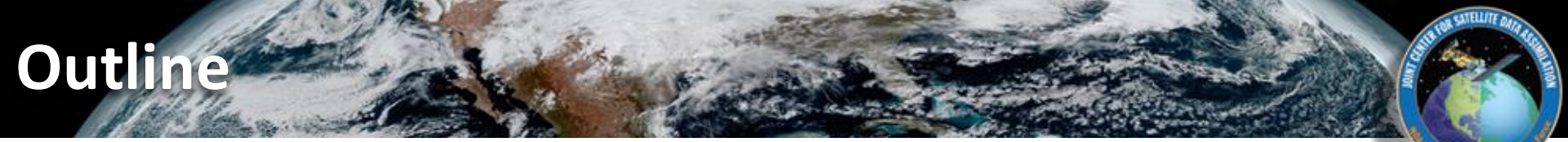
Spack-Stack: building JEDI bundles on your own machine

MPAS-JEDI Tutorial Boulder August 2026

Evan Parker

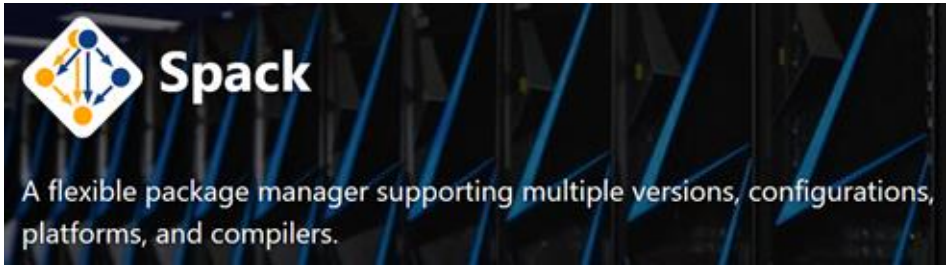
Contributors: Ashley Griffin Nate Crossette, Fabio Diniz, Christian Sampson

Outline



- What is Spack and spack-stack?
- Using spack-stack on supported machines
- Building spack-stack locally
- Building JEDI-bundle

Spack



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.

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:

Packages can peacefully coexist

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

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 an inter-agency collaboration to provision unified software environments for all users on HPCs, cloud platforms, macOS and Linux laptops. spack-stack greatly simplifies the development and deployment of JEDI and uses community code (spack) under the hood.

spack-stack powered by Spack

Developers and Users

- JCSDA
- NOAA NWS,EMC,OAR,EPIC
- Navy NRL

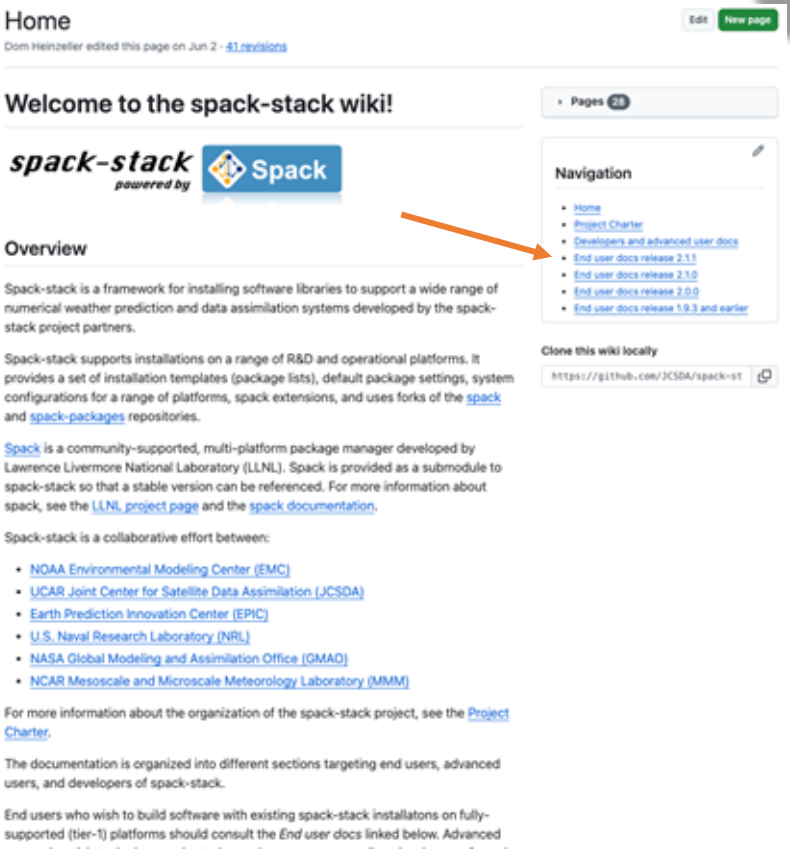
Users

- DTC
- NASA GMAO
- You?

Documentation

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

https://github.com/JCSDA/spack-stack/wiki

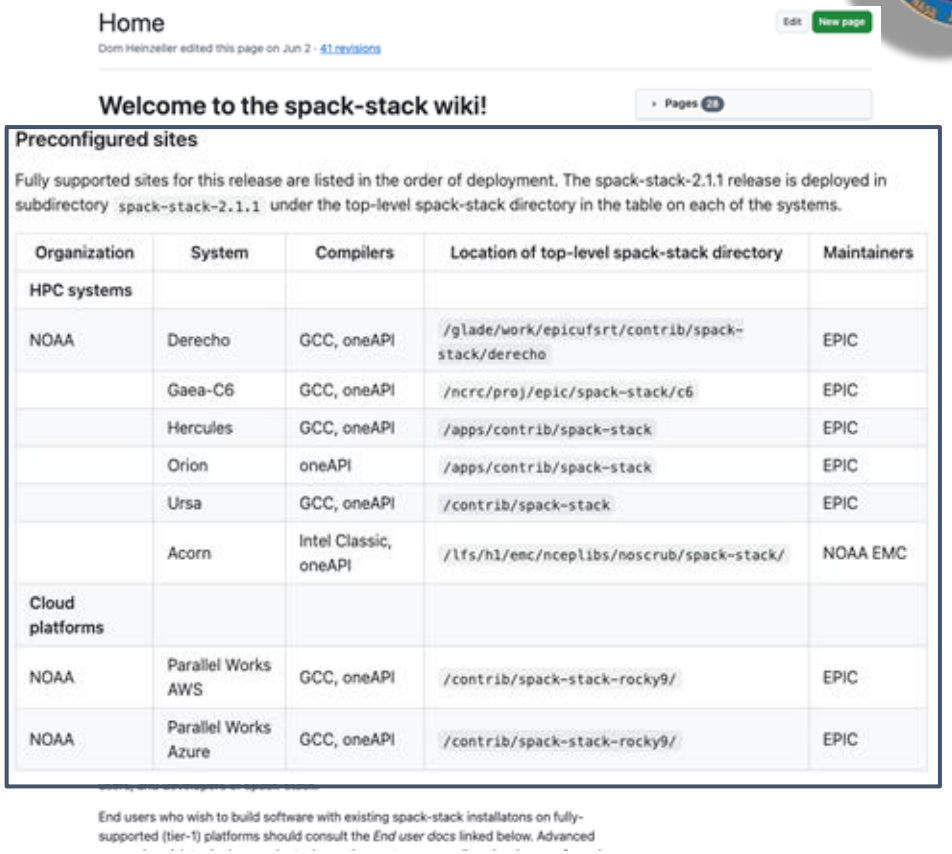


Documentation

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.

github.com/JCSDA/spack-stack/wiki/Release-2.1.1#preconfigured-sites



Home

Dom Heinzeller edited this page on Jun 2 · 41 revisions

Edit | New page

Welcome to the spack-stack wiki!

Pages 28

Preconfigured sites

Fully supported sites for this release are listed in the order of deployment. The spack-stack-2.1.1 release is deployed in subdirectory `spack-stack-2.1.1` under the top-level spack-stack directory in the table on each of the systems.

Organization	System	Compilers	Location of top-level spack-stack directory	Maintainers
HPC systems				
NOAA	Derecho	GCC, oneAPI	`/glade/work/epicufsrt/contrib/spack-stack/derecho`	EPIC
	Gaea-C6	GCC, oneAPI	`/ncrc/proj/epic/spack-stack/c6`	EPIC
	Hercules	GCC, oneAPI	`/apps/contrib/spack-stack`	EPIC
	Orion	oneAPI	`/apps/contrib/spack-stack`	EPIC
	Ursa	GCC, oneAPI	`/contrib/spack-stack`	EPIC
	Acorn	Intel Classic, oneAPI	`/lfs/h1/emc/nceplibs/noscrub/spack-stack/`	NOAA EMC
Cloud platforms				
NOAA	Parallel Works AWS	GCC, oneAPI	`/contrib/spack-stack-rocky9/`	EPIC
NOAA	Parallel Works Azure	GCC, oneAPI	`/contrib/spack-stack-rocky9/`	EPIC

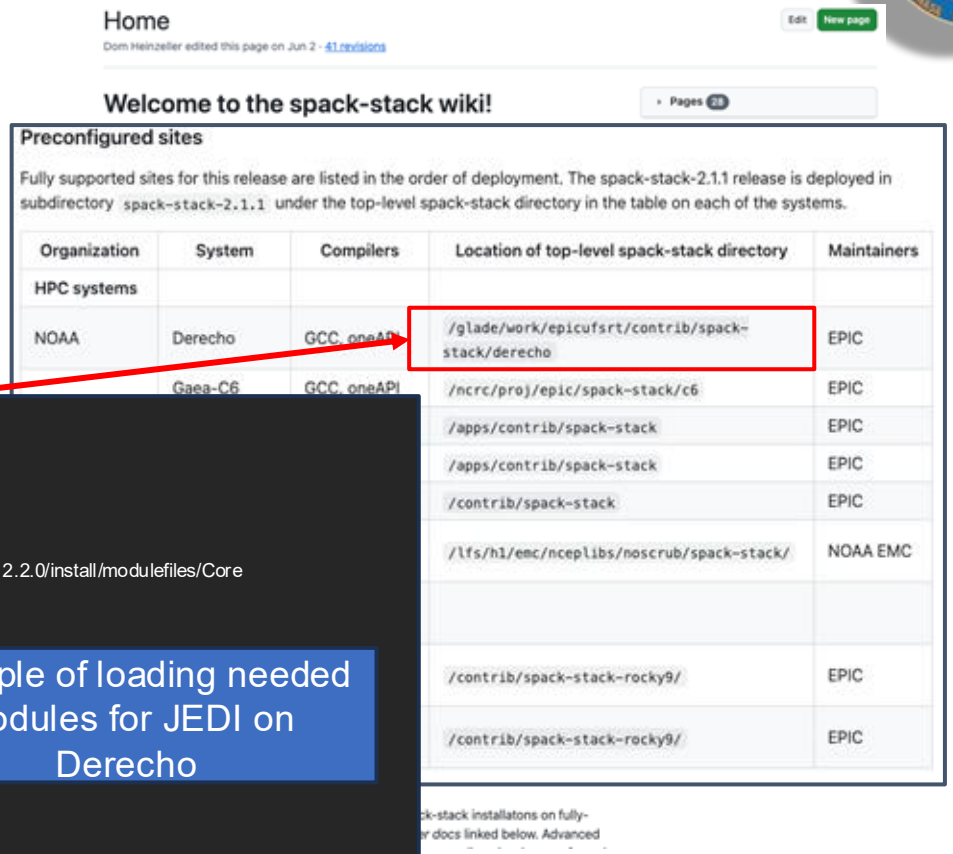
End users who wish to build software with existing spack-stack installatons on fully-supported (tier-1) platforms should consult the *End user docs* linked below. Advanced

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.

github.com/JCSDA/spack-stack/wiki/Release-2.1.1#preconfigured-sites



Home

Dom Heinzeller edited this page on Jun 2 · 41 revisions

Edit | New page

Welcome to the spack-stack wiki!

Pages 28

Preconfigured sites

Fully supported sites for this release are listed in the order of deployment. The spack-stack-2.1.1 release is deployed in subdirectory `spack-stack-2.1.1` under the top-level spack-stack directory in the table on each of the systems.

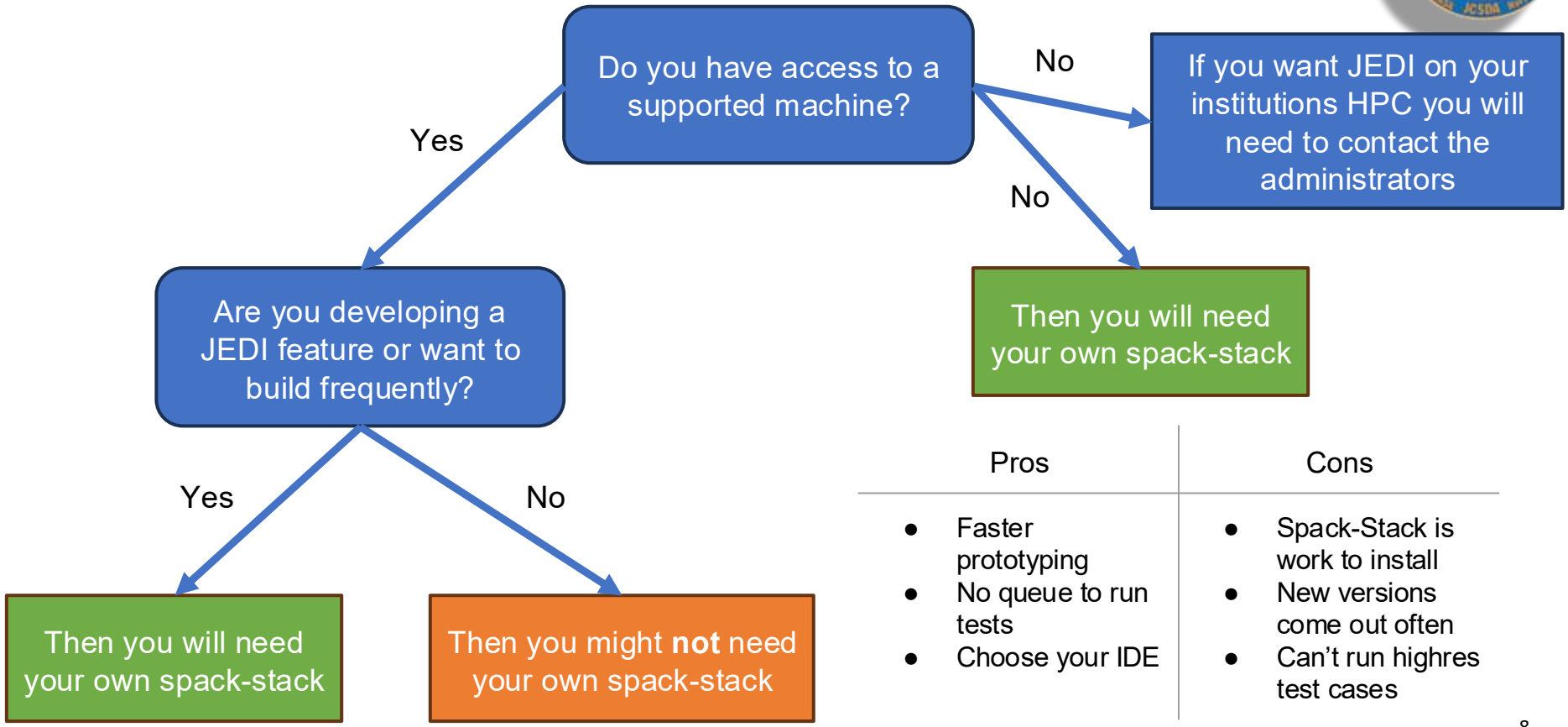
Organization	System	Compilers	Location of top-level spack-stack directory	Maintainers
HPC systems				
NOAA	Derecho	GCC, oneAPI	/glade/work/epicufsrt/contrib/spack-stack/derecho	EPIC
	Gaea-C6	GCC, oneAPI	/ncrc/proj/epic/spack-stack/c6	EPIC
	Hera	GCC, oneAPI, Intel Classic	/apps/contrib/spack-stack	EPIC
	Hercules	GCC, oneAPI, Intel Classic	/apps/contrib/spack-stack	EPIC
	Orion	GCC, oneAPI, Intel Classic	/contrib/spack-stack	EPIC
	WCOSS2	GCC, Intel Classic	/lfs/h1/emc/nceplibs/noscrub/spack-stack/	NOAA EMC
	Gaea-C5	GCC, oneAPI	/contrib/spack-stack-rocky9/	EPIC
	Jet	GCC, oneAPI, Intel Classic	/contrib/spack-stack-rocky9/	EPIC

...k-stack installatons on fully-
...r docs linked below. Advanced

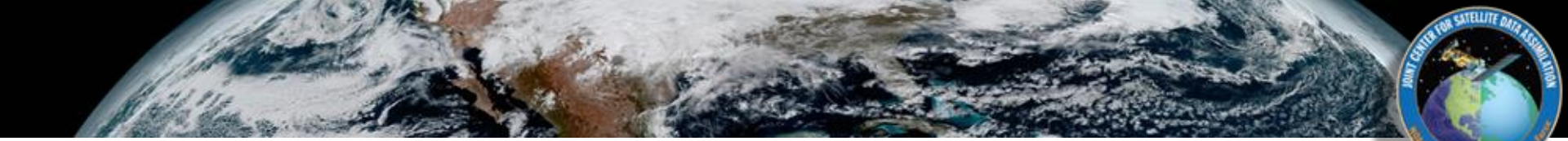
```
module purge
# ignore that the sticky module ncarenv/... is not unloaded
export LMOD_TMOD_FIND_FIRST=yes
module load ncarenv/23.09
module use /glade/work/epicufsrt/contrib/spack-stack/derecho/modulefiles

#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

Do you need local spack-stack?



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



Installing Spack-Stack

So you want to install spack-stack…

Linux

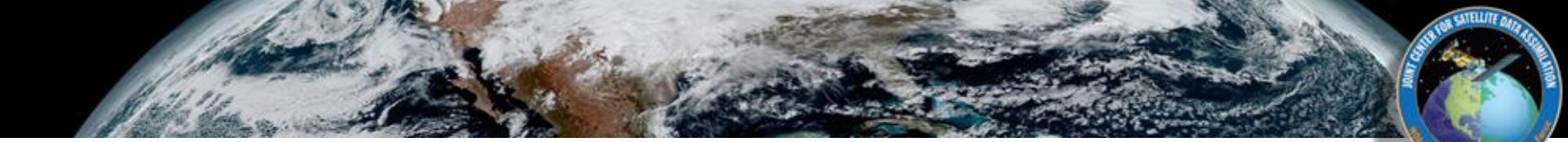
- Can you install spack-stack natively? Yes!
- Any distro can work, but Ubuntu and RedHat/CentOS have more support and are tested by maintainers
- Linux is the best option for ease of install and mostly for speed when compiling JEDI

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!

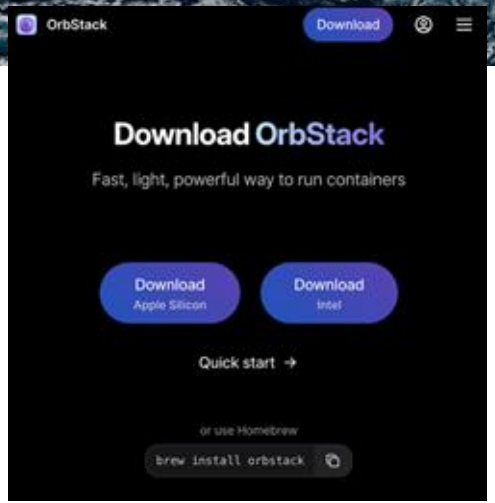
MacOS

- 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.



Installing spack-stack with OrbStack

Installing OrbStack



OrbStack: runs Docker containers and Linux

1. Download from https://orbstack.dev/download
2. Launch the app

Additional notes (if not automatically done):

- In your shell, add the OrbStack directory to your path as instructed when you launch the app. e.g.: export PATH=$PATH:/Users/<user>/.orbstack/bin
- In OrbStack settings, up the max allowed memory to at least 16GB

Documentation at https://docs.orbstack.dev/

OrbStack: Creating a virtual machine

Run: `orb create ubuntu:noble <vm-name>`

Note, the latest Ubuntu version does not work with spack-stack so you will need to specify Noble. See spack-stack docs under Prerequisites: Ubuntu (one-off) for updated system information.

Run: `orb` to take you to a shell in your default machine

Make sure you know what directory you're in before installing anything so you don't accidentally install things on the MacOS side of the filesystem. If you do a "pwd" command and you're in something that starts with "/Users/*", you're probably in the MacOS filesystem. Do a "cd" command to be taken to the Linux home directory.

```
agriffin@UCAR-OJUS ~ % orb create ubuntu:noble ubuntu-vm
agriffin@UCAR-OJUS ~ % orb
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

agriffin@ubuntu-vm:/Users/agriffin$ pwd
/Users/agriffin
agriffin@ubuntu-vm:/Users/agriffin$ echo ~
/home/agriffin
```

Building spack-stack: OrbStack prerequisites

This example is for Ubuntu Noble (24.04), using the gcc 13.3.0 compilers and OpenMPI 5.0.5

Add a `.bash_profile` containing the following. “umask” will be needed later in the spack-stack install.

```
# ~/.bash_profile

umask 0022
ulimit -S -s unlimited

if [[ ~/.bashrc ]]; then
  . ~/.bashrc

fi
cd ~
```

Building spack-stack: Ubuntu prerequisites

Follow the instructions for Prerequisites for Ubuntu (one-off):
https://github.com/JCSDA/spack-stack/wiki/New-Site-Configs#prerequisites-ubuntu-one-off

1. Install basic OS packages as root
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

When installing the gcc compilers, if you would like to make sure you install version 13 use:

```
# Compilers
apt install -y gcc-13 g++-13 gfortran-13 gdb gcc-plugin-dev
```

Building spack-stack: Create a new environment

Follow the instructions for Creating a new environment:
https://github.com/JCSDA/spack-stack/wiki/New-Site-Configs#creating-a-new-environment

In the docs, when you see [ ] this indicates that you have choices to make

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`).

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-stacl
cd spack-stack
source setup.sh
```

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 here and `configs/templates/` in the spack-stack repo for the available options.

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

Building spack-stack: Create a new environment

These steps allow spack to find all of the packages we already installed in the prerequisites.

Still in the environment directory, temporarily set environment variable `SPACK_SYSTEM_CONFIG_PATH` to modify site config files in `site` und unset `SPACK_DISABLE_LOCAL_CONFIG`.

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

Find external packages, add to site config's `packages.yaml`. If an external's bin directory hasn't been added to `$PATH`, you will need to prefix the command (`PATH=/missing/path/to/bin:$PATH spack external find ...`).

```
spack external find --scope system \
    --exclude bison --exclude meson \
    --exclude curl --exclude openssl \
    --exclude openssh --exclude python
spack external find --scope system grep
spack external find --scope system wget

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

Next, find compilers, add to `site/packages.yaml` as externals. If you have multiple versions of a compiler you will need to manually remove the unwanted versions from `site/packages.yaml` to prevent their use.

```
# Add compilers to the top of site/packages.yaml.
spack compiler find --scope system

# Edit site/packages.yaml to delete or comment unwanted compiler versions.
```

Do **not** forget to unset the `SPACK_SYSTEM_CONFIG_PATH` environment variable and restore the `SPACK_DISABLE_LOCAL_CONFIG` variable before the next steps.

Building spack-stack: Create a new environment

These steps allow you to define your compiler versions.

#9 can be skipped since we are using Ubuntu

#10 will also probably not be used since your OrbStack is a fresh “vm”, but it is a good idea to take a look at these files anyways

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"
```

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
```

Building spack-stack: Install

Finally we are ready to install spack-stack! This typically takes a while, sometimes several hours.

Tip: Use `nohup` or `screen` to keep your install job active (3 to 10 hours)

At this stage spack will build your stack by downloading packages and compiling code. It will give you everything you need to run mpas-jedi

```
spack concretize --force --fresh 2>&1 | tee log.concretize
${SPACK_STACK_DIR}/util/show_duplicate_packages.py
spack stack check-preferred-compiler
spack install [--verbose] [--fail-fast] 2>&1 | tee log.install
```

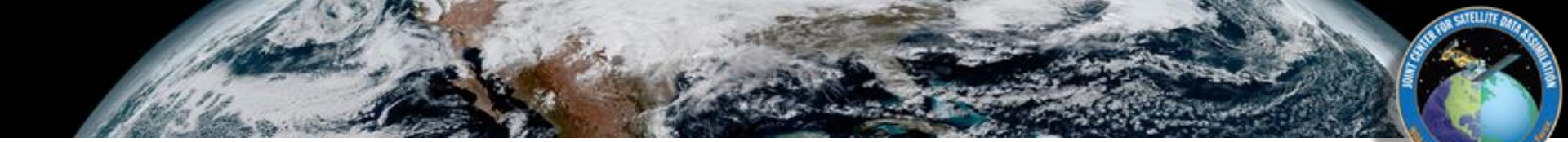
Create `tcl` module files (replace `tcl` with `lmod` if you have manually installed `lmod`).

```
spack module tcl refresh
```

Create meta-modules for the compiler and mpi provider.

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

Sometimes transient network issues (GitHub) 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.



Building a JEDI bundle

Building JEDI Bundle

Building JEDI documentation

- 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

Building JEDI Bundle: Prerequisites

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

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 sq
    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
```

```
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

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
```

5. Make –j[2,4 ,8, 16] the number of processes depends on your system.

```
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
```

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

10:38.64

min:sec

WOW!

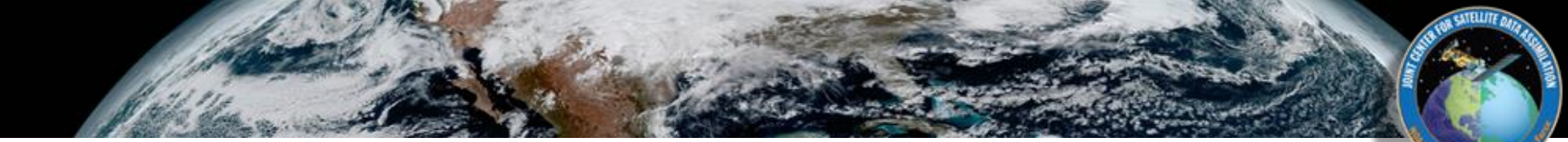
N. Crossette

~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

6. 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