification_chest_amp is a pre-trained densenet121 model for disease pattern detection in chest x-rays trained with Mixed Precision mode.
1
built by unknown
12/05/19
clara_mri_seg_brain_tu...
clara_mri_seg_brain_tumors_br16_full_no_...
1
built by unknown
12/05/19
clara_mri_seg_brain_tu...
clara_mri_seg_brain_tumors_br16_full_amp is a pre-trained model for volumetric (3D) segmentation of brain tumors from multimodal MRIs based on BraTS 2018 da...
1
built by unknown
12/05/19
clara_ct_seg_liver_and_...
clara_ct_seg_liver_and_tumor_no_amp is a pre-trained model for volumetric (3D) segmentation of the liver and lesion in portal venous phase CT image.
1
built by unknown
12/05/19
Documentation
User Forum
Collapse
NGC Version: 2.19.1

Installing Software - *Spack*

- ***Spack*** a package management tool designed to support multiple versions and configurations of software on a wide variety of platforms and environments.

- Designed for large supercomputer centers, where many users and application teams share common installations of software on clusters with exotic architectures, using non-standard libraries.

```
$ git clone -c feature.manyFiles=true https://github.com/spack/spack.git
$ cd spack/
$ git checkout releases/v0.19    # Check out the latest release
$ . share/spack/setup-env.sh     # Set up spack for use
$ spack list                     # List all available packages
$ spack list 'py-*'              # List available Python packages only
$ spack install gsl              # Install GSL
$ spack find                     # List installed packages
$ spack load gsl@2.7.1           # Load GSL version 2.7.1
$ spack unload gsl@2.7.1         # Unload GSL
```

Note: Moving forward, after the OS upgrade to *Rocky 8*, `spack` will be one of the primary methods for installing software in user and Lab directories.

https://github.com/fasrc/User_Codes/blob/master/Documents/Software/Spack.md

Installing Software - *configure/make*

Compiling and Installing Software:

```
# Compiler Choice, e.g.,
module load gcc/9.3.0-fasrc01

# Create space for source and local software, e.g.,
mkdir -p ~/sources
mkdir -p ~/software/gsl-2.7

# Get source code and unpack, e.g.,
cd ~/sources
wget ftp://ftp.gnu.org/gnu/gsl/gsl-2.7.tar.gz
tar xvfz gsl-2.7.tar.gz

# GNU-toolchain: configure, make, make install
cd gsl-2.7/
./configure --prefix=/n/home06/pkrastev/software/gsl-2.7
make -j4
make install
```

https://docs.rc.fas.harvard.edu/kb/installing-software-yourself/

Installing Software - *configure/make*

Making software available:

- We provide a convenience script `generate_setup.sh`, available with `fasrc` software module, to search the install directory and provide values for environment variables such as `PATH`, `LD_LIBRARY_PATH`, `CPATH`, etc., e.g.,

```
$ module load fasrc
$ generate_setup.sh --action echo ~/software/gsl-2.7
export PATH="/n/home06/pkrastev/software/gsl-2.7/bin:$PATH"
export LD_LIBRARY_PATH="/n/home06/pkrastev/software/gsl-2.7/lib:$LD_LIBRARY_PATH"
export LIBRARY_PATH="/n/home06/pkrastev/software/gsl-2.7/lib:$LIBRARY_PATH"
export PKG_CONFIG_PATH="/n/home06/pkrastev/software/gsl-2.7/lib/pkgconfig:$PKG_CONFIG_PATH"
export CPATH="/n/home06/pkrastev/software/gsl-2.7/include:$CPATH"
export FPATH="/n/home06/pkrastev/software/gsl-2.7/include:$FPATH"
export INFOPATH="/n/home06/pkrastev/software/gsl-2.7/share/info:$INFOPATH"
export MANPATH="/n/home06/pkrastev/software/gsl-2.7/share/man:$MANPATH"
```

- You can copy these variable definitions to the end of your .bashrc file for the software to become available by default when you log on

- Alternatively, you can copy these definitions to a setup.sh script, which you need to source, e.g.,

```
source ~/software/gsl-2.7/setup.sh
```

https://docs.rc.fas.harvard.edu/kb/installing-software-yourself/
https://github.com/jabrcx/miscnix/blob/master/bin/generate_setup.sh

Installing Software - *CMake*

Building software with CMake:

CMake is an open-source, cross-platform family of tools designed to build, test, and package software.

```
# Load a Cmake module, e.g.,
module load cmake/3.23.2-fasrc01

# Get source code and unpack, e.g.,
cd ~/sources
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.6/src/CMake-hdf5-1.10.6.tar.gz
tar xvfz CMake-hdf5-1.10.6.tar.gz

# Create install and build directories and configure, e.g.,
mkdir -p ~/software/hdf5-1.10.6
cd ~/sources/CMake-hdf5-1.10.6/hdf5-1.10.6
mkdir build && cd build
cmake -DCMAKE_INSTALL_PREFIX=/n/home06/pkrastev/software/hdf5-1.10.6 ..

# Build and install
make
make install
```

RC VDI Portal

- Interactive computing portal
  https://vdi.rc.fas.harvard.edu/pun/sys/dashboard
- Need to be on RC VPN
- Provides GUI apps, such as
  - Remote Desktop
  - Rstudio Server
  - Jupyter Notebook
  - MATLAB
  - COMSOL
  - and more

https://docs.rc.fas.harvard.edu/kb/vdi-apps/

OS Upgrade - *Rocky 8*

Centos 7 → Rocky 8

How will this affect software management on the FASRC cluster?

- Make available as modules **only** core packages: compilers (GNU/GCC, Intel, PGI), MPI (OpenMPI, Mpich, Intel MPI), I/O libraries (HDF5, NetCDF, etc.), numerical libraries (GSL, MKL, Eigen, Boost, PETSc, etc.), Python, R (core packages only), Perl, Julia, licensed software (MATLAB, Mathematica, IDL, etc.), simulation packages (CP2K, LAMMPS, GAMESS, GROMACS, etc.; base versions only)
- No longer supporting 6,000+ packages as modules
- Customized software stacks will be handled by `Spack` and `Singularity`
- Python - `conda` environments will be managed by `mamba`
- R - provide a base module; additional packages will be installed locally in User / Lab folders

Request Help - Resources

- Support
  https://docs.rc.fas.harvard.edu/kb/support
- Documentation
  https://docs.rc.fas.harvard.edu
- Portal
  https://portal.rc.fas.harvard.edu/rcrt/submit_ticket
- Email
  rchelp@rc.fas.harvard.edu
- Office Hours
  Wednesday noon-3pm - https://harvard.zoom.us/j/97676134704
- Code Examples and Use Cases
  https://github.com/fasrc/User_Codes
- Training
  https://www.rc.fas.harvard.edu/upcoming-training

Thank you! Questions? Comments?

Plamen Krastev, PhD
Harvard - Research Computing