HPC-Stack Users Guide

Jun 01, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Definition: The HPC-Stack is a repository that provides a unified, shell script-based build system to build the software stack required for numerical weather prediction (NWP) tools such as the Unified Forecast System (UFS) and the Joint Effort for Data assimilation Integration (JEDI) framework.

1.1 Background

The HPC-Stack provides libraries and dependencies in a consistent manner for NWP applications. It is part of the NCEPLIBS project and is model/system agnostic. The HPC-Stack was originally written to facilitate installation of third-party libraries in a systematic manner on macOS and Linux systems (specifically RHEL). It was later transferred, expanded and further enhanced in the Joint Effort for Data assimilation Integration (JEDI) project.

1.2 Instructions

Level 1 platforms (e.g., Cheyenne, Hera) already have the HPC-Stack installed. Users on those platforms do *not* need to install the HPC-Stack before building applications or models that require the HPC-Stack. Users working on systems that fall under Support Levels 2-4 will need to install the HPC-Stack the first time they try to run applications or models that depend on it.

Users can either build the HPC-Stack on their local system or use the centrally maintained stacks on each HPC platform. For a detailed description of installation options, see *Installing the HPC-Stack*.

1.3 Disclaimer

The United States Department of Commerce (DOC) GitHub project code is provided on an "as is" basis and the user assumes responsibility for its use. DOC has relinquished control of the information and no longer has responsibility to protect the integrity, confidentiality, or availability of the information. Any claims against the Department of Commerce stemming from the use of its GitHub project will be governed by all applicable Federal law. Any reference to specific commercial products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply their endorsement, recommendation or favoring by the Department of Commerce. The Department of Commerce seal and logo, or the seal and logo of a DOC bureau, shall

not be used in any manner to imply endorsement of any commercial product or activity by DOC or the United States Government.

CHAPTER

TWO

INSTALL AND BUILD THE HPC-STACK

Attention: The HPC-Stack is already installed on Level 1 systems (e.g., Cheyenne, Hera, Orion). Installation is not necessary.

HPC-Stack installation will vary from system to system because there are so many possible combinations of operating systems, compilers, MPI's, and package versions. Installation via an EPIC-provided container is recommended to reduce this variability. However, users may choose a non-container approach to installation if they prefer.

Note: MPI stands for Message Passing Interface. An MPI is a standardized communication system used in parallel programming. It establishes portable and efficient syntax for the exchange of messages and data between multiple processors that are used by a single computer program. An MPI is required for high-performance computing (HPC).

2.1 Install and Build the HPC-Stack in a Singularity Container

The Earth Prediction Innovation Center (EPIC) provides several containers available for the installation of the HPC-Stack either individually or combined with Unified Forecast System (UFS) applications:

- docker://noaaepic/ubuntu20.04-gnu9.3
- docker://noaaepic/ubuntu20.04-hpc-stack
- docker://noaaepic/ubuntu20.04-epic-srwapp
- docker://noaaepic/ubuntu20.04-epic-mrwapp

2.1.1 Install Singularity

To install the HPC-Stack via Singularity container, first install the Singularity package according to the Singularity Installation Guide. This will include the installation of dependencies and the installation of the Go programming language. SingularityCE Version 3.7 or above is recommended.

Warning: Docker containers can only be run with root privileges, and users cannot have root privileges on HPC's. Therefore, it is not possible to build the HPC-Stack inside a Docker container on an HPC system. A Docker image may be pulled, but it must be run inside a container such as Singularity. Docker can, however, be used to build the HPC-Stack on a *local* system.

2.1.2 Build and Run the Container

1. Pull and build the container.

```
singularity pull ubuntu20.04-gnu9.3.sif docker://noaaepic/ubuntu20.04-gnu9.3
singularity build --sandbox ubuntu20.04-gnu9.3 ubuntu20.04-gnu9.3.sif
cd ubuntu20.04-gnu9.3
```

   Make a directory (e.g., `contrib`) in the container if one does not exist:

```
mkdir contrib
cd ..
```

2. From the local working directory, start the container and run an interactive shell within it. This command also binds the local working directory to the container so that data can be shared between them.

```
singularity shell -e --writable --bind /<local_dir>:/contrib ubuntu20.04-gnu9.3
```

   Make sure to update `<local_dir>` with the name of your local working directory.

2.1.3 Build the HPC-Stack

1. Clone the HPC-Stack repository (from inside the Singularity shell initialized above).

```
git clone https://github.com/NOAA-EMC/hpc-stack
cd hpc-stack
```

2. Set up the build environment. Be sure to change the `prefix` argument in the code below to your system's install location (likely within the `hpc-stack` directory).

```
./setup_modules.sh -p <prefix> -c config/config_custom.sh
```

Here, <prefix> is the directory where the software packages will be installed with a default value of $HOME/opt. For example, if the HPC-Stack is installed in the user's directory, the prefix might be /home/$USER/hpc-stack/hpc-modules.

Enter YES/YES/YES when the option is presented. Then modify build_stack.sh with the following commands:

```
sed -i "10 a source /usr/share/lmod/6.6/init/bash" ./build_stack.sh
sed -i "10 a export PATH=/usr/local/sbin:/usr/local/bin:$PATH" ./build_stack.sh
sed -i "10 a export LD_LIBRARY_PATH=/usr/local/lib64:/usr/local/lib:$LD_LIBRARY_
↪PATH" ./build_stack.sh
```

3. Build the environment. This may take several hours to complete.

```
./build_stack.sh -p <prefix> -c config/config_custom.sh -y stack/stack_custom.
↪yaml -m
```

4. Load the required modules, making sure to change the <prefix> to the location of the module files.

```
source /usr/share/lmod/lmod/init/bash
module use <prefix>/hpc-modules/modulefiles/stack
module load hpc hpc-gnu hpc-openmpi
module avail
```

From here, the user can continue to install and run applications that depend on the HPC-Stack, such as the UFS Short Range Weather (SRW) Application.

2.2 Non-Container HPC-Stack Installation and Build (General/Linux)

2.2.1 Install Prerequisites

To install the HPC-Stack locally, the following pre-requisites must be installed:

- **Python 3:** Can be obtained either from the main distributor or from Anaconda.
- **Compilers:** Distributions of Fortran, C, and C++ compilers that work for your system.
- **Message Passing Interface (MPI)** libraries for multi-processor and multi-core communications, configured to work with your corresponding Fortran, C, and C++ compilers.
- **Programs and software packages:** Lmod, CMake, make, wget, curl, git, and the TIFF library.

Note: For detailed instructions on how to build the HPC-Stack on two particular configurations of MacOS, see Chapter 3

To determine whether these prerequisites are installed, query the environment variables (for Lmod) or the location and version of the packages (for cmake, make, wget, curl, git). For example:

```
echo $LMOD_PKG
which cmake
cmake --version
```

Methods for determining whether `libtiff` is installed vary between systems. Users can try the following approaches:

```
whereis libtiff
locate libtiff
ldconfig -p | grep libtiff
ls /usr/lib64/libtiff*
ls /usr/lib/libtiff*
```

If compilers or MPI's need to be installed, consult the *HPC-Stack Prerequisites* document for further guidance.

2.2.2 Configure the Build

Choose the COMPILER, MPI, and PYTHON version, and specify any other aspects of the build that you would like. For Level 1 systems, a default configuration can be found in the applicable `config/config_<platform>.sh` file. For Level 2-4 systems, selections can be made by editing the `config/config_custom.sh` file to reflect the appropriate compiler, MPI, and Python choices for your system. If Lmod is installed on your system, you can view package options using the `module avail` command.

Some of the parameter settings available are:

- HPC_COMPILER: This defines the vendor and version of the compiler you wish to use for this build. The format is the same as what you would typically use in a `module load` command. For example, `HPC_COMPILER=intel/2020`. Use `gcc -v` to determine your compiler and version.
- HPC_MPI: This is the MPI library you wish to use. The format is the same as for HPC_COMPILER. For example: `HPC_MPI=impi/2020`.
- HPC_PYTHON: This is the Python interpreter to use for the build. The format is the same as for HPC_COMPILER, for example: `HPC_PYTHON=python/3.7.5`. Use `python --version` to determine the current version of Python.

Other variables include USE_SUDO, DOWNLOAD_ONLY, NOTE, PKGDIR, LOGDIR, OVERWRITE, NTHREADS, MAKE_CHECK, MAKE_VERBOSE, and VENVTYPE. For more information on their use, see *HPC-Stack Parameters*.

Note: If you only want to install select components of the HPC-Stack, you can edit the `stack/stack_custom.yaml` file to omit unwanted components. The `stack/stack_custom.yaml` file lists the software packages to be built along with their version, options, compiler flags, and any other package-specific options. A full listing of components is available in the *HPC-Stack Components* section.

2.2.3 Set Up Compiler, MPI, Python & Module System

Note: This step is required if you are using `Lmod` modules for managing the software stack. Lmod is installed across all Level 1 and Level 2 systems and in the containers provided. If `LMod` is not desired or used, the user can skip ahead to Step 2.2.4.

After preparing the system configuration in `./config/config_<platform>.sh`, run the following command from the top directory:

```
./setup_modules.sh -p <prefix> -c <configuration>
```

where:

`<prefix>` is the directory where the software packages will be installed during the HPC-Stack build. The default value is $HOME/opt. The software installation trees will branch directly off of `<prefix>`, while the module files will be located in the `<prefix>/modulefiles` subdirectory.

Attention: Note that `<prefix>` requires an absolute path; it will not work with a relative path.

`<configuration>` points to the configuration script that you wish to use, as described in Step 2.2.2. The default configuration file is `config/config_custom.sh`.

Additional Options:

The compiler and MPI modules can be handled separately from the rest of the build in order to exploit site-specific installations that maximize performance. In this case, the compiler and MPI modules are preceded by an `hpc-` label. For example, to load the Intel compiler module and the Intel MPI (IMPI) software library, enter:

```
module load hpc-intel/2020
module load hpc-impi/2020
```

These `hpc-` modules are really meta-modules that load the compiler/MPI library and modify the MODULEPATH so that the user has access to the software packages that will be built in Step 2.2.4. On HPC systems, these meta-modules load the native modules provided by the system administrators.

In short, you may prefer not to load the compiler or MPI modules directly. Instead, loading the hpc-meta-modules as demonstrated above will provide everything needed to load software libraries.

It may be necessary to set certain source and path variables in the `build_stack.sh` script. For example:

```
source /usr/share/lmod/6.6/init/bash
source /usr/share/lmod/lmod/init/bash
export PATH=/usr/local/sbin:/usr/local/bin:$PATH
```

(continues on next page)

(continued from previous page)

```
export LD_LIBRARY_PATH=/usr/local/lib64:/usr/local/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH
```

It may also be necessary to initialize `Lmod` when using a user-specific `Lmod` installation:

```
module purge
export BASH_ENV=$HOME/<Lmod-installation-dir>/lmod/lmod/init/bash
source $BASH_ENV
export LMOD_SYSTEM_DEFAULT_MODULES=<module1>:<module2>:<module3>
module --initial_load --no_redirect restore
module use <$HOME>/<your-modulefiles-dir>
```

where:

- `<Lmod-installation-dir>` is the top directory where Lmod is installed
- `<module1>, ...,<moduleN>` is a comma-separated list of modules to load by default
- `<$HOME>/<your-modulefiles-dir>` is the directory where additional custom modules may be built with Lmod (e.g., $HOME/modulefiles).

2.2.4 Build the HPC-Stack

Now all that remains is to build the stack:

```
./build_stack.sh -p <prefix> -c <configuration> -y <yaml_file> -m
```

Here the -m option is only required when you need to build your own modules *and* LMod is used for managing the software stack. It should be omitted otherwise. `<prefix>` and `<configuration>` are the same as in Step 2.2.3, namely a reference to the absolute-path installation prefix and a corresponding configuration file in the `config` directory. As in Step 2.2.3, if this argument is omitted, the default is to use `$HOME/opt` and `config/config_custom.sh` respectively. `<yaml_file>` represents a user configurable yaml file containing a list of packages that need to be built in the stack along with their versions and package options. The default value of `<yaml_file>` is `stack/stack_custom.yaml`.

Warning: Steps Step 2.2.2, Step 2.2.3, and Step 2.2.4 need to be repeated for each compiler/MPI combination that you wish to install. The new packages will be installed alongside any previously-existing packages that may already have been built from other compiler/MPI combinations.

From here, the user can continue to install and run applications that depend on the HPC-Stack.

CHAPTER

THREE

INSTALL AND BUILD HPC-STACK ON MACOS

HPC-Stack can be installed and built on MacOS systems. The following two options have been tested:

- **Option 1:** MacBookAir 2020, M1 chip (arm64, running natively), 4+4 cores, Big Sur 11.6.4, GNU compiler suite v.11.2.0_3 (gcc, gfortran, g++); no MPI pre-installed
- **Option 2:** MacBook Pro 2015, 2.8 GHz Quad-Core Intel Core i7 (x86_64), Catalina OS X 10.15.7, GNU compiler suite v.11.2.0_3 (gcc, gfortran, g++); no MPI pre-installed

Note: Examples throughout this chapter presume that the user is running Terminal.app with a bash shell environment. If this is not the case, users will need to adjust commands to fit their command line application and shell environment.

3.1 Prerequisites for Building HPC-Stack

3.1.1 Install Homebrew and Xcode Command-Line Tools (CLT)

Open Terminal.app and a web browser. Go to https://brew.sh, copy the command-line installation directive, and run it in a new Terminal window. Terminal will request a sudo access password. The installation command will look similar to the following:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
↪install.sh)"
```

This will install Homebrew, Xcode CLT, and Ruby.

An alternative way to install the Xcode command-line tools (CLT) is as follows:

```
xcode-select --install
```

3.1.2 Install Compilers

Install GNU compiler suite (version 11) and gfortran:

```
brew install gcc@11
```

Create symbolic links from the version-specific binaries to gcc and g++. A sudo password may be requested. The path will likely be /opt/homebrew/bin/gcc-11 (Option 1), or /usr/local/bin/gcc-11 (Option 2).

```
which gcc-11
cd /usr/local/bin/        (OR cd /opt/homebrew/bin/ )
ln -s gcc-11 gcc
ln -s g++-11 g++
```

There is no need to create a link for gfortran if this is the first installation of this compiler. If an earlier version of gfortran exists, you may rename it (e.g., to "gfortran-old") and create a link to the new installation:

```
ln -s gfortran-11 gfortran
```

Verify the paths for the compiler binaries:

```
which gcc
which g++
which gfortran
```

Verify that they show the correct version of GNU installed:

```
gcc --version
g++ --version
gfortran --version
```

3.1.3 Install CMake

Install the cmake utility via homebrew:

```
brew install cmake
```

3.1.4 Install/Upgrade Make

To install the make utility via homebrew:

```
brew install make
```

To upgrade the make utility via homebrew:

```
brew upgrade make
```

3.1.5 Install Openssl@3

To install the openssl@3 package, run:

```
brew install openssl@3
```

Note the messages at the end of the installation. Depending on what they say, users may need to add the location of the openssl@3 binaries to the environment variable `$PATH`. To add it to the `PATH`, run:

```
echo 'export PATH="/opt/homebrew/opt/openssl@3/bin:$PATH"' >> ~/.bashrc
```

Users may also need to set certain flags so that the compilers can find the openssl@3 package:

```
export LDFLAGS="-L/opt/homebrew/opt/openssl@3/lib"
export CPPFLAGS="-I/opt/homebrew/opt/openssl@3/include"
```

3.1.6 Install Lmod

Install Lmod, which is the module management environment, run:

```
brew install lmod
```

You may need to add the Lmod environment initialization to your shell profile, e.g., to `$HOME/.bashrc`.

For the Option 1 installation, add:

```
export BASH_ENV="/opt/homebrew/opt/lmod/init/bash"
source $BASH_ENV
```

For the Option 2 installation, add:

```
export BASH_ENV="/usr/local/opt/lmod/init/bash"
source $BASH_ENV
```

3.1.7 Install libpng

The libpng library has issues when building on MacOS during the HPC-Stack bundle build. Therefore, it must be installed separately. To install the libpng library, run:

```
brew install libpng
```

3.1.8 Install wget

Install the Wget software package:

```
brew install wget
```

3.1.9 Install or Update Python3

First, verify that Python3 is installed, and check the current version:

```
which python3
python3 --version
```

The first command should return `/usr/bin/python3` and the second should return `Python 3.8.2` or similar (the exact version is unimportant).

If necessary, download an updated version of Python3 for MacOS from https://www.python.org/downloads. The version 3.9.11 64-bit universal2 installer package is recommended (i.e., `python-3.9.11-macos11.pkg`). Double-click on the installer package, and accept the license terms. An administrative level password will be requested for the installation. At the end of the installation, run `Install Certificates.command` by double-clicking on the shell script in Finder.app that opens and runs it.

Start a new bash session (type `bash` in the existing terminal), and verify the installed version:

```
python3 --version
```

The output should now correspond to the Python version you installed.

3.1.10 Install Git

Install git and dependencies:

```
brew install git
```

3.2 Building HPC-Stack

3.2.1 Clone HPC-Stack

Download HPC-Stack code from GitHub:

```
git clone https://github.com/NOAA-EMC/hpc-stack.git
cd hpc-stack
```

The configuration files are `./config/config_<machine>.sh`. For Option 1, `<machine>` is `mac_m1_gnu` and for Option 2, `<machine>` is `mac_gnu`.

The `./stack/stack_<machine>.yaml` file lists the libraries that will be built as part of HPC-Stack, in addition to library-specific options. These can be altered based on user preferences.

3.2.2 Lmod Environment

Verify the initialization of Lmod environment, or add it to the configuration file `./config/config_<machine>.sh`, as in Step 3.1.6.

For Option 1:

```
export BASH_ENV="/opt/homebrew/opt/lmod/init/profile"
source $BASH_ENV
```

For Option 2:

```
export BASH_ENV="/usr/local/opt/lmod/init/profile"
source $BASH_ENV
```

3.2.3 Specify Compiler, Python, and MPI

Specify the combination of compilers, python libraries, and MPI libraries in the configuration file `./config/config_<machine>.sh`.

```
export HPC_COMPILER="gnu/11.2.0_3"
export HPC_MPI="openmpi/4.1.2"      (Option 1 only)
export HPC_MPI="mpich/3.3.2"        (Option 2 only)
export HPC_PYTHON="python/3.10.2"
```

Comment out any export statements not relevant to the system, and make sure that version numbers reflect the versions installed on the system (which may differ from the versions listed here).

3.2.4 Set Appropriate Flags

When using gfortran version 10 or higher, verify that the following flags are set in `config_<machine>.sh`:

For Option 1:

```
export STACK_FFLAGS="-fallow-argument-mismatch -fallow-invalid-boz"
```

For Option 2:

```
export STACK_FFLAGS="-fallow-argument-mismatch -fallow-invalid-boz"
export STACK_CXXFLAGS="-march=native"
```

3.2.5 Set Environment Variables

Set the environmental variables for compiler paths in `./config/config_<machine>.sh`. The variable `GNU` below refers to the directory where the compiler binaries are located. For example, with Option 1, `GNU=/opt/homebrew/bin/gcc`, and with Option 2: `GNU=/usr/local/bin`.

```
export GNU="path/to/compiler/binaries"
export CC=$GNU/gcc
export FC=$GNU/gfortran
export CXX=$GNU/g++
export SERIAL_CC=$GNU/gcc
export SERIAL_FC=$GNU/gfortran
export SERIAL_CXX=$GNU/g++
```

3.2.6 Specify MPI Libraries

Specify the MPI libraries to be built within the HPC-Stack in `./stack/stack_<machine>.yaml`. The `openmpi/4.1.2` (Option 1) and `mpich/3.3.2` (Option 2) have been built successfully.

Option 1:

```
mpi:
build: YES
flavor: openmpi
version: 4.1.2
```

Option 2:

```
mpi:
build: YES
flavor: mpich
version: 3.3.2
```

3.2.7 Libpng

Set build `libpng` library to NO in `./stack/stack_<machine>.yaml` to avoid problems during the HPC-Stack build. Leave the defaults for other libraries and versions in `./stack/stack_<machine>.yaml`.

```
libpng:
build: NO
```

3.2.8 Set Up the Modules and Environment

Set up the modules and environment:

```
./setup_modules.sh -c config/config_<machine>.sh -p $HPC_INSTALL_DIR | tee setup_
↪modules.log
```

where `<machine>` is `mac_m1_gnu` (Option 1), or `mac_gnu` (Option 2), and `$HPC_INSTALL_DIR` is the *absolute* path for the installation directory of the HPC-Stack. You will be asked to choose whether or not to use "native" installations of Python, the compilers, and the MPI. "Native" means that they are already installed on your system. Thus, you answer "YES" to python, "YES" to gnu compilers, and "NO" for MPI/mpich.

3.2.9 Building HPC-Stack

Build the modules:

```
./build_stack.sh -c config/config_<machine>.sh -p $HPC_INSTALL_DIR  -y stack/stack_
↪<machine>.yaml -m 2>&1 | tee build_stack.log
```

Attention:

- The option `-p` requires an absolute path (full path) of the installation directory!
- The `-m` option is needed to build separate modules for each library package.

CHAPTER

FOUR

INSTALLATION OF THE HPC-STACK PREREQUISITES

A wide variety of compiler and MPI options are available. Certain combinations may play well together, whereas others may not.

The following system, compiler, and MPI combinations have been tested successfully:

Table 4.1: Sample System, Compiler, and MPI Options

System	Compilers	MPI
SUSE Linux Enterprise Server 12.4	Intel compilers 2020.0 (ifort, icc, icps)	Intel MPI wrappers (mpif90, mpicc, mpicxx)
Linux CentOS 7	Intel compilers 2020.0 (ifort, icc, icps)	Intel MPI (mpiifort, mpiicc, mpiicpc)

Compilers and MPI libraries can be downloaded from the following websites:

Compilers:

- GNU/GCC (version 9.x)
- Intel

MPI's

- OpenMPI
- MPICH
- IntelMPI

CHAPTER

FIVE

BUILD PARAMETERS

5.1 Compiler & MPI

HPC_COMPILER:
This defines the vendor and version of the compiler you wish to use for this build. The format is the same as what you would typically use in a module load command. For example, `HPC_COMPILER=intel/2020`. Options include:

- `gnu/6.5.0`
- `gnu/9.2.0`
- `intel/18.0.5.274`
- `intel/19.0.5.281`
- `intel/2020`
- `intel/2020.2`
- `intel/2021.3.0`

For information on setting compiler flags, see Section 7.1 Additional Notes.

HPC_MPI:
The MPI library you wish to use for this build. The format is the same as for HPC_COMPILER; for example: `HPC_MPI=impi/2020`. Current MPI types accepted are openmpi, mpich, impi, cray, and cray*. Options include:

- `impi/2020`
- `impi/2018.4.274`
- `impi/2019.0.5`
- `impi/2020`
- `impi/2020.2`
- `impi/2021.3.0`
- `mvapich2/2.3`
- `openmpi/4.1.2`

Note: For example, when using Intel-based compilers and Intel's implementation of the MPI interface, the `config/config_custom.sh` should contain the following specifications:

```
export SERIAL_CC=icc
export SERIAL_FC=ifort
export SERIAL_CXX=icpc

export MPI_CC=mpiicc
export MPI_FC=mpiifort
export MPI_CXX=mpiicpc
```

This will set the C, Fortran, and C++ compilers and MPI's.

Note: To verify that your chosen MPI build (e.g., mpiicc) is based on the corresponding serial compiler (e.g., icc), use the `-show` option to query the MPI's. For example,

```
mpiicc -show
```

will display output like this:

```
$ icc -I<LONG_INCLUDE_PATH_FOR_MPI> -L<ANOTHER_MPI_LIBRARY_PATH> -L<ANOTHER_MPI_
↪PATH> -<libraries, liners, build options...> -X<something> --<enable/disable/
↪with some options> -l<library> -l<another_library> -l<yet-another-library>
```

The message you need from this prompt is "icc", which confirms that your mpiicc build is based on icc. It may happen that if you query the "mpicc -show" on your system, it is based on "gcc" (or something else).

5.2 Other Parameters

HPC_PYTHON:
The Python interpretor you wish to use for this build. The format is the same as for `HPC_COMPILER`, for example: `HPC_PYTHON=python/3.7.5`.

USE_SUDO:
If the directory where the software packages will be installed (`$PREFIX`) requires root permission to write to, such as `/opt/modules`, then this flag should be enabled. For example, `USE_SUDO=Y`.

DOWNLOAD_ONLY:
The stack allows the option to download the source code for all the software without performing the installation. This is especially useful for installing the stack on machines that do not allow internet connectivity to websites hosting the software (e.g., GitHub). For more information, see Section 7.4 Additional Notes.

Note: To enable a boolean flag, use a single-digit `Y` or `T`. To disable, use `N` or `F` (case insensitive).

PKGDIR:
is the directory where tarred or zipped software files will be downloaded and compiled. Unlike `$PREFIX`, this is a relative path based on the root path of the repository. Individual software packages can be downloaded manually to this directory and untarred, but this is not required. Build scripts will look for the directory `pkg/<pkgName-pkgVersion>` (e.g., `pkg/hdf5-1_10_3`).

LOGDIR:
The directory where log files from the build will be written, relative to the root path of the repository.

OVERWRITE:
If set to `T`, this flag will cause the build script to remove the current installation, if any exists, and replace it with the new version of each software package in question. If this variable is not set, the build will bypass software packages that are already installed.

NTHREADS:
The number of threads to use for parallel builds.

MAKE_CHECK:
Run make check after build.

MAKE_VERBOSE:
Print out extra information to the log files during the build.

VENVTYPE:
Set the type of python environment to build. Value depends on whether using pip or conda. Set `VENVTYPE=pyvenv` when using pip and `VENVTYPE=condaenv` when using Miniconda for creating virtual environments. Default is `pyvenv`.

CHAPTER

SIX

HPC-STACK COMPONENTS

The HPC-Stack packages are built in Step 2.2.4 using the `build_stack.sh` script. The following software can optionally be built with the scripts under `libs`.

- **Compilers and MPI libraries**
  - GNU/GCC
  - Intel
  - OpenMPI
  - MPICH
  - `hpc-` Meta-modules for all the above as well as Intel and IMPI
- **HPC Stack - Third Party Libraries**
  - CMake
  - Udunits
  - PNG
  - JPEG
  - Jasper
  - SZip
  - Zlib
  - HDF5
  - PNetCDF
  - NetCDF
  - ParallelIO
  - nccmp
  - nco
  - CDO
  - FFTW

  - GPTL
  - Tau2
  - Boost
  - Eigen
  - GSL-Lite
  - JSON for C++
  - JSON Schema Validator for C++
  - pybind11
  - MADIS
  - SQLite
  - PROJ
  - GEOS
- **UFS Dependencies**
  - ESMF
  - FMS
- **NCEP Libraries**
  - NCEPLIBS-bacio
  - NCEPLIBS-sigio
  - NCEPLIBS-sfcio
  - NCEPLIBS-gfsio
  - NCEPLIBS-w3nco
  - NCEPLIBS-sp
  - NCEPLIBS-ip
  - NCEPLIBS-ip2
  - NCEPLIBS-g2
  - NCEPLIBS-g2c
  - NCEPLIBS-g2tmpl
  - NCEPLIBS-nemsio
  - NCEPLIBS-nemsiogfs
  - NCEPLIBS-w3emc
  - NCEPLIBS-landsfcutil
  - NCEPLIBS-bufr

  - NCEPLIBS-wgrib2
  - NCEPLIBS-prod_util
  - NCEPLIBS-grib_util
  - NCEPLIBS-ncio
  - NCEPLIBS-wrf_io
  - EMC_crtm
  - UPP
- **JEDI Dependencies**
  - ecbuild
  - eckit
  - fckit
  - atlas
- **Python and Virtual Environments**
  - Miniconda3
  - r2d2

CHAPTER

SEVEN

HPC-STACK ADDITIONAL NOTES

7.1 Setting Compiler Flags and Other Options

Often it is necessary to specify compiler flags (e.g., `gfortran-10 -fallow-argument-mismatch` for the packages via `FFLAGS`. There are 2 ways this can be achieved:

1. **For all packages:** One can define variable e.g., `STACK_FFLAGS=-fallow-argument-mismatch` in the config file `config_custom.sh`. This will append `STACK_FFLAGS` to `FFLAGS` in every build script under libs.
2. **Package specific flags:** To compile only the specific package under `libs` with the above compiler flag, one can define variable `FFLAGS=-fallow-argument-mismatch` in the `<package>` section of the YAML file `stack_custom.yaml`. This will append `STACK_<package>_FFLAGS` to `FFLAGS` in the build script for that package only.

7.2 Adding a New Library or Package

If you want to add a new library to the stack you need to follow these steps:

1. Write a new build script in `libs`, using existing scripts as a template.
2. Define a new section in the `yaml` file for that library/package in `config` directory.
3. If the package is a python virtual environment, add a `requirements.txt` or `environment.yml` file listing the python packages required to install the package. These files should be named and placed in `pyvenv/package_name.txt` and `pyvenv/package_name.yml`. `VENVTYPE=pyvenv` will use the `pyvenv/package_name.txt` and `VENVTYPE=condaenv` will use `pyvenv/package_name.yml`.
4. Add a call to the new build script in `build_stack.sh`.
5. Create a new module template at the appropriate place in the modulefiles directory, using exising files as a template.
6. Update the *HPC Components* file to include the name of the new library or package.

7.3 Configuring for a new HPC

If you want to port this to a new HPC, you need to follow these steps:

1. Write a new config file `config/config_<hpc>.sh`, using existing config files as a template. Also create a new yaml file `config/stack_<hpc>.yaml`, using existing yaml files as a template.
2. Add/remove basic modules for that HPC.
3. Choose the appropriate Compiler/MPI combination.
4. If a template modulefile does not exist for that Compiler/MPI combinattion, create module templates at the appropriate place in the `modulefiles` directory, using existing files as a template (e.g., `hpc-ips` or `hpc-smpi`).
5. If the new HPC system provides some basic modules for e.g., Git, CMake, etc., they can be loaded in `config/config_<hpc>.sh`.

7.4 Using the DOWNLOAD_ONLY Option

If an HPC (e.g., NOAA RDHPCS Hera) does not allow access to online software via `wget` or `git clone`, you will have to download all the packages using the `DOWNLOAD_ONLY` option in the `config_custom.sh`. Execute `build_stack.sh` as you would on a machine that does allow access to online software with `DOWNLOAD_ONLY=YES` and all the packages will be downloaded in the `pkg` directory. Transfer the contents of the `pkg` directory to the machine where you wish to install the HPC-Stack, and execute `build_stack.sh`. The `build_stack.sh` script will detect the already-downloaded packages and use them rather than fetching them.

7.5 Using the HPC-Stack

- If Lmod is used to manage the software stack, you will need to activate the HPC-Stack in order to use it. This is done by loading the `hpc` module under `$PREFIX/modulefiles/stack` as follows:

```
module use $PREFIX/modulefiles/stack
module load hpc/1.0.0
```

This will put the `hpc-<compilerName>` module in your `MODULEPATH`, which can be loaded as:

```
module load hpc-<compilerName>/<compilerVersion>
```

- If the HPC-Stack is not managed via modules, you need to add `$PREFIX` to the PATH as follows:

```
export PATH="$PREFIX/bin:$PATH"
export LD_LIBRARY_PATH="$PREFIX/lib:$LD_LIBRARY_PATH"
export CMAKE_PREFIX_PATH="$PREFIX"
```

7.6 Known Workaround for Certain Installations of Lmod

- On some machines (e.g., WCOSS_DELL_P3), LMod is built to disable loading of default modulefiles and requires the user to load the module with an explicit version of the module (e.g., `module load netcdf/4.7.4` instead of `module load netcdf`). The latter looks for the `default` module which is either the latest version or a version that is marked as default. To circumvent this, it is necessary to place the following lines in `modulefiles/stack/hpc/hpc.lua` prior to executing `setup_modules.sh` or in `$PREFIX/modulefiles/stack/hpc/1.0.0.lua` after executing `setup_modules.sh`.

```
setenv("LMOD_EXACT_MATCH", "no")
setenv("LMOD_EXTENDED_DEFAULT", "yes")
```

  See more on the Lmod website.

7.7 Known Issues

- NetCDF-C++ does not build with LLVM Clang. It can be disabled by setting `disable_cxx: YES` in the stack file under the NetCDF section.
- Json-schema-validator does not build with LLVM Clang. It can be disabled in the stack file in the json-schema-validator-section.