DS 7347 HPC Wiki

Table of Contents

[Spack Tutorial 1](#_Toc109748069)

[SMU M2 Tutorials 1](#_Toc109748070)

[Docker Hub 4](#_Toc109748071)

[Spack Troubleshooting 6](#_Toc109748072)

# Spack Tutorial

<https://spack-tutorial.readthedocs.io/en/ecp21/>

<https://www.youtube.com/watch?v=RlczUgwFCJg>

# install Spack: first clone repo

**git clone https://github.com/spack/spack ~/.spack/Spack**

# To use Spack everywhere, add Spack to PATH

**. ~/.spack/Spack/share/spack/setup-env.sh**

# To use the spack command every login, append that command to ~/.bash\_profile  
**echo '. ~/.spack/Spack/share/spack/setup-env.sh' >> ~/.bash\_profile**  
# then run source ~/.bash\_profile

Spack System Requirements

<https://spack.readthedocs.io/en/latest/getting_started.html>

NOTE: make sure you **apt install** the respective packages here

* python3
* build-essential
* gcc
* git
* curl
* gfortran
* gzip
* unzip

**spack env create lab01 spack.yaml**

**spack env activate lab01**

**spack concretize --test all**

**spack install**

# SMU M2 Tutorials

Makefile Tutorial

<https://s2.smu.edu/hpc/documentation/development_tools/makefiles.html>

Version Control

<https://s2.smu.edu/hpc/documentation/development_tools/version_control_systems.html>

M2 Workshop Examples' GitHubs

<https://github.com/SouthernMethodistUniversity/m2_examples/tree/master/workshops>

* BLAS Benchmark
* OpenACC
* OpenNMP
* Slurm
* Vectorization

**Running Spack Container on M2**

1. First need to build your **spack.yaml** file

* requires **container** section
* **concretization** finds dependency tree that works for all specs

QUESTION: how do you know which images and version are available and their exact names?

**spack containerize --list-os**

**spack --version**

NOTE: CURRENT STATE DOES NOT SUPPORT **MATRIX**

spack:

specs:

- python@3.8.12

- py-pandas

- py-matplotlib

- py-numpy+blas+lapack

- openblas threads=openmp

concretization: together

container:

format: docker

images:

os: ubuntu:20.04

spack: v0.17.2

os\_packages:

final:

- libgfortran5

- libgomp1

extra\_instructions:

final:

RUN cd /opt/view/bin; find . -xtype l -exec sh -c 'ln -f -s $(basename $(readlink $0)) $(basename $0)' {​​​​​​​}​​​​​​ \;​​​​

2.  Use Spack to create **Dockerfile** from **spack.yaml**

**spack containerize > Dockerfile**

3. Build the Docker Image from spack's exported Dockerfile

**docker build -t lab02:py3 .**

**/!\WARNING**: This process can take 45+ minutes

TIP: try build Spack env locally with same **spack.yaml** file before running **docker build**

**spack install --fail-fast**

validate containers on your system

**docker container ls -a**

4. Run / Load image on M2

NOTE: test the docker image locally

**docker run -it --rm lab02:py3**

# the --rm removes container after it exits

OPTION 1: SCP .tar file

NOTE: this assumes you have a **lab02** directory on M2 in $HOME

**scp lab02.tar jotsap@m2.smu.edu:~/lab02/**

# then login to M2

**ssh jotsap@m2.scp.com**

# dont forget to load singularity

**module load singularity**  
**#** build the singularity file from the .tar file

**cd lab02/**

**singularity build -F lab02.sif docker-archive:~/lab02.tar**

# run the .sif file

**singularity exec ./lab02.sif /opt/view/bin/python3.8**

# to launch python 3 w/in the container

**singularity exec lab02.sif /opt/view/bin/python3**

OPTION 2: Use Docker Hub [see Docker Hub Tutorial below]

**ssh jotsap@m2.smu.edu 'bash -l -c "module load singularity\  
&& singularity run docker://jotsap/smulab:py3 -c "import numpy as np; print(np.pi)\""'**

# Docker Hub

1. make a docker repo

Goto <https://hub.docker.com/> and signup, then create a public repo

Once created you'll see the repo like so:

mine is **jotsap/smulab**

Graphical user interface, text, application, email, website

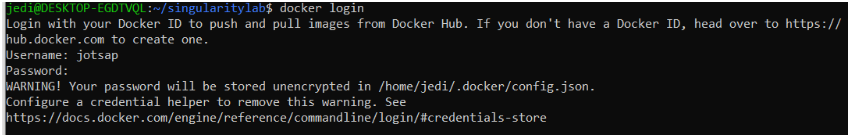
Description automatically generated

2. before you can push your docker image you need to login from the command prompt

**docker login**

It will prompt you for your username: jotsap is mine in this case  
password

then duo



3. Need to tag your image as what it will appear in the repo

**docker tag lab02:latest jotsap/smulab:lab02**

you can see containers by running

**docker containers ls -a**

Graphical user interface, text

Description automatically generated

4. Then push your image referencing the tag you just configured

**docker push jotsap/smulab:lab02**

Text

Description automatically generated

You can also see them now on your docker hub

Table

Description automatically generated

# Spack Troubleshooting

When trying to build your Spack environments, you may get the following error when trying to install modules, such as **openblas** or **py-numpy** into that environment:

Text

Description automatically generated

Notice towards the top, line 172 "need C or / and Fortran symbols." So what I found is that in the **compilers.yaml** file, for fortran portion, nothing was defined.

In your WSL prompt if you type **which gcc** and **which gfortran** it should point to /usr/bin/gcc or /usr/bin/gfortran respectively. IF NOT then you first need to run the respective **sudo apt install gcc** and **sudo apt install gfortran** commands. You also may need to update your .bashrc or .bash\_profile [whichever you use].

NOTE: you can try forcing Spack to find & enable compilers after the **apt install** but for me specifically, this did NOT update the **compilers.yaml** file, but certarinly worth a try:

**spack compilers** #to list active compilers

**spack compiler find**#to try and find newly installed compilers

**spack compiler add** #add them to the confiig

So first validate your **compilers.yaml** file:

cat .spack/linux/compilers.yaml

lets look at the output

compilers:

- compiler:

spec: gcc@9.4.0

paths:

cc: /usr/bin/gcc

cxx: /usr/bin/g++

f77: none

fc: none

flags: {​​​​​​​}​​​​​​​

operating\_system: ubuntu20.04

target: x86\_64

modules: []

environment: {​​​​​​​​​​​​​​​​​​​​​}​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​

extra\_rpaths: []

- compiler:

spec: gcc@10.3.0

paths:

cc: /usr/bin/gcc-10

cxx: null

f77: /usr/bin/gfortran

fc: /usr/bin/gfortran

flags: {​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​}​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​

operating\_system: ubuntu20.04

target: x86\_64

modules: []

environment: {​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​}​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​

extra\_rpaths: []

The **f77** and **fc** fields show **none**. What you need to do is open **compilers.yaml** using nano or vi, and change the **none** to the value shown by **which gfortran** which is presumably /usr/bin/gfortan

NOTE: if you want to be *super* safe, you can backup your **compilers.yaml** file:

**cp compilers.yaml compilers.old** #make a copy BEFORE you edit

compilers:

- compiler:

    spec: gcc@9.4.0

    paths:

      cc: /usr/bin/gcc

      cxx: /usr/bin/g++

      f77: /usr/bin/gfortran

      fc: /usr/bin/gfortran

    flags: {​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​}​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​

    operating\_system: ubuntu20.04

**mirrors.yaml**

<https://spack.readthedocs.io/en/latest/mirrors.html>

# GitHub Tutorial

<https://s2.smu.edu/hpc/workshops/2020/summer/git/>

<https://git-scm.com/docs/gittutorial>

<https://www.w3schools.com/git/>

Git Book

<https://git-scm.com/book/en/v2>

Git Branching 101

<https://www.atlassian.com/git/tutorials/using-branches>

[[YouTube Tutorial](https://youtu.be/S2TUommS3O0)]

NOTE: this assumes no conflicts; that is more advanced tutorial

[**Git Merge Conflicts**](https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts)

Most of the time, Git will figure out how to automatically integrate new changes. Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it. In these cases, Git cannot automatically determine what is correct. Conflicts only affect the developer conducting the merge, the rest of the team is unaware of the conflict. Git will mark the file as being conflicted and halt the merging process. It is then the developers' responsibility to resolve the conflict.

[**Git Merge Strategies**](https://www.atlassian.com/git/tutorials/using-branches/merge-strategy)

Git will take two (or more) commit pointers and attempt to find a common base commit between them. Git has several different methods to find a base commit, these methods are called "merge strategies". Once Git finds a common base commit it will create a new "merge commit" that combines the changes of the specified merge commits. Technically, a merge commit is a regular commit which just happens to have two parent commits.

The git merge and git pull commands can be passed an -s (strategy) option. The -s option can be appended with the name of the desired merge strategy. If not explicitly specified, Git will select the most appropriate merge strategy based on the provided branches.

**GIT BRANCHING WALK-THRU:**

This is a simple example where Jeremy is adding a **readme** file to *his* branch

git branch

# show current branch

git checkout -b **jeremy-branch**

# creates new branch **"jeremy-branch"** and moves us into it

### in this scenario we are adding a new readme file to **jeremy-branch**

git add readme.txt

# add new readme file ONLY to current branch "**jeremy-banch"**

git status

# validate

git commit -m "updated readme"

# commit in current branch NOT "master"

git status

# validate commit

git branch

# validate active branch

### now to merge

git checkout master

#switch to 'master' branch

git branch

#validate active branch

git merge **jeremy-branch**

# merge changes into 'master' branch

git status

# validate

git log

#see history

GitOps Overview

<https://www.atlassian.com/git/tutorials/gitops>

DevOps Overview

<https://www.atlassian.com/devops/what-is-devops>