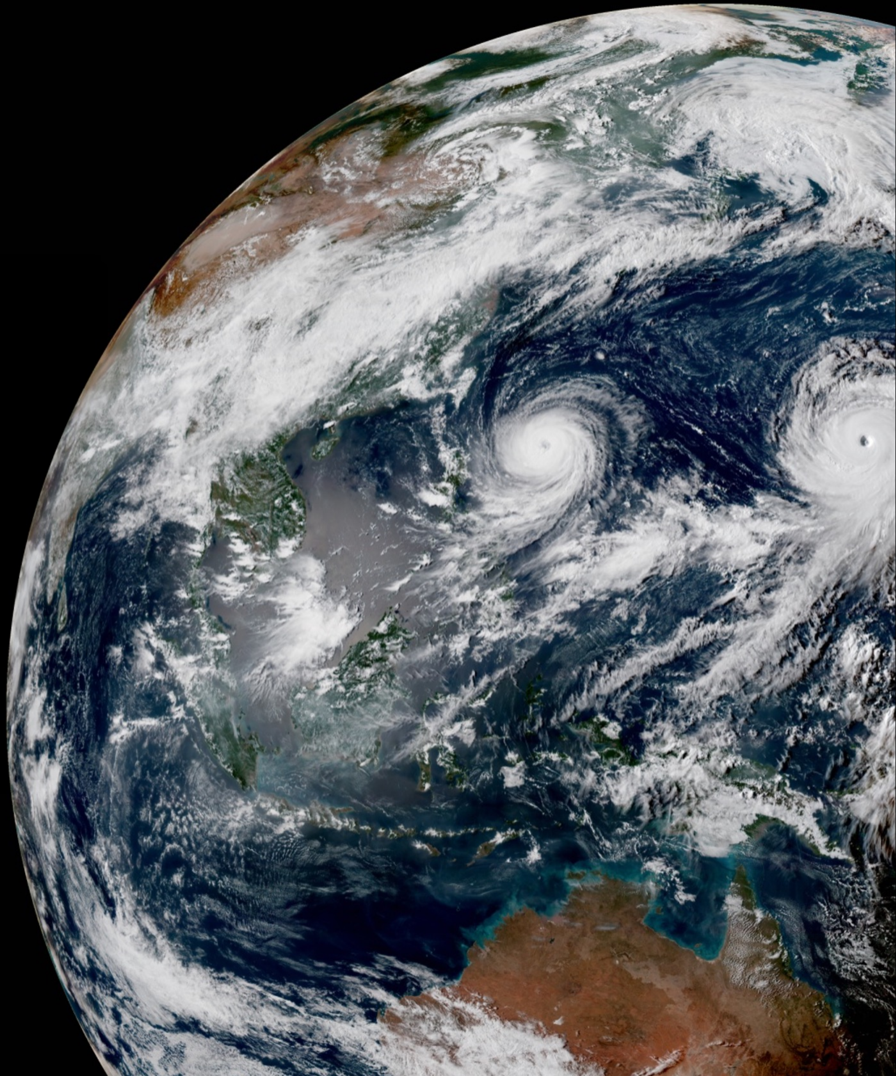
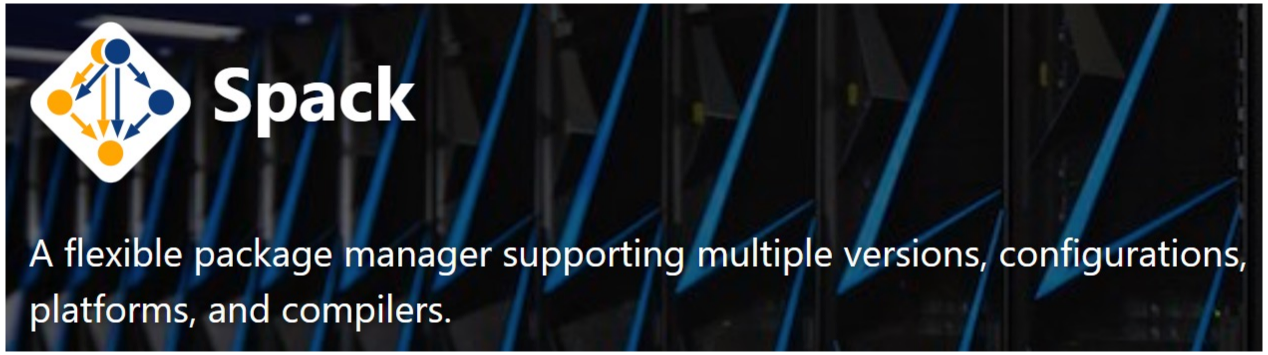
Spack-Stack: building JEDI bundles on your own machine

MPAS-JEDI Tutorial St. Andrews, UK June 2025
Christian Sampson

Contributors: Nate Crossette, Fabio Diniz



Spack



Spack

A flexible package manager supporting multiple versions, configurations, platforms, and compilers.

Spack is a community supported package manager for supercomputers, Linux and macOS developed by Lawrence Livermore National Lab. Its aim is to make installing scientific software easy

Custom versions & configurations

The installation of Spack can be customized in a variety of ways. Users can specify the package version, compiler, compile-time options, and even cross-compile platform, all from the command line.

Packages can peacefully coexist

Spack installs every unique package/dependency configuration into its own prefix, so new installs will not break existing ones.

Customize dependencies

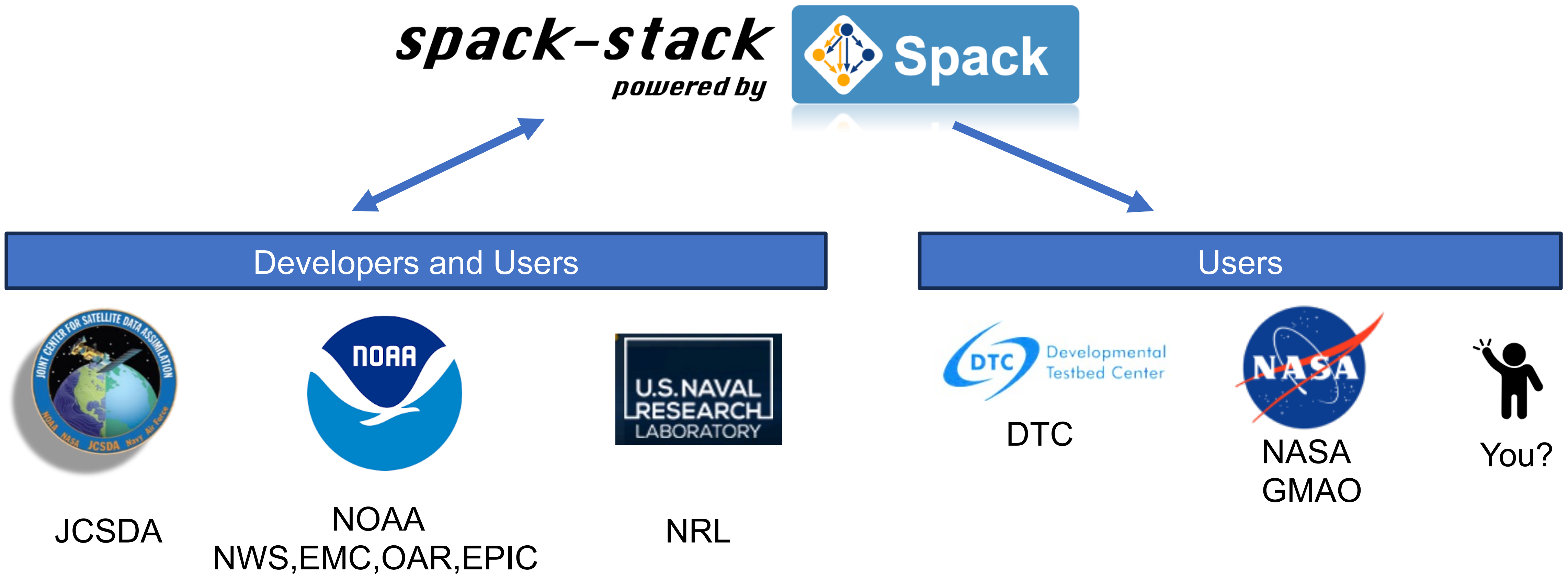
Spack allows dependencies of particular installations to be customized extensively. Suppose that `hdf5` depends on `openmpi` and indirectly on `hwloc`. Using `^`, users can add custom configurations for dependencies:

Creating packages is easy

Spack packages are simple Python scripts. The `spack create` command will generate boilerplate to get you started, and you can create a package in a matter of minutes. You write the build instructions; Spack builds the dependencies for you.

Spack-Stack

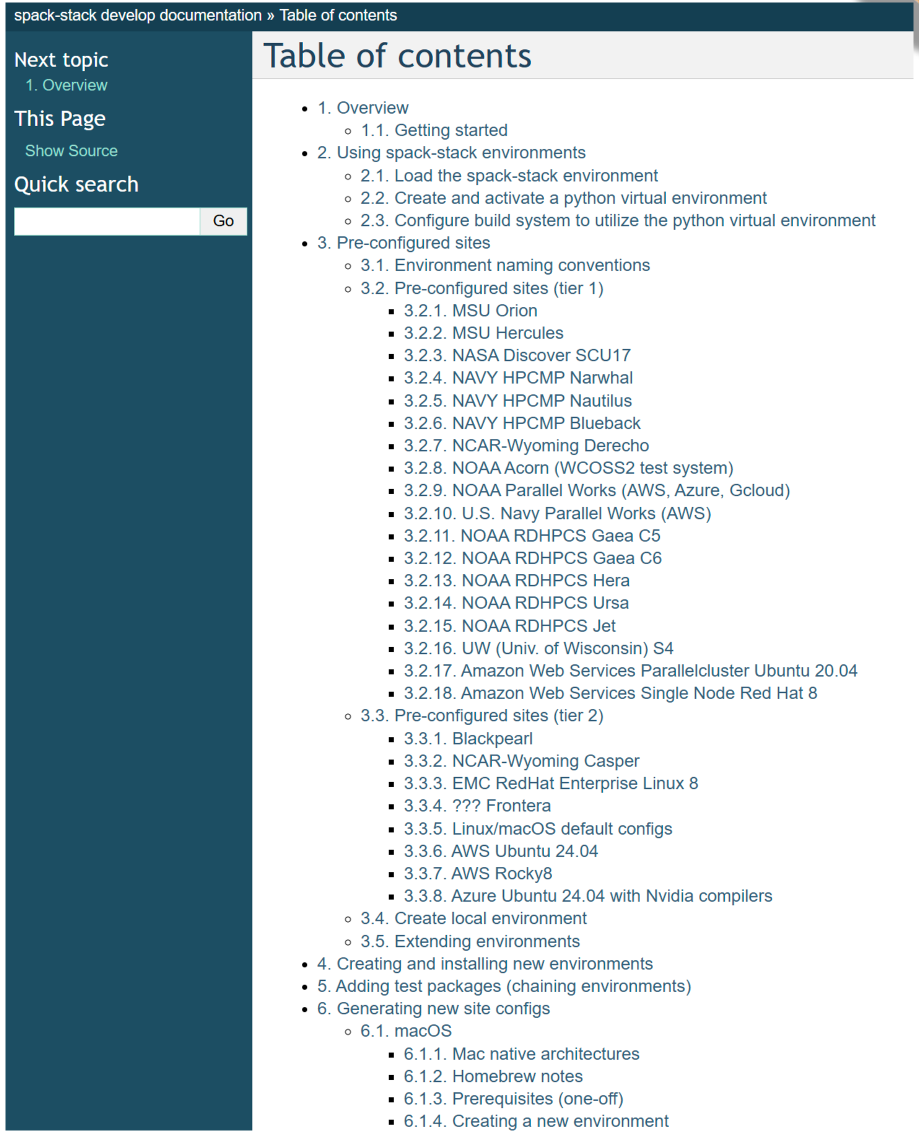
Spack-stack is a framework for installing software libraries to support NOAA's Unified Forecast System (UFS) applications and the Joint Effort for Data assimilation Integration (JEDI) coupled to several Earth system prediction models (MPAS, NEPTUNE, UM, FV3, GEOS, UFS).



Documentation

For the latest information make sure to check out the documentation.

https://spack-stack.readthedocs.io/en/latest/



spack-stack develop documentation » Table of contents

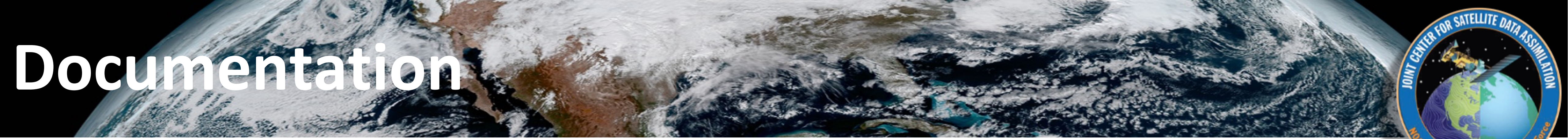
Next topic
1. Overview

This Page
Show Source

Quick search

Go

Table of contents

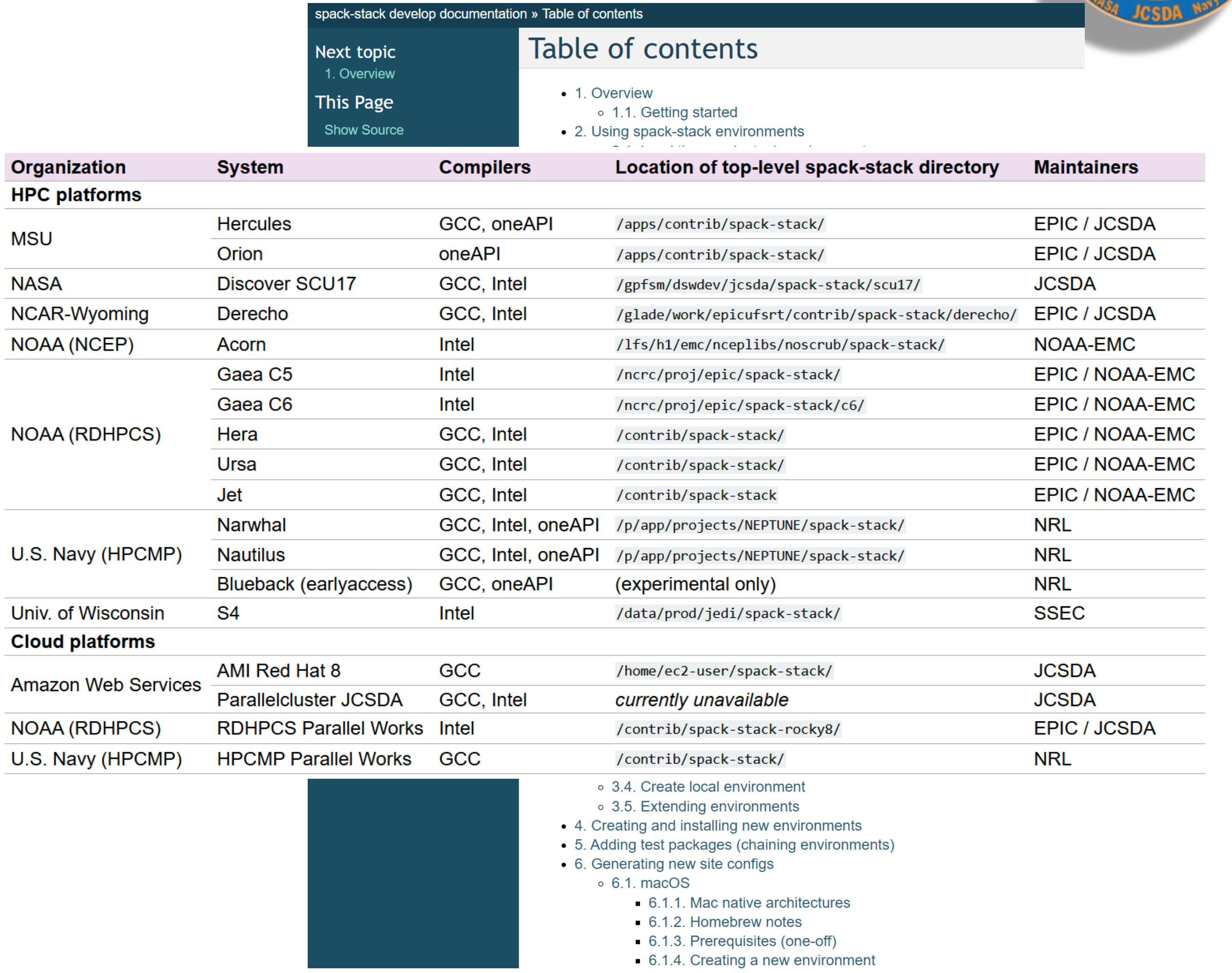


Documentation

For the latest information make sure to check out the documentation.

Here you can find out which HPC's are already supported with Spack-Stack amongst other things.

https://spack-stack.readthedocs.io/en/latest/



spack-stack develop documentation » Table of contents

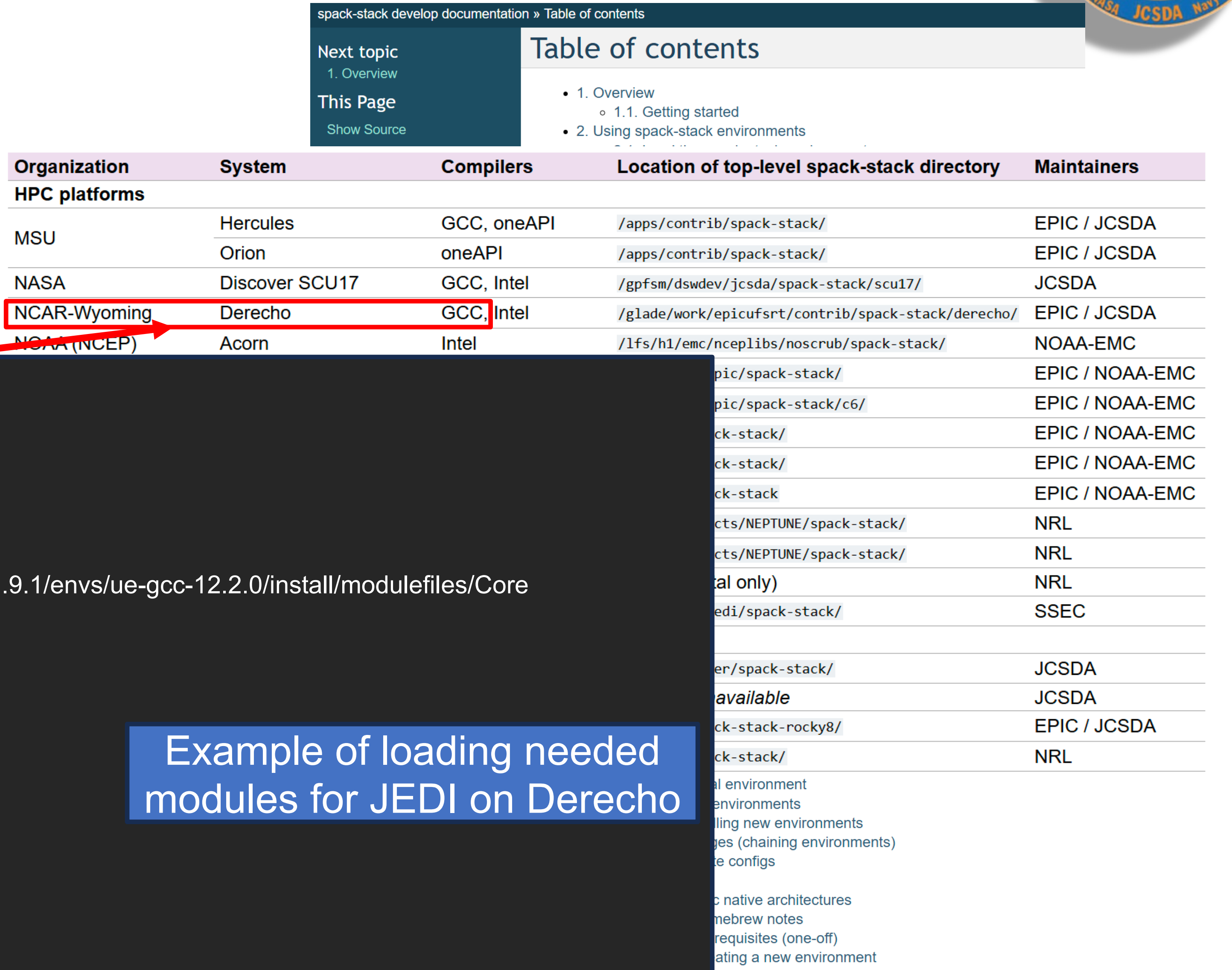
Table of contents

Organization	System	Compilers	Location of top-level spack-stack directory	Maintainers
HPC platforms				
MSU	Hercules	GCC, oneAPI	`/apps/contrib/spack-stack/`	EPIC / JCSDA
	Orion	oneAPI	`/apps/contrib/spack-stack/`	EPIC / JCSDA
NASA	Discover SCU17	GCC, Intel	`/gpfsm/dswdev/jcsda/spack-stack/scu17/`	JCSDA
NCAR-Wyoming	Derecho	GCC, Intel	`/glade/work/epicufsrt/contrib/spack-stack/derecho/`	EPIC / JCSDA
NOAA (NCEP)	Acorn	Intel	`/lfs/h1/emc/nceplibs/noscrub/spack-stack/`	NOAA-EMC
NOAA (RDHPCS)	Gaea C5	Intel	`/ncrc/proj/epic/spack-stack/`	EPIC / NOAA-EMC
	Gaea C6	Intel	`/ncrc/proj/epic/spack-stack/c6/`	EPIC / NOAA-EMC
	Hera	GCC, Intel	`/contrib/spack-stack/`	EPIC / NOAA-EMC
	Ursa	GCC, Intel	`/contrib/spack-stack/`	EPIC / NOAA-EMC
	Jet	GCC, Intel	`/contrib/spack-stack`	EPIC / NOAA-EMC
U.S. Navy (HPCMP)	Narwhal	GCC, Intel, oneAPI	`/p/app/projects/NEPTUNE/spack-stack/`	NRL
	Nautilus	GCC, Intel, oneAPI	`/p/app/projects/NEPTUNE/spack-stack/`	NRL
	Blueback (earlyaccess)	GCC, oneAPI	(experimental only)	NRL
Univ. of Wisconsin	S4	Intel	`/data/prod/jedi/spack-stack/`	SSEC
Cloud platforms				
Amazon Web Services	AMI Red Hat 8	GCC	`/home/ec2-user/spack-stack/`	JCSDA
	Parallelcluster JCSDA	GCC, Intel	*currently unavailable*	JCSDA
NOAA (RDHPCS)	RDHPCS Parallel Works	Intel	`/contrib/spack-stack-rocky8/`	EPIC / JCSDA
U.S. Navy (HPCMP)	HPCMP Parallel Works	GCC	`/contrib/spack-stack/`	NRL

Documentation (JEDI-bundle)

For the latest information make sure to check out the documentation.

Here you can find out which HPC's are already supported with Spack-Stack amongst other things.



spack-stack develop documentation » Table of contents

Next topic
1. Overview

This Page
Show Source

Table of contents

- 1. Overview
  - 1.1. Getting started
- 2. Using spack-stack environments

Organization	System	Compilers	Location of top-level spack-stack directory	Maintainers
HPC platforms				
MSU	Hercules	GCC, oneAPI	`/apps/contrib/spack-stack/`	EPIC / JCSDA
	Orion	oneAPI	`/apps/contrib/spack-stack/`	EPIC / JCSDA
NASA	Discover SCU17	GCC, Intel	`/gpfsm/dswdev/jcsda/spack-stack/scu17/`	JCSDA
NCAR-Wyoming	Derecho	GCC, Intel	`/glade/work/epicufsrt/contrib/spack-stack/derecho/`	EPIC / JCSDA
NOAA (NCEP)	Acorn	Intel	`/lfs/h1/emc/nceplibs/noscrub/spack-stack/`	NOAA-EMC
			`pic/spack-stack/`	EPIC / NOAA-EMC
			`pic/spack-stack/c6/`	EPIC / NOAA-EMC
			`ck-stack/`	EPIC / NOAA-EMC
			`ck-stack/`	EPIC / NOAA-EMC
			`ck-stack`	EPIC / NOAA-EMC
			`cts/NEPTUNE/spack-stack/`	NRL
			`cts/NEPTUNE/spack-stack/`	NRL
			al only)	NRL
			`edi/spack-stack/`	SSEC
			`er/spack-stack/`	JCSDA
			available	JCSDA
			`ck-stack-rocky8/`	EPIC / JCSDA
			`ck-stack/`	NRL

l environment
environments
ling new environments
ges (chaining environments)
te configs

c native architectures
mebrew notes
requisites (one-off)
ating a new environment

```
module purge
# ignore that the sticky module ncarenv/... is not unloaded
export LMOD_TMOD_FIND_FIRST=yes
module load ncarenv/23.09
module use /glade/work/epicufsrt/contrib/spack-stack/derecho/modulefiles

#load compilers etc
module use /glade/work/epicufsrt/contrib/spack-stack/derecho/spack-stack-1.9.1/envs/ue-gcc-12.2.0/install/modulefiles/Core
module load stack-gcc/12.2.0
module load stack-cray-mpich/8.1.27
module load stack-python/3.11.7

module load singularity

# See README.md
export LD_LIBRARY_PATH="${JEDI_BUILD}/lib:${LD_LIBRARY_PATH}“

# Load JEDI modules
module load jedi-fv3-env
module load jedi-mpas-env
module load soca-env
```

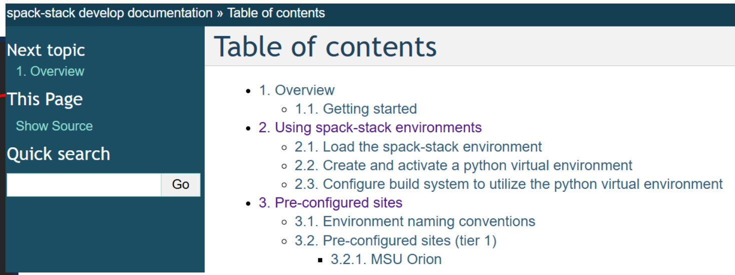
Example of loading needed modules for JEDI on Derecho

Documentation (JEDI-bundle)

Bits of info in a few places, but there are links.

```
module purge
# ignore that the sticky module ncarenv/... is not unloaded
export LMOD_TMOD_FIND_FIRST=yes
module load ncarenv/23.09
module use /glade/work/epicufsrt/contrib/spack-stack/derecho/modulefiles

#load compilers etc
module use /glade/work/epicufsrt/contrib/spack-stack/derecho/spack-stack-1.9.1/envs/ue-gcc-12.2.0/install/modulefiles/Core
module load stack-gcc/12.2.0
module load stack-cray-mpich/8.1.27
module load stack-python/3.11.7

module load singularity

# See README.md
export LD_LIBRARY_PATH="${JEDI_BUILD}/lib:${LD_LIBRARY_PATH}“

# Load JEDI modules
module load jedi-fv3-env
module load jedi-mpas-env
module load soca-env
```

https://spack-stack.readthedocs.io/en/latest/



spack-stack develop documentation » Table of contents

Next topic
1. Overview

This Page
Show Source

Quick search

Go

Table of contents

https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/using/jedi_environment/modules.html

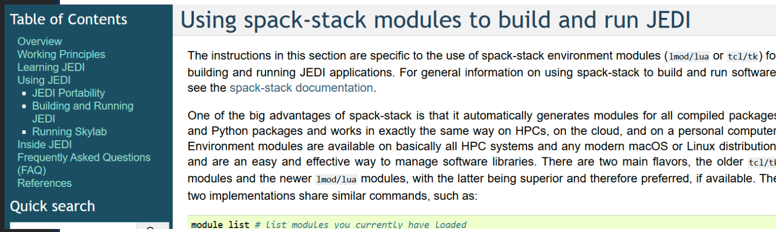


Table of Contents

Quick search

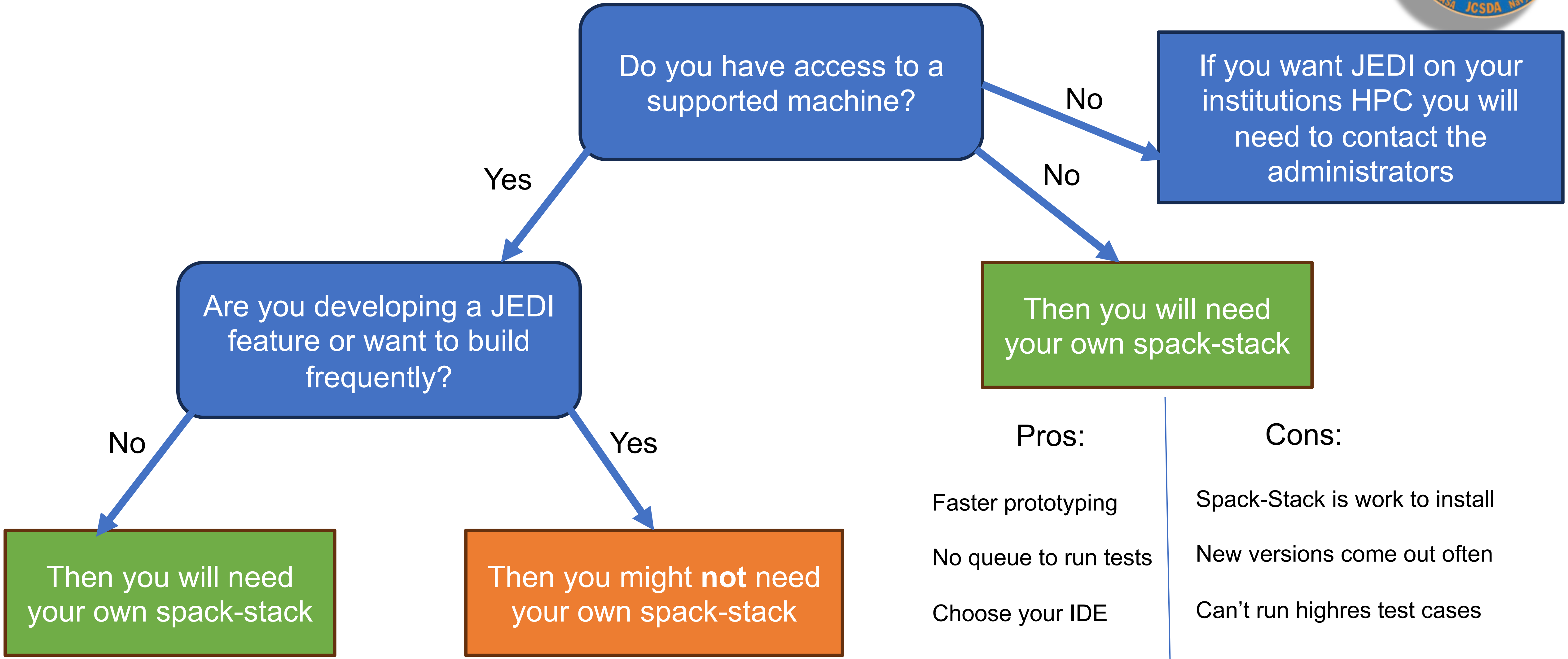
Using spack-stack modules to build and run JEDI

The instructions in this section are specific to the use of spack-stack environment modules (`lmod/lua` or `tcl/tk`) fo building and running JEDI applications. For general information on using spack-stack to build and run software see the spack-stack documentation.

One of the big advantages of spack-stack is that it automatically generates modules for all compiled package and Python packages and works in exactly the same way on HPCs, on the cloud, and on a personal computer Environment modules are available on basically all HPC systems and any modern macOS or Linux distribution and are an easy and effective way to manage software libraries. There are two main flavors, the older `tcl/t` modules and the newer `lmod/lua` modules, with the latter being superior and therefore preferred, if available. The two implementations share similar commands, such as:

```
module list # list modules you currently have loaded
```

Local Spack-Stack?


Do you have access to a supported machine?
No
If you want JEDI on your institutions HPC you will need to contact the administrators
Yes
No
Are you developing a JEDI feature or want to build frequently?
Then you will need your own spack-stack
No
Yes
Then you will need your own spack-stack
Then you might **not** need your own spack-stack

Pros:	Cons:
Faster prototyping	Spack-Stack is work to install
No queue to run tests	New versions come out often
Choose your IDE	Can't run highres test cases

So you want to install spack-stack…

Linux

- Can you install spack-stack natively? Yes!
- Any distro can work, but Ubuntu and RedHat/CentOS have more support and are tested by maintainers
- Linux is the best option for ease of install and mostly for speed when compiling JEDI

Windows

- Can install spack-stack natively? No!
- You can however install spack-stack in the Windows Subsystem for Linux (WSL) !
- WSL is a light weight VM maintained my Microsoft and meant to run in an integrated way in windows.
- You can choose the distros(Ubuntu, RedHat/CentOS) to use with WSL with others available as well!

macOS

- Can you install spack-stack natively? Yes, but..
- masOS updates often break your install and other difficulties can arise.
- The maintainers no longer support native macOS spack-stack installs
- However you can use OrbStack, it is like WSL for mac, and you can choose your distro as well.
- Mac users at JCSDA use OrbStack.

So you want to install spack-stack…

Linux

- Can you install spack-stack natively? Yes!
- Any distro can work, but Ubuntu and RedHat/CentOS have more support and are tested by maintainers
- Linux is the best option for ease of install and mostly for speed when compiling JEDI

Windows

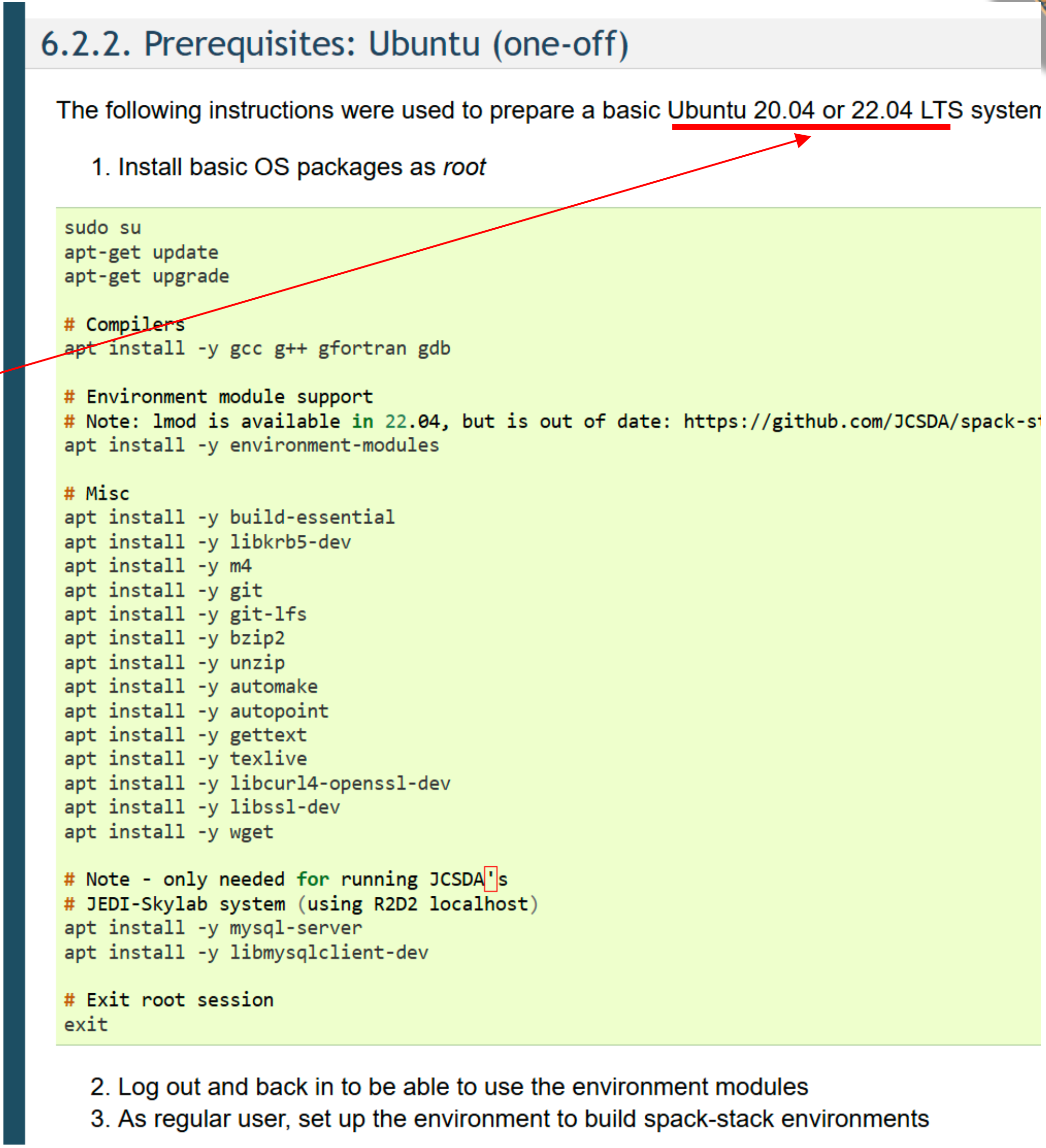
- Can install spack-stack natively? No!
- You can however install spack-stack in the Windows Subsystem for Linux (WSL) !
- WSL is a light weight VM maintained my Microsoft and meant to run in an integrated way in windows.
- You can choose the distros(Ubuntu, RedHat/CentOS) to use with WSL with others available as well!

macOS

- Can you install spack-stack natively? Yes, but..
- masOS updates often break your install and other difficulties can arise.
- The maintainers no longer support native macOS spack-stack installs
- However you can use OrbStack, it is like WSL for mac, and you can choose your distro as well.
- Mac users at JCSDA use OrbStack.

General Step 1: Prerequisites

- You will need to install some basic packages before installing spack-stack.
- In the documents there are examples for macOS, RedHat/centOS and Ubuntu.
- There will be distro version numbers listed there, those are ones that have been tested and are good choices to make the install smoother.
- You can of course choose what ever distro/version you like, but some ship with newer compilers or other packages that can cause issues when assembling the entire stack.
- Note, if you aim to install natively in macOS you will need to install Homebrew to then install the prerequisite packages.



6.2.2. Prerequisites: Ubuntu (one-off)

The following instructions were used to prepare a basic Ubuntu 20.04 or 22.04 LTS systen

1. Install basic OS packages as *root*

```
sudo su
apt-get update
apt-get upgrade

# Compilers
apt install -y gcc g++ gfortran gdb

# Environment module support
# Note: lmod is available in 22.04, but is out of date: https://github.com/JCSDA/spack-s
apt install -y environment-modules

# Misc
apt install -y build-essential
apt install -y libkrb5-dev
apt install -y m4
apt install -y git
apt install -y git-lfs
apt install -y bzip2
apt install -y unzip
apt install -y automake
apt install -y autopoint
apt install -y gettext
apt install -y texlive
apt install -y libcurl4-openssl-dev
apt install -y libssl-dev
apt install -y wget

# Note - only needed for running JCSDA's
# JEDI-Skylab system (using R2D2 localhost)
apt install -y mysql-server
apt install -y libmysqlclient-dev

# Exit root session
exit
```

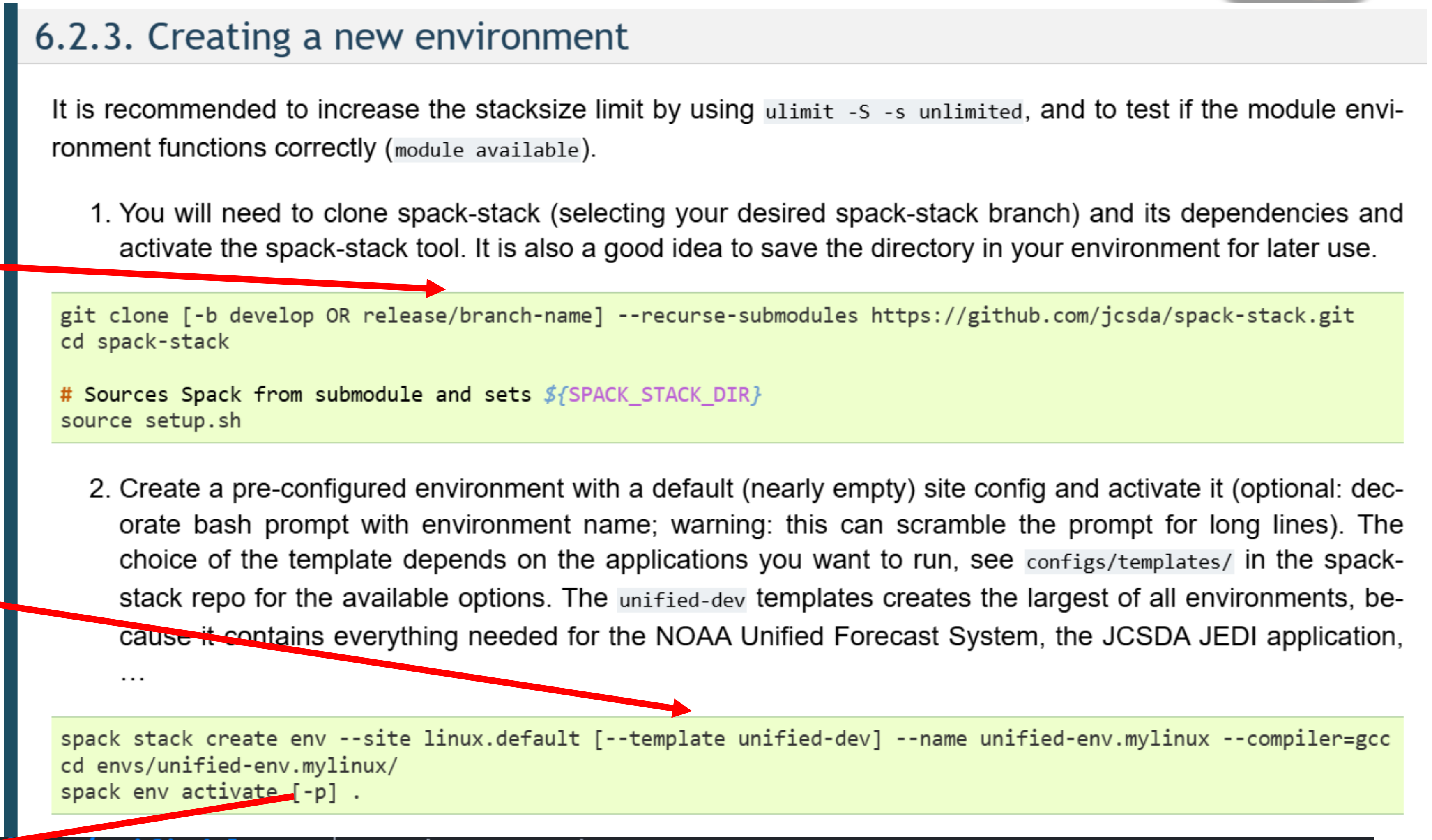
2. Log out and back in to be able to use the environment modules
3. As regular user, set up the environment to build spack-stack environments

General Step 2: Create a new environment

In the documentation when you see [ ] this indicates that you have choices to make.

Currently spack-stack 1.9.1
-b release/1.9.1

There are other templates, but unified-dev has everything you need.



6.2.3. Creating a new environment

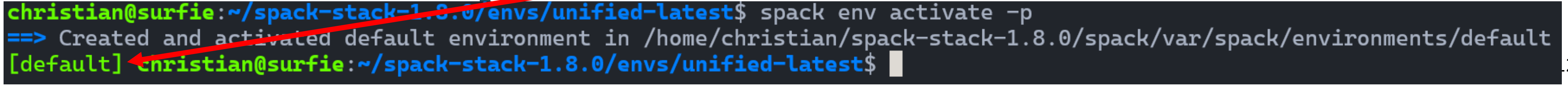
It is recommended to increase the stacksize limit by using `ulimit -S -s unlimited`, and to test if the module environment functions correctly (`module available`).

1. You will need to clone spack-stack (selecting your desired spack-stack branch) and its dependencies and activate the spack-stack tool. It is also a good idea to save the directory in your environment for later use.

```
git clone [-b develop OR release/branch-name] --recurse-submodules https://github.com/jcsda/spack-stack.git
cd spack-stack

# Sources Spack from submodule and sets ${SPACK_STACK_DIR}
source setup.sh
```

2. Create a pre-configured environment with a default (nearly empty) site config and activate it (optional: decorate bash prompt with environment name; warning: this can scramble the prompt for long lines). The choice of the template depends on the applications you want to run, see `configs/templates/` in the spack-stack repo for the available options. The `unified-dev` templates creates the largest of all environments, because it contains everything needed for the NOAA Unified Forecast System, the JCSDA JEDI application, …

```
spack stack create env --site linux.default [--template unified-dev] --name unified-env.mylinux --compiler=gcc
cd envs/unified-env.mylinux/
spack env activate [-p] .
```



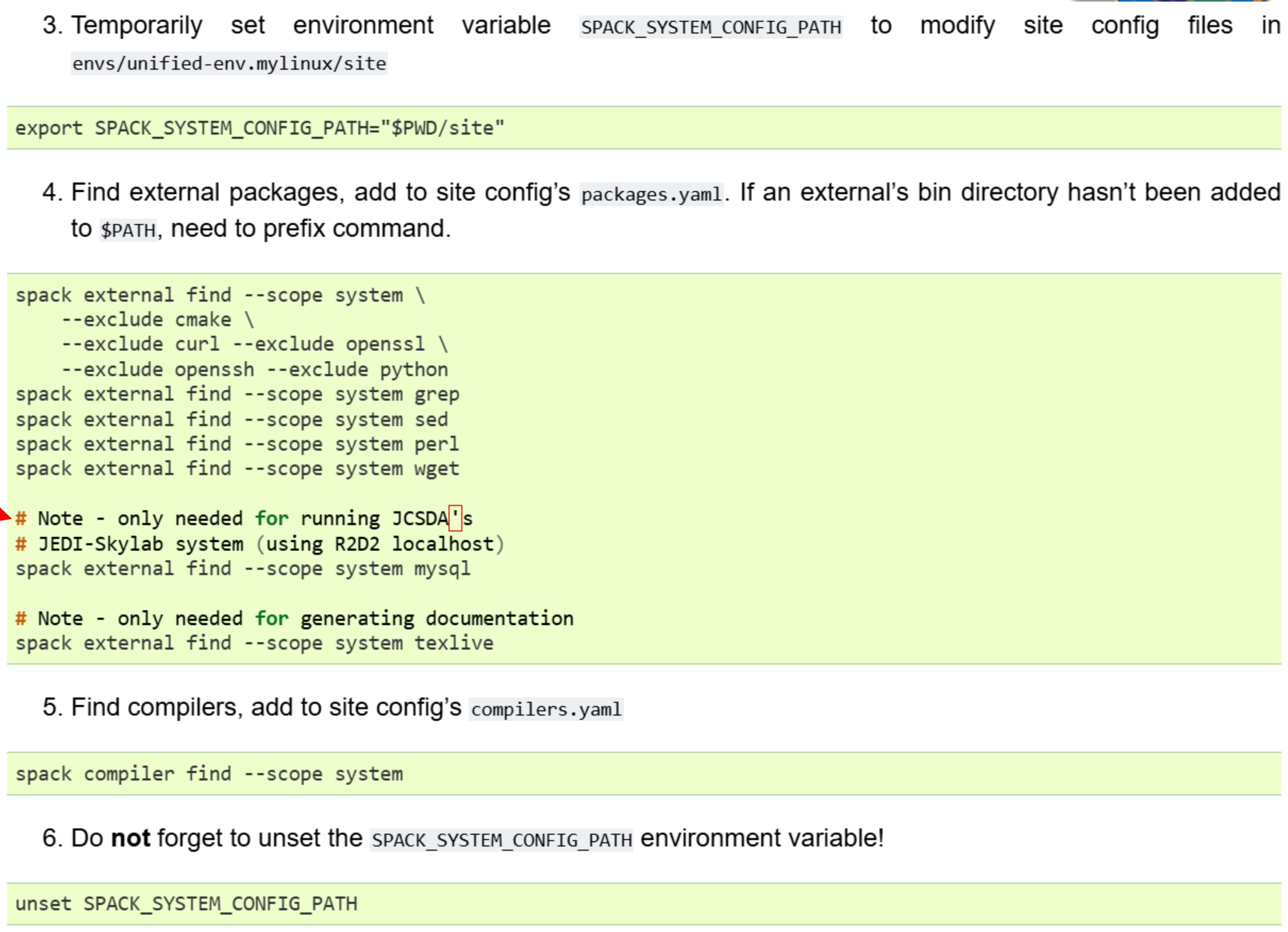
```
christian@surfie:~/spack-stack-1.8.0/envs/unified-latest$ spack env activate -p
==> Created and activated default environment in /home/christian/spack-stack-1.8.0/spack/var/spack/environments/default
[default] christian@surfie:~/spack-stack-1.8.0/envs/unified-latest$
```

General Step 3: Find your systems packages

Here we will let spack find all of our prerequisites we installed in general step 1.

Some things you might not need, just make sure to read the comments in the docs.

Don't Forget!



3. Temporarily set environment variable `SPACK_SYSTEM_CONFIG_PATH` to modify site config files in `envs/unified-env.mylinux/site`

```
export SPACK_SYSTEM_CONFIG_PATH="$PWD/site"
```

4. Find external packages, add to site config's `packages.yaml`. If an external's bin directory hasn't been added to `$PATH`, need to prefix command.

```
spack external find --scope system \
    --exclude cmake \
    --exclude curl --exclude openssl \
    --exclude openssh --exclude python
spack external find --scope system grep
spack external find --scope system sed
spack external find --scope system perl
spack external find --scope system wget

# Note - only needed for running JCSDA's
# JEDI-Skylab system (using R2D2 localhost)
spack external find --scope system mysql

# Note - only needed for generating documentation
spack external find --scope system texlive
```

5. Find compilers, add to site config's `compilers.yaml`

```
spack compiler find --scope system
```

6. Do **not** forget to unset the `SPACK_SYSTEM_CONFIG_PATH` environment variable!

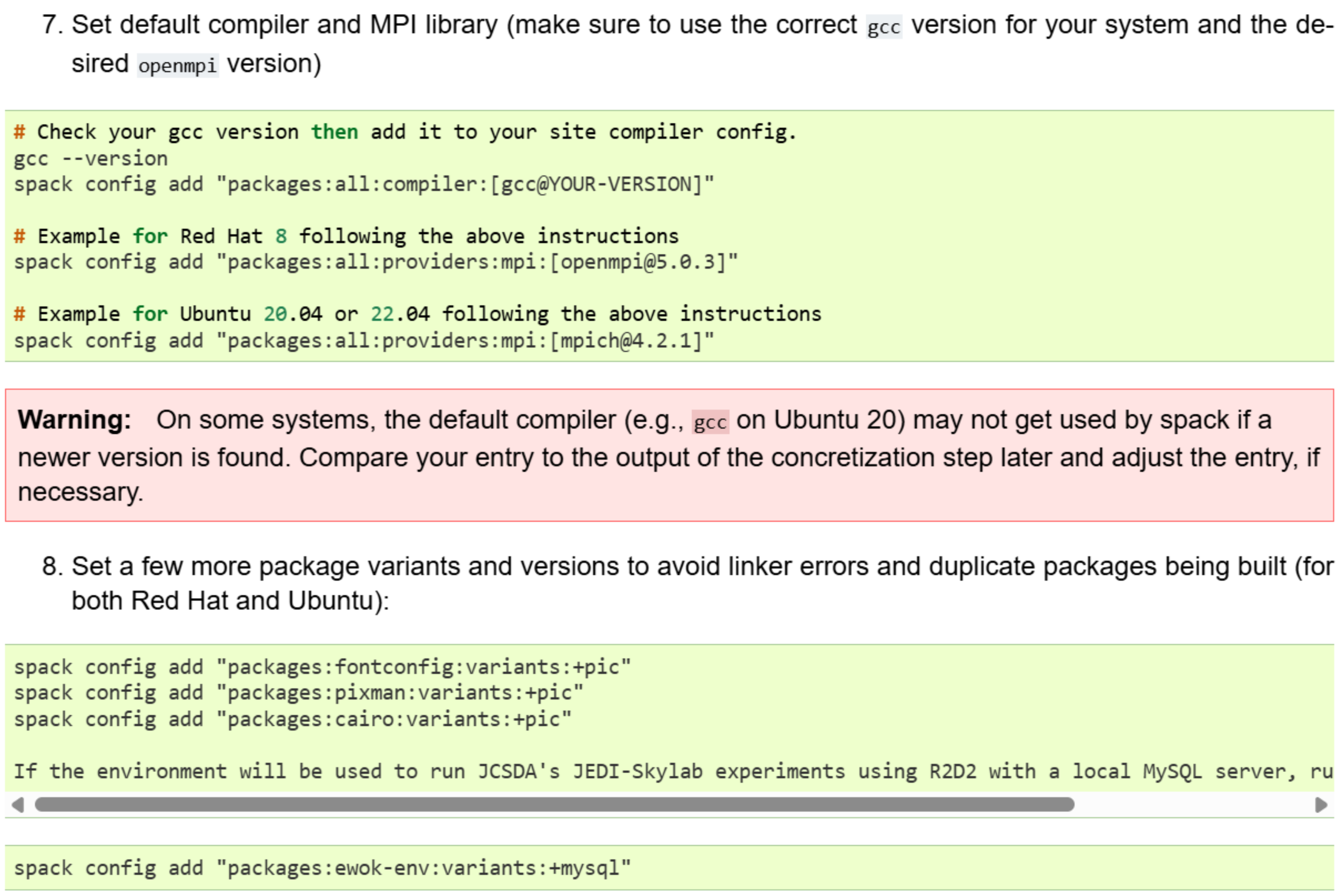
```
unset SPACK_SYSTEM_CONFIG_PATH
```

General Step 4: Set versions of important pkgs

Here we will tell spack what versions we want to use for our compilers and parallel processing along with a few other things.

Pay close attention as instructions can be system dependent

Again, there may be some things you do not need. It wont hurt to add them though if you aren't sure.



7. Set default compiler and MPI library (make sure to use the correct `gcc` version for your system and the desired `openmpi` version)

```
# Check your gcc version then add it to your site compiler config.
gcc --version
spack config add "packages:all:compiler:[gcc@YOUR-VERSION]"

# Example for Red Hat 8 following the above instructions
spack config add "packages:all:providers:mpi:[openmpi@5.0.3]"

# Example for Ubuntu 20.04 or 22.04 following the above instructions
spack config add "packages:all:providers:mpi:[mpich@4.2.1]"
```

Warning: On some systems, the default compiler (e.g., `gcc` on Ubuntu 20) may not get used by spack if a newer version is found. Compare your entry to the output of the concretization step later and adjust the entry, if necessary.

8. Set a few more package variants and versions to avoid linker errors and duplicate packages being built (for both Red Hat and Ubuntu):

```
spack config add "packages:fontconfig:variants:+pic"
spack config add "packages:pixman:variants:+pic"
spack config add "packages:cairo:variants:+pic"

If the environment will be used to run JCSDA's JEDI-Skylab experiments using R2D2 with a local MySQL server, ru
```

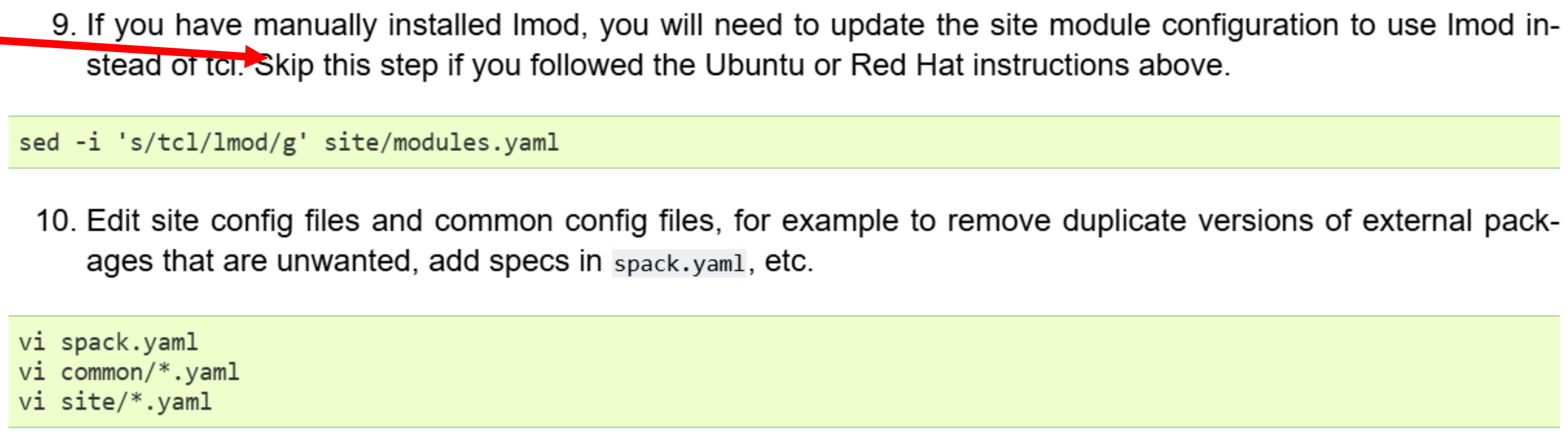
```
spack config add "packages:ewok-env:variants:+mysql"
```

General Step 5: A bit of clean up if needed.

Much of this might not be necessary and note for the linux distros tested you would skip step 9.

If you are installing spack on a clean system install you likely don't have to worry about editing these yamls. So, if you are using WSL or OrbStack having a fresh VM is not a bad idea.

Note: for macOS you should remove all compilers except apple-clang

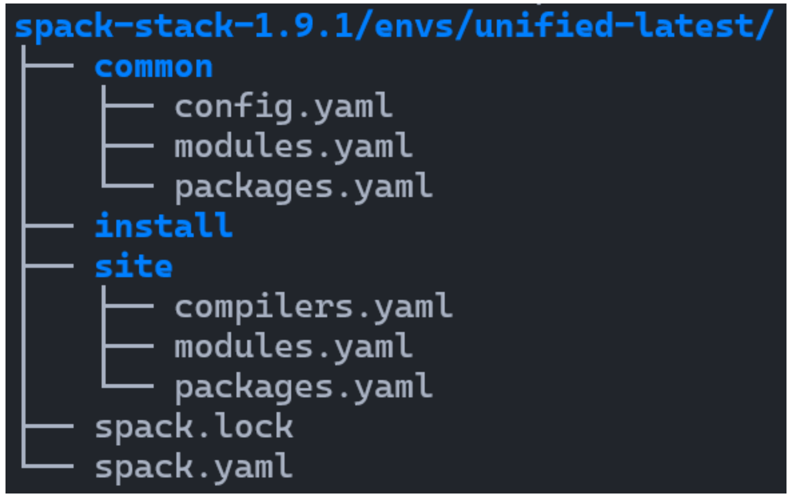


9. If you have manually installed lmod, you will need to update the site module configuration to use lmod instead of tcl. Skip this step if you followed the Ubuntu or Red Hat instructions above.

```
sed -i 's/tcl/lmod/g' site/modules.yaml
```

10. Edit site config files and common config files, for example to remove duplicate versions of external packages that are unwanted, add specs in `spack.yaml`, etc.

```
vi spack.yaml
vi common/*.yaml
vi site/*.yaml
```



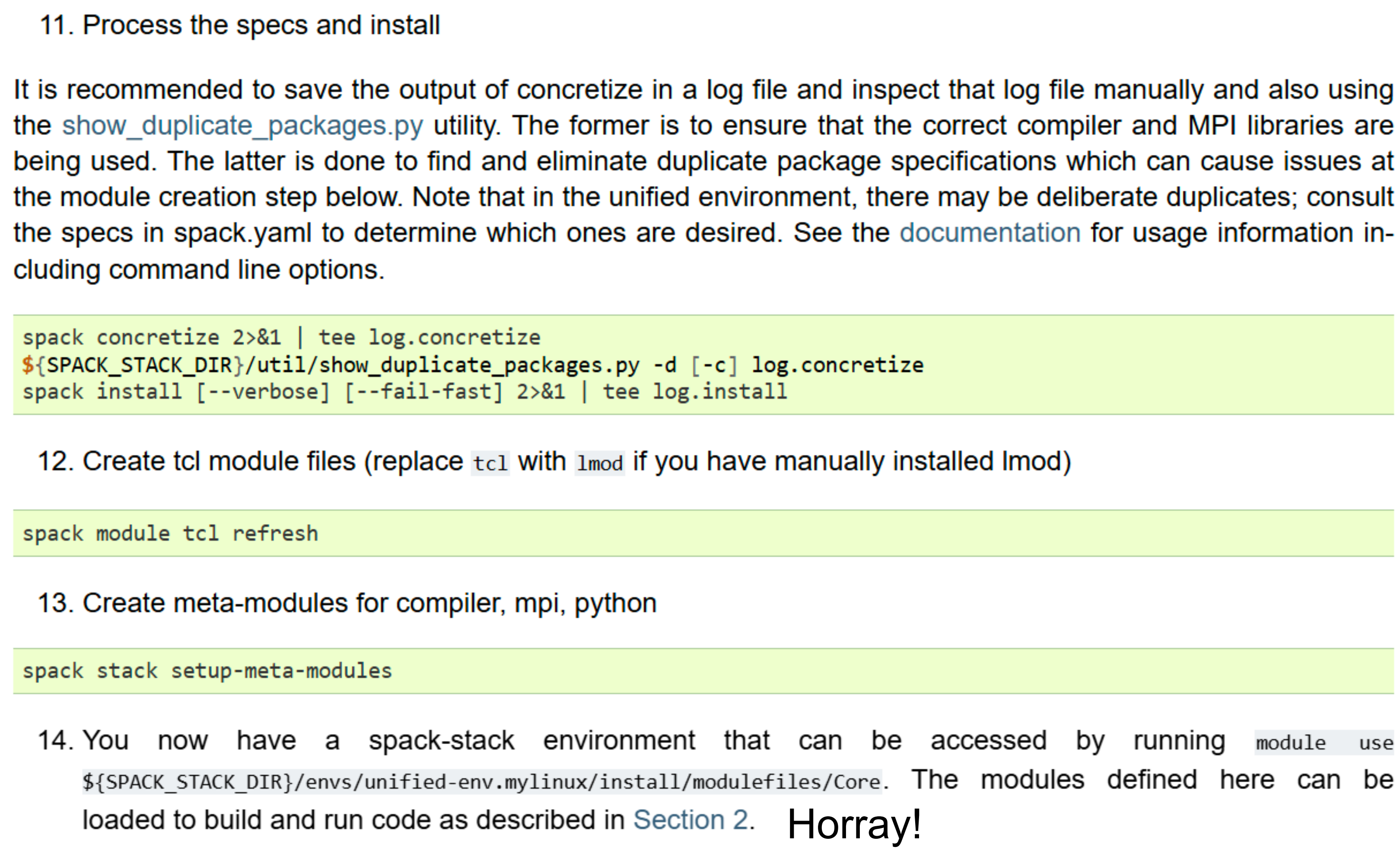
```
spack-stack-1.9.1/envs/unified-latest/
├── common
│   ├── config.yaml
│   ├── modules.yaml
│   └── packages.yaml
├── install
├── site
│   ├── compilers.yaml
│   ├── modules.yaml
│   └── packages.yaml
├── spack.lock
└── spack.yaml
```

General Step 6: Install

Now we are ready to install! This part will take a while, possibly several hours, you machine will get warm.

Spack will be building your stack, it will be cloning github repos, compiling code and downloading needed packages to run everything mpas-jedi.

Sometimes cloning a repo may time out! This will cause the install to fail. If you see the error is a download or clone time out, fear not! Just kick off the install again, sometimes you may have to do this a couple times.



11. Process the specs and install

It is recommended to save the output of concretize in a log file and inspect that log file manually and also using the show_duplicate_packages.py utility. The former is to ensure that the correct compiler and MPI libraries are being used. The latter is done to find and eliminate duplicate package specifications which can cause issues at the module creation step below. Note that in the unified environment, there may be deliberate duplicates; consult the specs in spack.yaml to determine which ones are desired. See the documentation for usage information including command line options.

```
spack concretize 2>&1 | tee log.concretize
${SPACK_STACK_DIR}/util/show_duplicate_packages.py -d [-c] log.concretize
spack install [--verbose] [--fail-fast] 2>&1 | tee log.install
```

12. Create tcl module files (replace `tcl` with `lmod` if you have manually installed lmod)

```
spack module tcl refresh
```

13. Create meta-modules for compiler, mpi, python

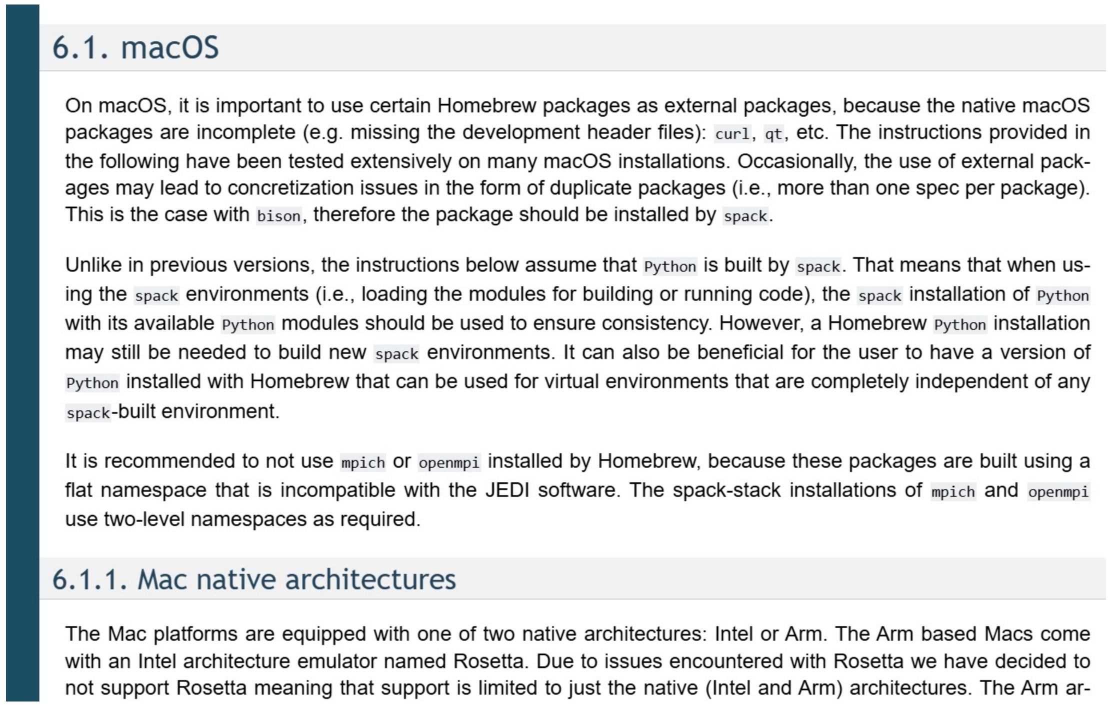
```
spack stack setup-meta-modules
```

14. You now have a spack-stack environment that can be accessed by running `module use ${SPACK_STACK_DIR}/envs/unified-env.mylinux/install/modulefiles/Core`. The modules defined here can be loaded to build and run code as described in Section 2.

Horray!

Make sure you follow for your system…

In this presentation we looked at examples of the "general steps" with examples of them for ***Linux***, which can also be used with WSL on Windows or OrbStack on Mac. Just note that if you want to install natively on mac there are different specific instructions making up the general ones!



6.1. macOS

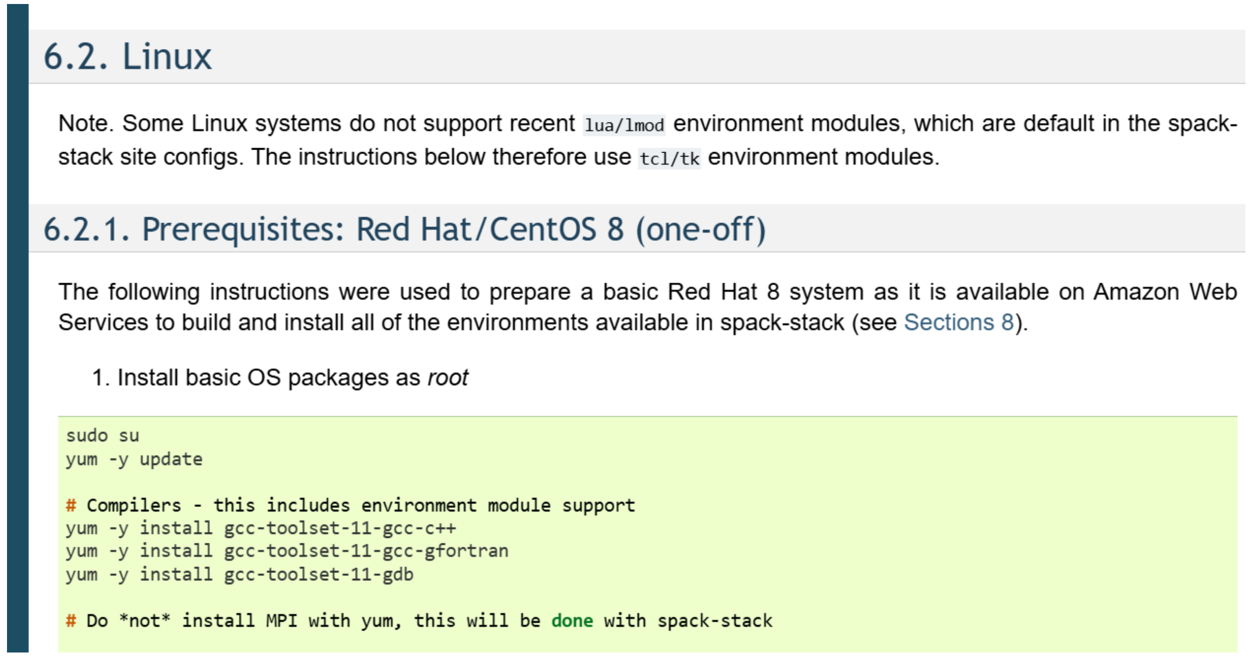
On macOS, it is important to use certain Homebrew packages as external packages, because the native macOS packages are incomplete (e.g. missing the development header files): `curl`, `qt`, etc. The instructions provided in the following have been tested extensively on many macOS installations. Occasionally, the use of external packages may lead to concretization issues in the form of duplicate packages (i.e., more than one spec per package). This is the case with `bison`, therefore the package should be installed by `spack`.

Unlike in previous versions, the instructions below assume that `Python` is built by `spack`. That means that when using the `spack` environments (i.e., loading the modules for building or running code), the `spack` installation of `Python` with its available `Python` modules should be used to ensure consistency. However, a Homebrew `Python` installation may still be needed to build new `spack` environments. It can also be beneficial for the user to have a version of `Python` installed with Homebrew that can be used for virtual environments that are completely independent of any `spack`-built environment.

It is recommended to not use `mpich` or `openmpi` installed by Homebrew, because these packages are built using a flat namespace that is incompatible with the JEDI software. The spack-stack installations of `mpich` and `openmpi` use two-level namespaces as required.

6.1.1. Mac native architectures

The Mac platforms are equipped with one of two native architectures: Intel or Arm. The Arm based Macs come with an Intel architecture emulator named Rosetta. Due to issues encountered with Rosetta we have decided to not support Rosetta meaning that support is limited to just the native (Intel and Arm) architectures. The Arm ar-



6.2. Linux

Note. Some Linux systems do not support recent `lua/lmod` environment modules, which are default in the spack-stack site configs. The instructions below therefore use `tcl/tk` environment modules.

6.2.1. Prerequisites: Red Hat/CentOS 8 (one-off)

The following instructions were used to prepare a basic Red Hat 8 system as it is available on Amazon Web Services to build and install all of the environments available in spack-stack (see Sections 8).

1. Install basic OS packages as *root*

```
sudo su
yum -y update

# Compilers - this includes environment module support
yum -y install gcc-toolset-11-gcc-c++
yum -y install gcc-toolset-11-gcc-gfortran
yum -y install gcc-toolset-11-gdb

# Do *not* install MPI with yum, this will be done with spack-stack
```

https://spack-stack.readthedocs.io/en/1.9.1/NewSiteConfigs.html

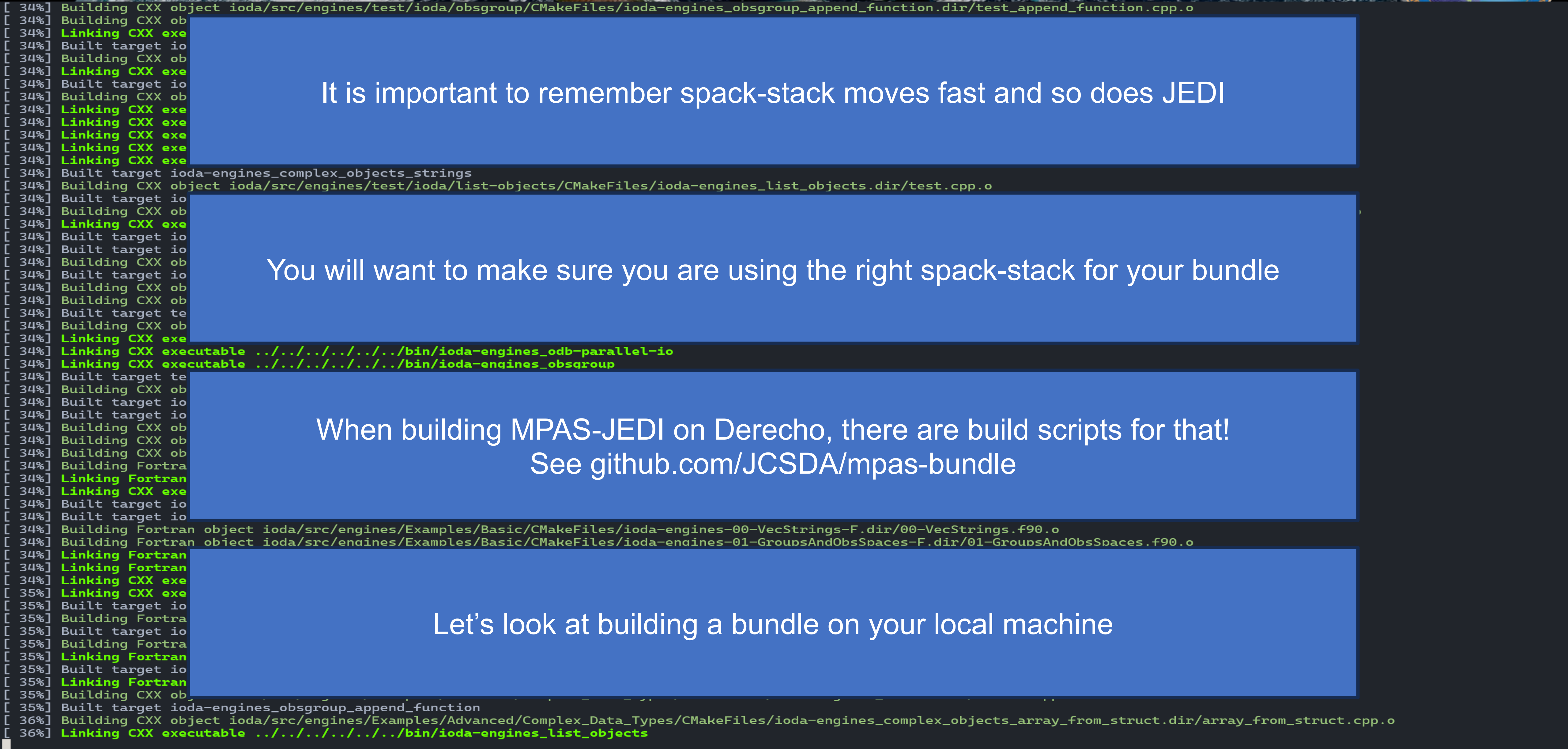
So, you have spack-stack and want to build JEDI..

It is important to remember spack-stack moves fast and so does JEDI

You will want to make sure you are using the right spack-stack for your bundle

When building MPAS-JEDI on Derecho, there are build scripts for that!
See github.com/JCSDA/mpas-bundle

Let’s look at building a bundle on your local machine

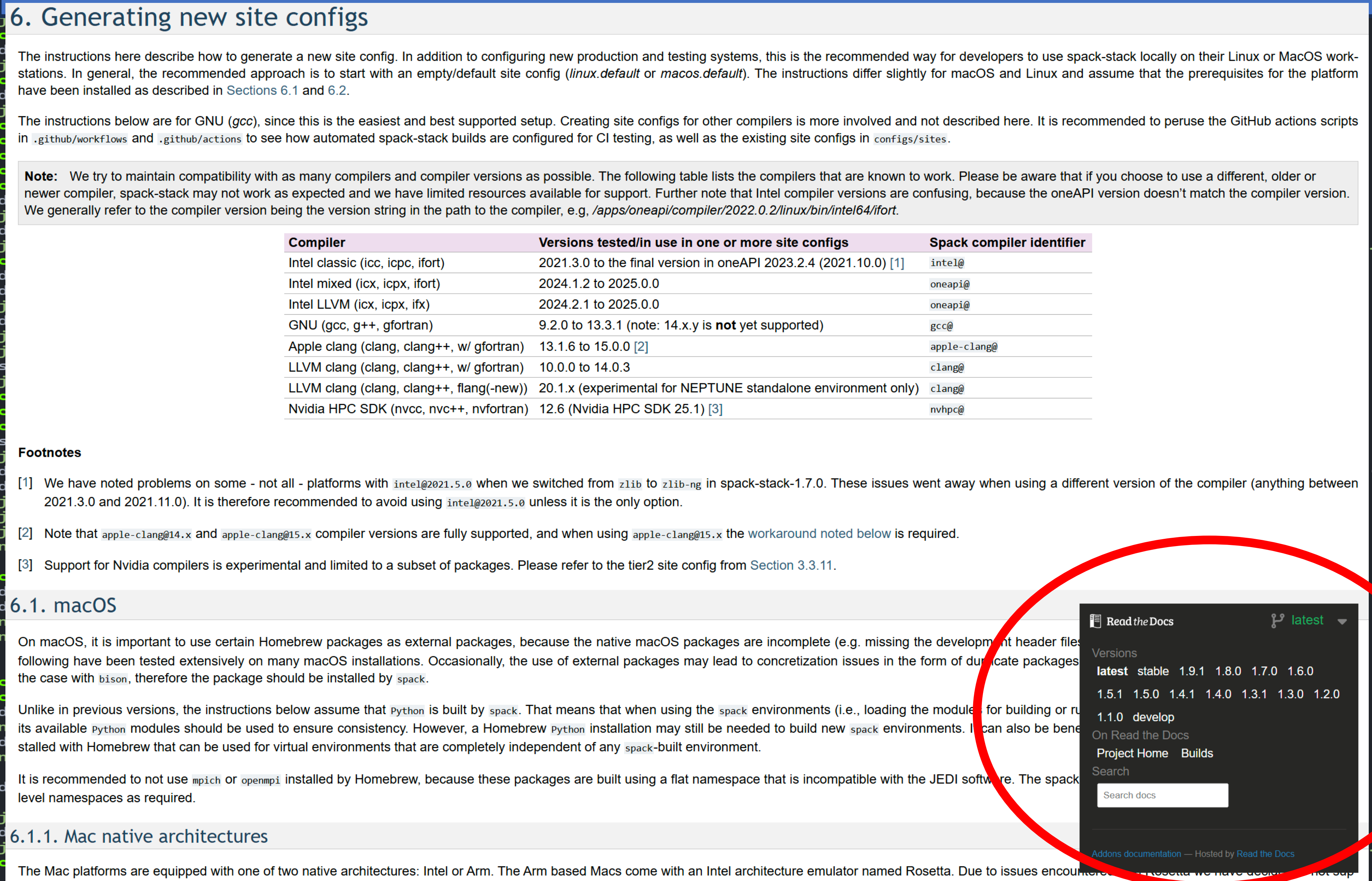


```
[ 34%] Building CXX object ioda/src/engines/test/ioda/obsgroup/CMakeFiles/ioda-engines_obsgroup_append_function.dir/test_append_function.cpp.o
[ 34%] Building CXX ob
[ 34%] Linking CXX exe
[ 34%] Built target io
[ 34%] Building CXX ob
[ 34%] Linking CXX exe
[ 34%] Built target io
[ 34%] Building CXX ob
[ 34%] Linking CXX exe
[ 34%] Linking CXX exe
[ 34%] Linking CXX exe
[ 34%] Linking CXX exe
[ 34%] Linking CXX exe
[ 34%] Built target ioda-engines_complex_objects_strings
[ 34%] Building CXX object ioda/src/engines/test/ioda/list-objects/CMakeFiles/ioda-engines_list_objects.dir/test.cpp.o
[ 34%] Built target io
[ 34%] Building CXX ob
[ 34%] Linking CXX exe
[ 34%] Built target io
[ 34%] Built target io
[ 34%] Building CXX ob
[ 34%] Built target io
[ 34%] Building CXX ob
[ 34%] Building CXX ob
[ 34%] Built target te
[ 34%] Building CXX ob
[ 34%] Linking CXX exe
[ 34%] Linking CXX executable ../../../../../../bin/ioda-engines_odb-parallel-io
[ 34%] Linking CXX executable ../../../../../../bin/ioda-engines_obsgroup
[ 34%] Built target te
[ 34%] Building CXX ob
[ 34%] Built target io
[ 34%] Built target io
[ 34%] Building CXX ob
[ 34%] Building CXX ob
[ 34%] Building CXX ob
[ 34%] Building Fortra
[ 34%] Linking Fortran
[ 34%] Linking CXX exe
[ 34%] Built target io
[ 34%] Built target io
[ 34%] Building Fortran object ioda/src/engines/Examples/Basic/CMakeFiles/ioda-engines-00-VecStrings-F.dir/00-VecStrings.f90.o
[ 34%] Building Fortran object ioda/src/engines/Examples/Basic/CMakeFiles/ioda-engines-01-GroupsAndObsSpaces-F.dir/01-GroupsAndObsSpaces.f90.o
[ 34%] Linking Fortran
[ 34%] Linking Fortran
[ 34%] Linking CXX exe
[ 35%] Linking CXX exe
[ 35%] Built target io
[ 35%] Building Fortra
[ 35%] Built target io
[ 35%] Building Fortra
[ 35%] Linking Fortran
[ 35%] Built target io
[ 35%] Linking Fortran
[ 35%] Building CXX ob
[ 35%] Built target ioda-engines_obsgroup_append_function
[ 36%] Building CXX object ioda/src/engines/Examples/Advanced/Complex_Data_Types/CMakeFiles/ioda-engines_complex_objects_array_from_struct.dir/array_from_struct.cpp.o
[ 36%] Linking CXX executable ../../../../../../bin/ioda-engines_list_objects
```

So, you

You will want to make sure you are using the right spack-stack for your bundle

```
[ 34%] Building CXX ob
[ 34%] Building CXX obj
[ 34%] Linking CXX exec
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Linking CXX exec
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Linking CXX exec
[ 34%] Linking CXX exec
[ 34%] Linking CXX exec
[ 34%] Linking CXX exec
[ 34%] Linking CXX exec
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Linking CXX exec
[ 34%] Built target iod
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Building CXX obj
[ 34%] Built target tes
[ 34%] Building CXX obj
[ 34%] Linking CXX exec
[ 34%] Linking CXX exec
[ 34%] Linking CXX exec
[ 34%] Built target tes
[ 34%] Building CXX obj
[ 34%] Built target iod
[ 34%] Built target iod
[ 34%] Building CXX obj
[ 34%] Building CXX obj
[ 34%] Building CXX obj
[ 34%] Building Fortran
[ 34%] Linking Fortran
[ 34%] Linking CXX exec
[ 34%] Built target iod
[ 34%] Built target iod
[ 34%] Building Fortran
[ 34%] Building Fortran
[ 34%] Linking Fortran
[ 34%] Linking Fortran
[ 34%] Linking CXX exec
[ 35%] Linking CXX exec
[ 35%] Built target iod
[ 35%] Building Fortran
[ 35%] Built target iod
[ 35%] Building Fortran
[ 35%] Linking Fortran
[ 35%] Built target iod
[ 35%] Linking Fortran
[ 35%] Building CXX obj
[ 35%] Built target iod
[ 36%] Building CXX obj
[ 36%] Linking CXX exec
```



6. Generating new site configs

The instructions here describe how to generate a new site config. In addition to configuring new production and testing systems, this is the recommended way for developers to use spack-stack locally on their Linux or MacOS workstations. In general, the recommended approach is to start with an empty/default site config (*linux.default* or *macos.default*). The instructions differ slightly for macOS and Linux and assume that the prerequisites for the platform have been installed as described in Sections 6.1 and 6.2.

The instructions below are for GNU (*gcc*), since this is the easiest and best supported setup. Creating site configs for other compilers is more involved and not described here. It is recommended to peruse the GitHub actions scripts in `.github/workflows` and `.github/actions` to see how automated spack-stack builds are configured for CI testing, as well as the existing site configs in `configs/sites`.

Note: We try to maintain compatibility with as many compilers and compiler versions as possible. The following table lists the compilers that are known to work. Please be aware that if you choose to use a different, older or newer compiler, spack-stack may not work as expected and we have limited resources available for support. Further note that Intel compiler versions are confusing, because the oneAPI version doesn't match the compiler version. We generally refer to the compiler version being the version string in the path to the compiler, e.g, */apps/oneapi/compiler/2022.0.2/linux/bin/intel64/ifort*.

Compiler	Versions tested/in use in one or more site configs	Spack compiler identifier
Intel classic (icc, icpc, ifort)	2021.3.0 to the final version in oneAPI 2023.2.4 (2021.10.0) [1]	`intel@`
Intel mixed (icx, icpx, ifort)	2024.1.2 to 2025.0.0	`oneapi@`
Intel LLVM (icx, icpx, ifx)	2024.2.1 to 2025.0.0	`oneapi@`
GNU (gcc, g++, gfortran)	9.2.0 to 13.3.1 (note: 14.x.y is **not** yet supported)	`gcc@`
Apple clang (clang, clang++, w/ gfortran)	13.1.6 to 15.0.0 [2]	`apple-clang@`
LLVM clang (clang, clang++, w/ gfortran)	10.0.0 to 14.0.3	`clang@`
LLVM clang (clang, clang++, flang(-new))	20.1.x (experimental for NEPTUNE standalone environment only)	`clang@`
Nvidia HPC SDK (nvcc, nvc++, nvfortran)	12.6 (Nvidia HPC SDK 25.1) [3]	`nvhpc@`

Footnotes

[1] We have noted problems on some - not all - platforms with `intel@2021.5.0` when we switched from `zlib` to `zlib-ng` in spack-stack-1.7.0. These issues went away when using a different version of the compiler (anything between 2021.3.0 and 2021.11.0). It is therefore recommended to avoid using `intel@2021.5.0` unless it is the only option.

[2] Note that `apple-clang@14.x` and `apple-clang@15.x` compiler versions are fully supported, and when using `apple-clang@15.x` the workaround noted below is required.

[3] Support for Nvidia compilers is experimental and limited to a subset of packages. Please refer to the tier2 site config from Section 3.3.11.

6.1. macOS

On macOS, it is important to use certain Homebrew packages as external packages, because the native macOS packages are incomplete (e.g. missing the developm nt header files following have been tested extensively on many macOS installations. Occasionally, the use of external packages may lead to concretization issues in the form of du licate packages the case with `bison`, therefore the package should be installed by `spack`.

Unlike in previous versions, the instructions below assume that `Python` is built by `spack`. That means that when using the `spack` environments (i.e., loading the module for building or ru its available `Python` modules should be used to ensure consistency. However, a Homebrew `Python` installation may still be needed to build new `spack` environments. I can also be bene stalled with Homebrew that can be used for virtual environments that are completely independent of any `spack`-built environment.

It is recommended to not use `mpich` or `openmpi` installed by Homebrew, because these packages are built using a flat namespace that is incompatible with the JEDI softw re. The spack level namespaces as required.

6.1.1. Mac native architectures

The Mac platforms are equipped with one of two native architectures: Intel or Arm. The Arm based Macs come with an Intel architecture emulator named Rosetta. Due to issues encou

Documentation for other versions found here

So, you have spack-stack and want to build JEDI..

You will need to configure git, once per system but don't forget!

```
git lfs install --skip-repo
git config --global user.name <username-for-github>
git config --global user.email <email-used-for-github>
git config --global credential.helper 'cache --timeout=3600'
git config --global --add credential.helper 'store'
```

Two Choices

jedi-bundle

(https://github.com/JCSDA/jedi-bundle)

- Large bundle which includes everything to run DA with mpas, fv3, and mom6 models
    - JEDI (oops, vader, saber, ufo, etc.)
    - fv3-jedi, mpas-jedi, and soca model interfaces
    - fv3-dynamical core, MPAS-A model, mom6 model

You can also comment out the other models in Cmakelists.txt to only build MPAS!

mpas-bundle

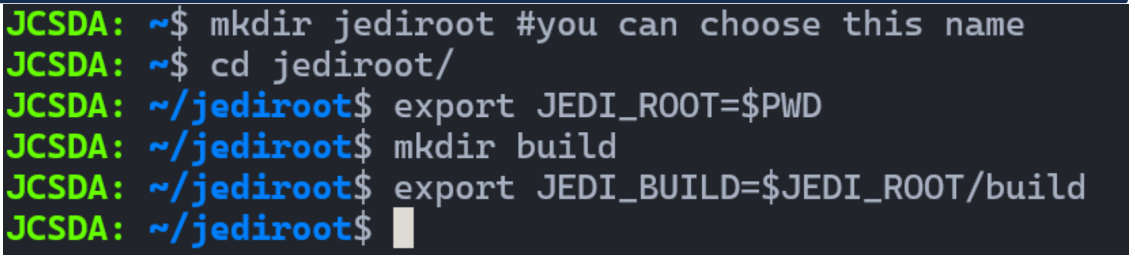
(https://github.com/JCSDA/mpas-bundle)

- Small bundle includes everything to run DA with only mpas:
- JEDI (oops, vader, saber, ufo, etc),
- MPAS-JEDI (interface), & MPAS-A mode

(Recommended if only using MPAS)

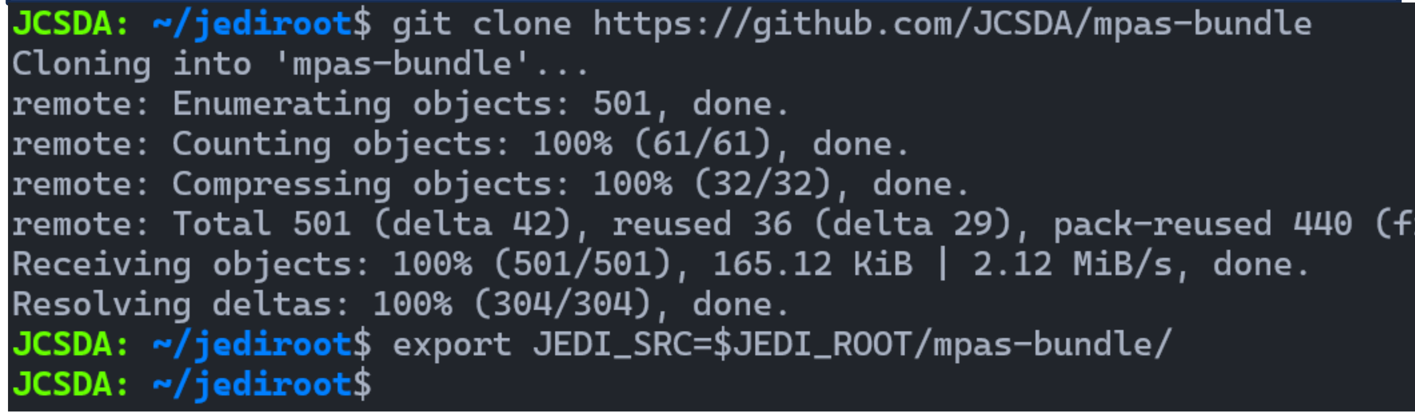
Building a Bundle (get everything you need)

1. Make a JEDI_ROOT and build directory and export the paths.



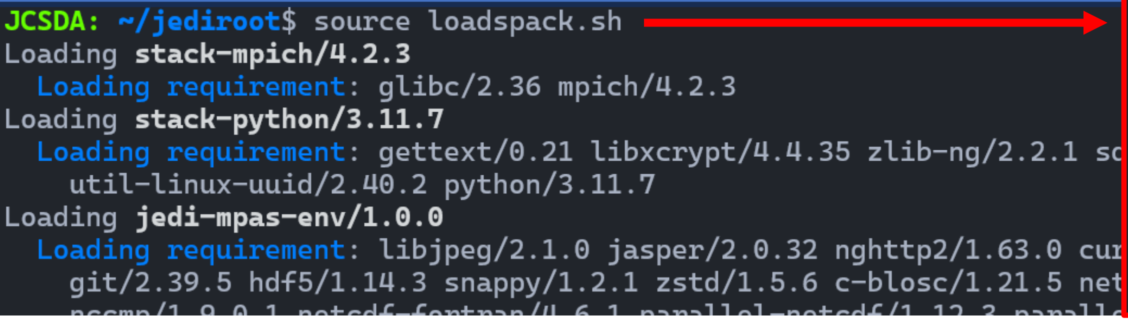
```
JCSDA: ~$ mkdir jediroot #you can choose this name
JCSDA: ~$ cd jediroot/
JCSDA: ~/jediroot$ export JEDI_ROOT=$PWD
JCSDA: ~/jediroot$ mkdir build
JCSDA: ~/jediroot$ export JEDI_BUILD=$JEDI_ROOT/build
JCSDA: ~/jediroot$ █
```

2. Clone mpas-bundle and set your src path

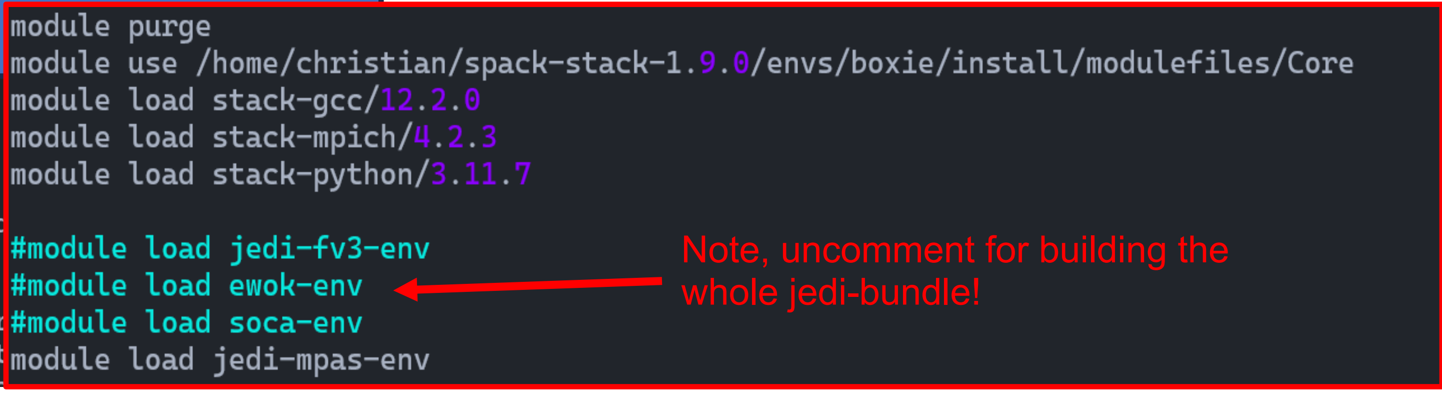


```
JCSDA: ~/jediroot$ git clone https://github.com/JCSDA/mpas-bundle
Cloning into 'mpas-bundle'...
remote: Enumerating objects: 501, done.
remote: Counting objects: 100% (61/61), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 501 (delta 42), reused 36 (delta 29), pack-reused 440 (f
Receiving objects: 100% (501/501), 165.12 KiB | 2.12 MiB/s, done.
Resolving deltas: 100% (304/304), done.
JCSDA: ~/jediroot$ export JEDI_SRC=$JEDI_ROOT/mpas-bundle/
JCSDA: ~/jediroot$
```

3. Load the packages you need



```
JCSDA: ~/jediroot$ source loadspack.sh
Loading stack-mpich/4.2.3
  Loading requirement: glibc/2.36 mpich/4.2.3
Loading stack-python/3.11.7
  Loading requirement: gettext/0.21 libxcrypt/4.4.35 zlib-ng/2.2.1 sc
    util-linux-uuid/2.40.2 python/3.11.7
Loading jedi-mpas-env/1.0.0
  Loading requirement: libjpeg/2.1.0 jasper/2.0.32 nghttp2/1.63.0 cur
    git/2.39.5 hdf5/1.14.3 snappy/1.2.1 zstd/1.5.6 c-blosc/1.21.5 net
    nccmp/1.9.0.1 netcdf-fortran/4.6.1 parallel-netcdf/1.12.3 paralle
```



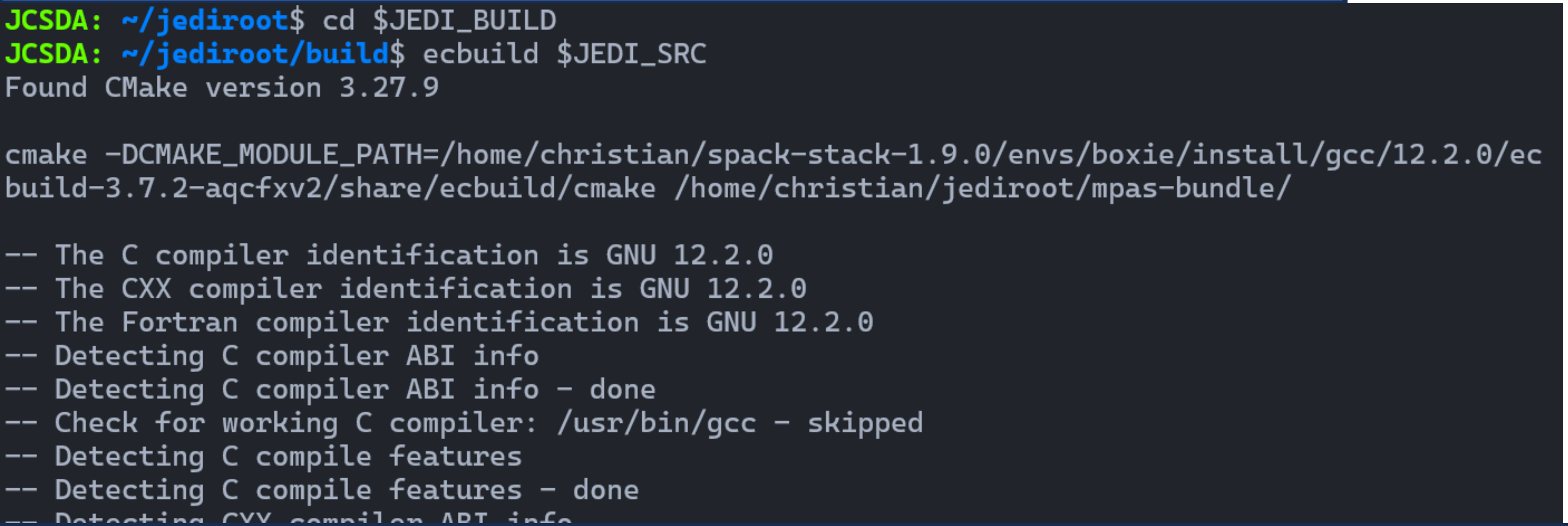
```
module purge
module use /home/christian/spack-stack-1.9.0/envs/boxie/install/modulefiles/Core
module load stack-gcc/12.2.0
module load stack-mpich/4.2.3
module load stack-python/3.11.7

#module load jedi-fv3-env
#module load ewok-env
#module load soca-env
module load jedi-mpas-env
```

Note, uncomment for building the whole jedi-bundle!

Now we are ready to build!

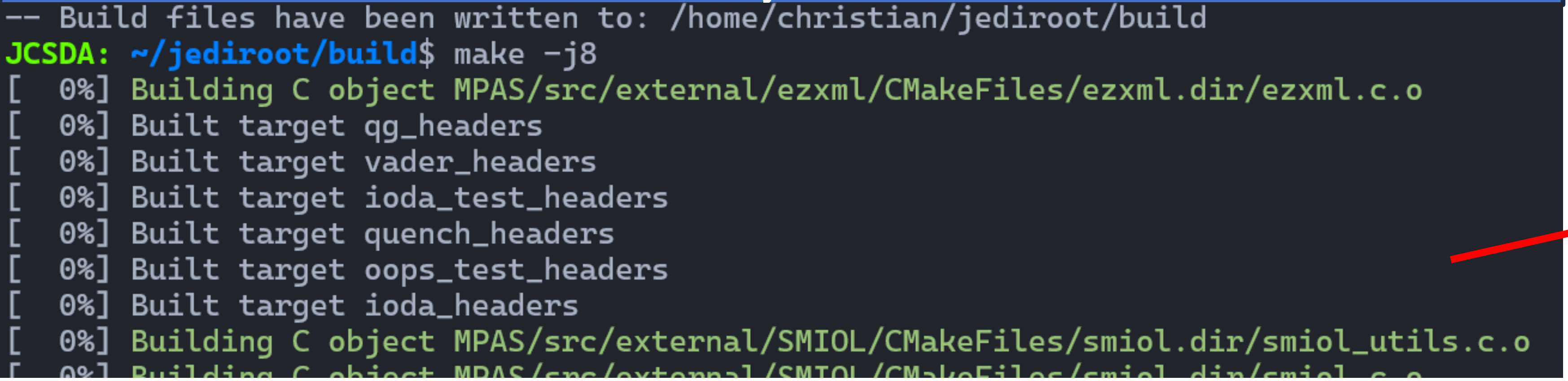
Building a Bundle (configure and build)

4. Run ecbuild and grab a coffee….

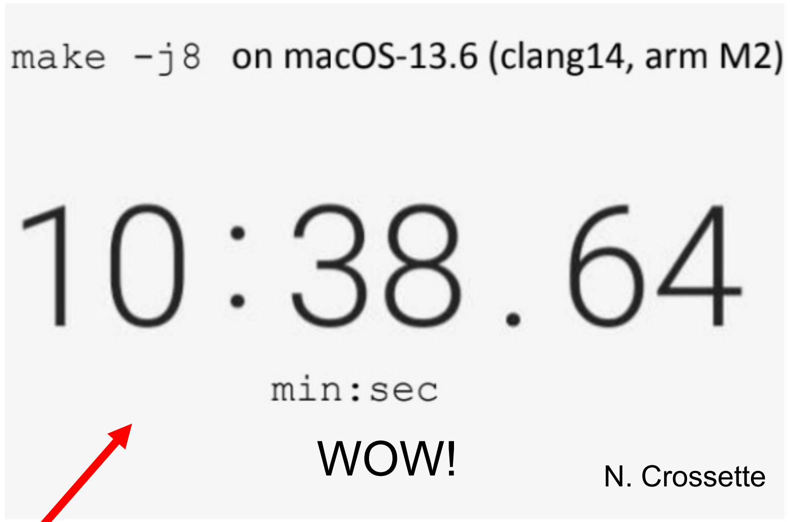


```
JCSDA: ~/jediroot$ cd $JEDI_BUILD
JCSDA: ~/jediroot/build$ ecbuild $JEDI_SRC
Found CMake version 3.27.9

cmake -DCMAKE_MODULE_PATH=/home/christian/spack-stack-1.9.0/envs/boxie/install/gcc/12.2.0/ec
build-3.7.2-aqcfxv2/share/ecbuild/cmake /home/christian/jediroot/mpas-bundle/

-- The C compiler identification is GNU 12.2.0
-- The CXX compiler identification is GNU 12.2.0
-- The Fortran compiler identification is GNU 12.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
```

6. Make –j[2,4 ,8, 16] the number of processes depends on your system.
More coffee may be warranted …



```
-- Build files have been written to: /home/christian/jediroot/build
JCSDA: ~/jediroot/build$ make -j8
[  0%] Building C object MPAS/src/external/ezxml/CMakeFiles/ezxml.dir/ezxml.c.o
[  0%] Built target qg_headers
[  0%] Built target vader_headers
[  0%] Built target ioda_test_headers
[  0%] Built target quench_headers
[  0%] Built target oops_test_headers
[  0%] Built target ioda_headers
[  0%] Building C object MPAS/src/external/SMIOL/CMakeFiles/smiol.dir/smiol_utils.c.o
[  0%] Building C object MPAS/src/external/SMIOL/CMakeFiles/smiol.dir/smiol.c.o
```



make -j8 on macOS-13.6 (clang14, arm M2)

10:38.64

min:sec

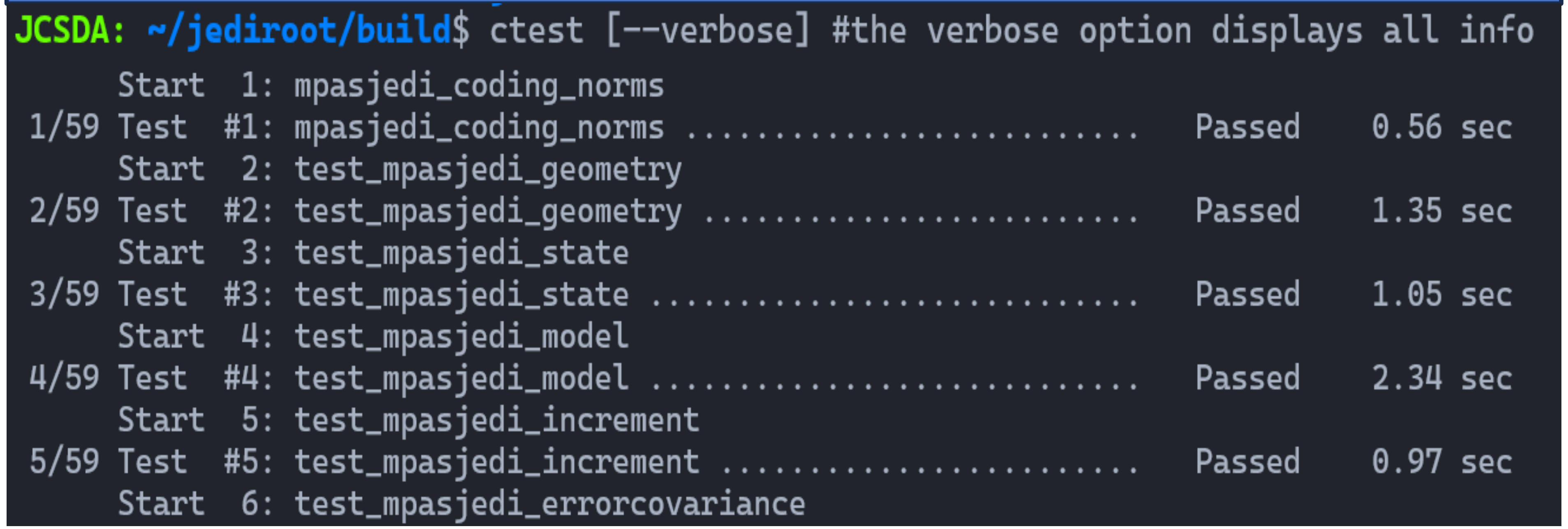
WOW!

N. Crossette

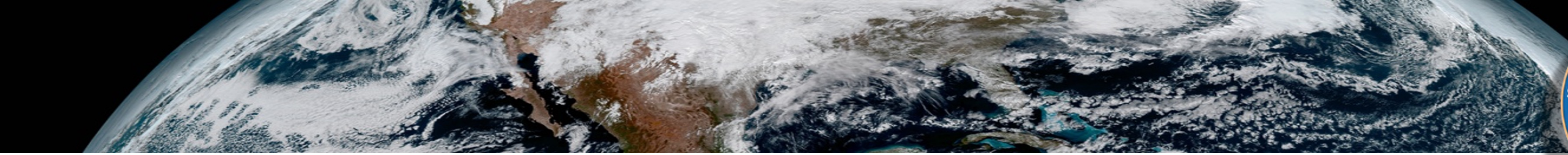
~18 min build with make –j8 on Debian Bookworm with AMD chip and GNU compiler (haven't tried clang) 14 min with make –j16 (hyper-threading)

Building a Bundle (run the ctests)

7. Run the ctests, this is to make sure your build was successful! Time for more coffee!



```
JCSDA: ~/jediroot/build$ ctest [--verbose] #the verbose option displays all info
      Start  1: mpasjedi_coding_norms
1/59 Test  #1: mpasjedi_coding_norms ..........................   Passed    0.56 sec
      Start  2: test_mpasjedi_geometry
2/59 Test  #2: test_mpasjedi_geometry .........................   Passed    1.35 sec
      Start  3: test_mpasjedi_state
3/59 Test  #3: test_mpasjedi_state ............................   Passed    1.05 sec
      Start  4: test_mpasjedi_model
4/59 Test  #4: test_mpasjedi_model ............................   Passed    2.34 sec
      Start  5: test_mpasjedi_increment
5/59 Test  #5: test_mpasjedi_increment ........................   Passed    0.97 sec
      Start  6: test_mpasjedi_errorcovariance
```



Questions?

JEDI Documentation: https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/index.html
JEDI Forum: https://forums.jcsda.org/ (requires account to post/comment)
Github: https://github.com/JCSDA (public)