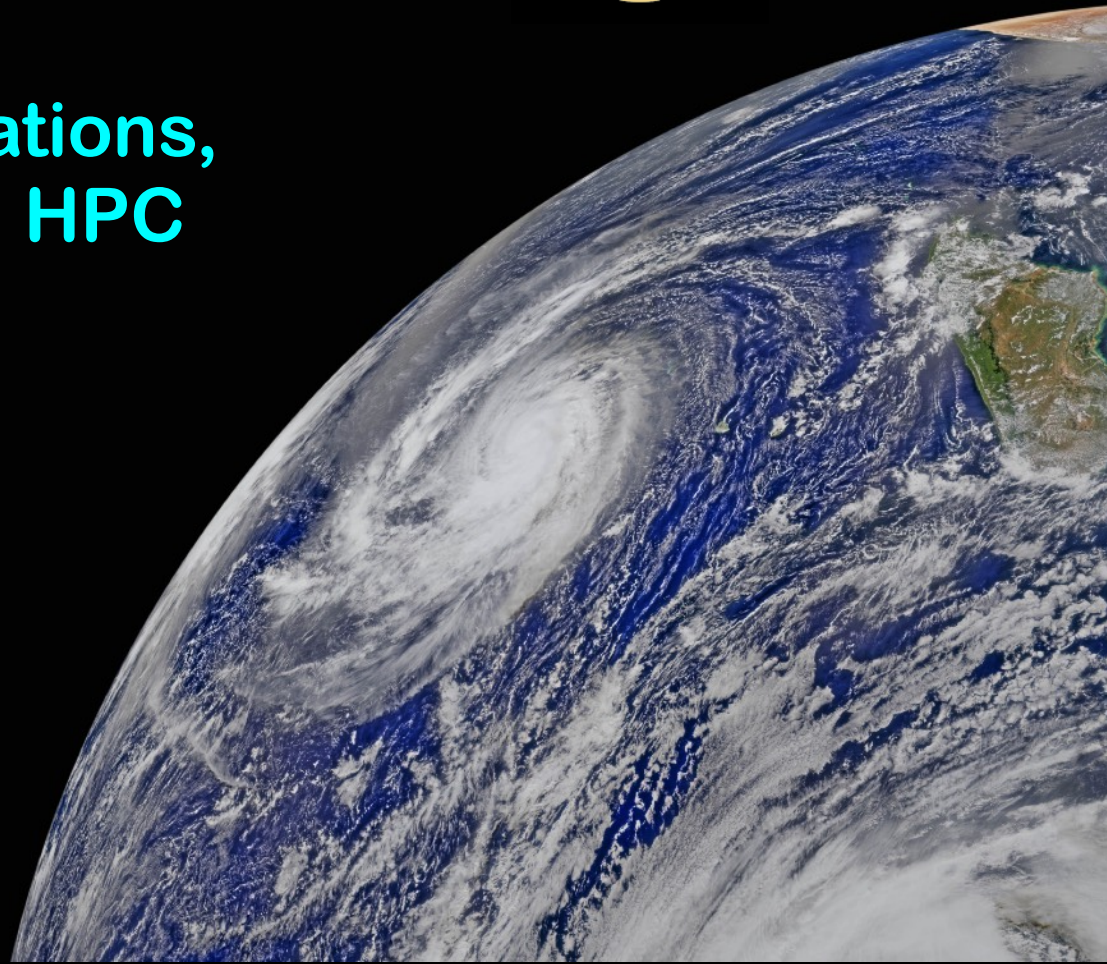
Building and Running JEDI

Laptops, Workstations, Clusters, Cloud, HPC



Outline

I) Acquire dependencies

- ✦ JEDI Portability overview
- ✦ Software containers
- ✦ HPC environment modules
- ✦ Cloud

II) Build JEDI

- ✦ JEDI bundles
- ✦ CMake, ecbuild

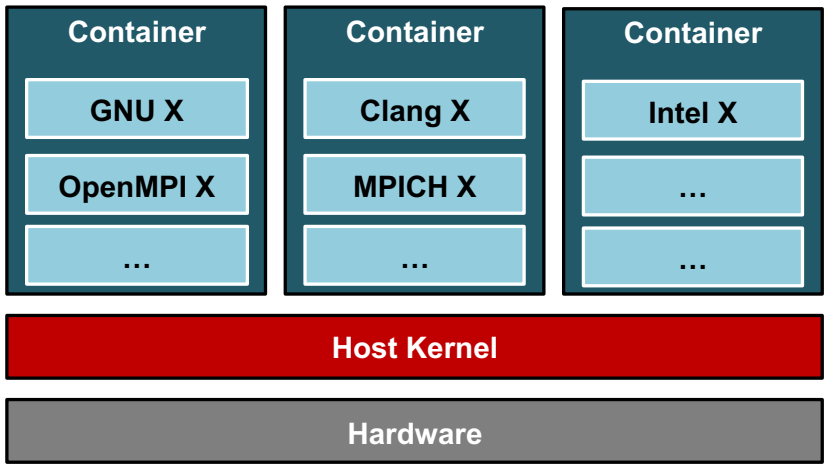
How can I Run JEDI?

- **Application container**
  - ✦ A software container that includes JEDI and all it's dependencies, ready to run
- **Development container**
  - ✦ Includes JEDI dependencies - you download and build JEDI yourself
- **Pre-Made Environment Modules**
  - ✦ JEDI dependencies available on Hera, Orion, Discover, S4, Cheyenne, Gaffney, and the Amazon cloud (through AMIs)
  - ✦ You download and build JEDI yourself
- **Build your own Environment Modules**
  - ✦ Jedi-stack build system: https://github.com/JCSDA/jedi-stack
  - ✦ You build JEDI and all of its dependencies

What is a container?

Software container (working definition)

A packaged user environment that can be "unpacked" and used across different systems, from laptops to cloud to HPC



Container benefits

- Portability
- Reproducibility
  - Version control (git)
- Bring your own environment
- Efficiency / workflow
  - Develop on laptops, run on HPC/cloud
  - Get new users up and running quickly

JEDI Software Dependencies

- **Essential**
  - Compilers, MPI
  - CMake
  - SZIP, ZLIB
  - LAPACK / MKL, Eigen 3
  - NetCDF4, HDF5
  - udunits
  - Boost (headers only)
  - ecbuild, eckit, fckit
  - bufr
- **Useful**
  - PNETCDF
  - Parallel IO
  - nccmp, NCO
  - Python tools (netcdf4, matplotlib, cartopy…)
  - json-schema-validator

What do the containers and modules contain?

Common versions among users and developers minimize stack-related debugging

Environment Modules

Example: Discover (NCCS)

```
(base) mmiesch@discover34:~> module purge
(base) mmiesch@discover34:~> module load jedi/intel-impi
(base) mmiesch@discover34:~> module list

Currently Loaded Modules:
  1) git/2.24.0                9) udunits/2.2.26          17) eigen/3.3.7
  2) git-lfs/2.10.0           10) mpi/impi/19.1.0.166     18) bufrlib/11.3.2
  3) jedi-python/3.8.3        11) jedi-impi/19.1.0.166    19) cmake/3.17.0
  4) comp/gcc/9.2.0           12) hdf5/1.12.0             20) ecbuild/jcsda-3.3.2.jcsda3
  5) comp/intel/19.1.0.166    13) pnetcdf/1.12.1          21) eckit/jcsda-1.11.6.jcsda2
  6) jedi-intel/19.1.0.166    14) netcdf/4.7.4            22) nco/4.7.9
  7) szip/2.1.1               15) nccmp/1.8.7.0           23) pio/2.5.1-debug
  8) zlib/1.2.11              16) boost-headers/1.68.0    24) jedi/intel-impi/19.1.0.166-v0.4
```

jedi-stack leverages native compilers and mpi libraries
Other stack components are built from these

Container Technologies

Docker

- Main Advantages: industry standard, widely supported, runs on native Mac/Windows OS
- Main Disadvantange: Security (root privileges)

Charliecloud

- Main Advantages: Simplicity, no need for root privileges
- Main Disadvantages: Fewer features than Singularity, Relies on Docker (to build, not to run)

Singularity

- Main Advantages: Reproducibility, HPC support
- Main Disadvantage: Not available on all HPC systems
- Preferred platform for scientific applications

Current containers

- **Development**
  - gnu-openmpi-dev (D, S, C)
  - clang-mpich-dev (D, S, C)
  - intel-impi-dev (D, S, C)
- **Application**
  - Tutorial (S)
  - intel-impi-app (S ⇒ S)

Distribution

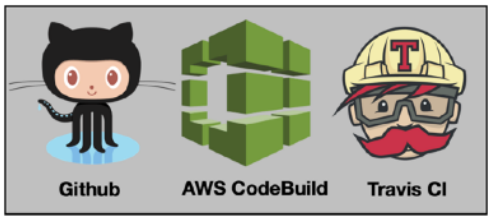
Docker Hub
Sylabs cloud
AWS S3 (public)
AWS S3 (private)

```
singularity pull library://jcsda/public/jedi-gnu-openmpi-dev
singularity shell -e jedi-gnu-openmpi_latest.sif
```

amazon web services

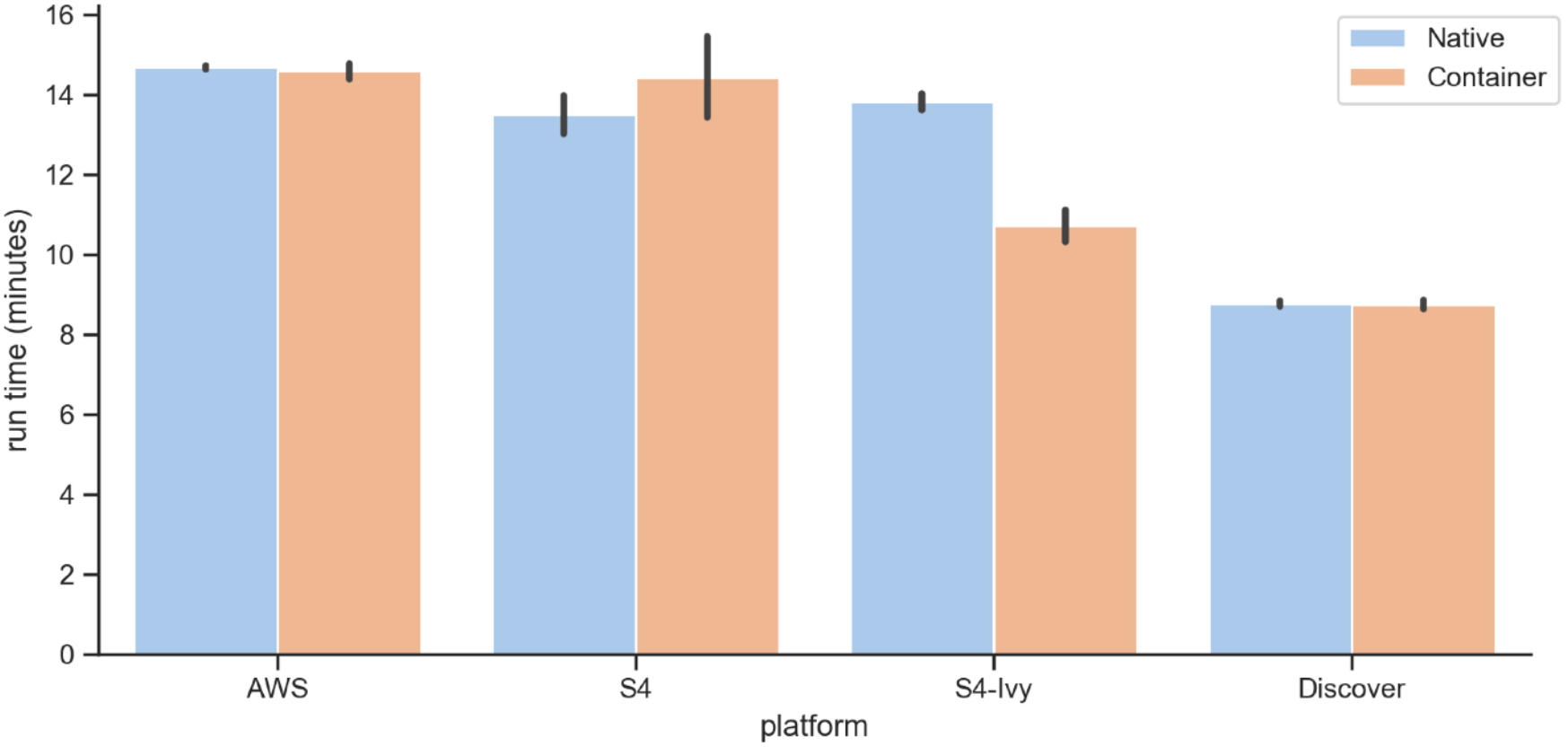
JCSDA provides a public ubuntu 18.04 AMI that comes with Singularity, Charliecloud, and Docker pre-installed

Unified Build System



Tagged jedi-stack releases can be used to build tagged containers, AMIs, and HPC environment modules, ensuring common software environments across platforms

Supercontainers!



With a little care, containers can be run across nodes on HPC systems with no overhead

JEDI 3DVar Application

864 MPI tasks, 12M observations
FV3-gfs c192

II: JEDI Build System

The JEDI is code organized into bundles that identify all the GitHub repositories necessary to build and run the applications

CMake build system: ecbuild = CMake macro package developed and maintained by ECMWF

Edit this file to select the branches you're working on (you may want to leave out the UPDATE option)

CMakeLists.txt file for fv3-bundle

```
# C
pro

[..

# E
ecb                                                    git"   TAG v2.3-jedi )

# Core JEDI repositories
ecbuild_bundle( PROJECT oops  GIT "https://github.com/jcsda/oops.git"  BRANCH develop UPDATE )
ecbuild_bundle( PROJECT saber GIT "https://github.com/jcsda/saber.git" BRANCH develop UPDATE )
ecbuild_bundle( PROJECT ioda  GIT "https://github.com/jcsda/ioda.git"  BRANCH develop UPDATE )
ecbuild_bundle( PROJECT ufo   GIT "https://github.com/jcsda/ufo.git"   BRANCH develop UPDATE )

# FMS and FV3 dynamical core
ecbuild_bundle( PROJECT fms GIT "https://github.com/jcsda/FMS.git"                    TAG 1.0.0.jcsda )
ecbuild_bundle( PROJECT fv3 GIT "https://github.com/jcsda/GFDL_atmos_cubed_sphere.git" TAG 1.0.0.jcsda )

# fv3-jedi and associated repositories
ecbuild_bundle( PROJECT femps       GIT "https://github.com/jcsda/femps.git"               BRANCH develop UPDATE )
ecbuild_bundle( PROJECT fv3-jedi-lm GIT "https://github.com/jcsda/fv3-jedi-linearmodel.git" BRANCH develop UPDATE )
ecbuild_bundle( PROJECT fv3-jedi    GIT "https://github.com/jcsda/fv3-jedi.git"            BRANCH develop UPDATE )
```

Building a Bundle

```
git clone https://github.com/JCSDA/fv3-bundle.git      1
mkdir build                                            2
cd build
ecbuild ../fv3-bundle                                  3
make update                                            4
make -j4                                               5
ctest                                                  6
```

1. Download the bundle repository from GitHub
2. Create a build directory
3. Run ecbuild (CMake) to generate a build system
4. Pull the latest source code from GitHub
5. Compile
6. Run the test suite for the bundle

See Maryam's lecture on Thursday for more details on the JEDI test suite and how to use ctest

ecbuild Usage

Debug build illustrates some of the useful options for ecbuild, make, and ctest

```
ecbuild --build=Debug ../fv3-bundle
make VERBOSE=1 -j4
export OOPS_TRACE=1
export OOPS_DEBUG=1
ctest -VV -R fv3jedi_test_tier1_letkf
```

Use release build for better performance

```
ecbuild --build=Release ../fv3-bundle
```

ecbuild Usage

The ecbuild executable is just a text file and you can view it directly for useful information

```
more `which ecbuild`
```

```
USAGE:

  ecbuild [--help] [--version] [--toolchains]
  ecbuild [option...] [--] [cmake-argument...] <path-to-source>
  ecbuild [option...] [--] [cmake-argument...] <path-to-existing-build>

DESCRIPTION:

  ecbuild is a build system based on CMake, but providing a lot of macro's
  to make it easier to work with. Upon execution,
  the equivalent cmake command is printed.

  ecbuild/cmake must be called from an out-of-source build directory and
  forbids in-source builds.

SYNOPSIS:

    --help         Display this help
    --version      Display ecbuild version
    --toolchains   Display list of pre-installed toolchains (see below)
```

```
Available values for "option":

    --cmakebin=<path>
          Set which cmake binary to use. Default is 'cmake'

    --prefix=<prefix>
          Set the install path to <prefix>.
          Equivalent to cmake argument "-DCMAKE_INSTALL_PREFIX=<prefix>"

    --build=<build-type>
          Set the build-type to <build-type>.
          Equivalent to cmake argument "-DCMAKE_BUILD_TYPE=<build-type>"
          <build-type> can be any of:
             - debug : Lowest optimization level, useful for debugging
             - release : Highest optimization level, for best performance
             - bit : Highest optimization level while staying bit-reproducible
             - ...others depending on project

    --log=<log-level>
          Set the ecbuild log-level
          Equivalent to "-DECBUILD_LOG_LEVEL=<log-level>"
          <log-level> can be any of:
             - DEBUG
             - INFO
             - WARN
             - ERROR
             - CRITICAL
             - OFF
          Every choice outputs also the log-levels listed below itself

    --static
          Build static libraries.
          Equivalent to "-DBUILD_SHARED_LIBS=OFF"

    --dynamic, --shared
          Build dynamic libraries (usually the default).
          Equivalent to "-DBUILD_SHARED_LIBS=ON"
```

Running a JEDI Application

Each application just takes a single configuration file as input, in yaml format

```
# Define JEDI bin directory where the executables are found
# ---------------------------------------------------------
export jedibin=$HOME/jedi/build/bin

# Run the BUMP parameter scripts to produce the B matrix
# ------------------------------------------------------
mpirun -np 6 $jedibin/fv3jedi_parameters.x config/bumpparameters_nicas_gfs.yaml

# Run the variational application
# -------------------------------
mpirun -np 18 $jedibin/fv3jedi_var.x config/4denvar.yaml

# Compute the increment for plotting
# ----------------------------------
mpirun -np 6 $jedibin/fv3jedi_diffstates.x config/4denvar-increment.yaml
```

A JEDI Configuration file

```yaml
cost function:
  cost type: 4D-Ens-Var
  analysis variables: [ua,va,T,ps,sphum,ice_wat,liq_wat,o3mr]
  window begin: '2018-04-14T21:00:00Z'
  window length: PT6H
  subwindow: PT3H
  background:
    states:
    - filetype: gfs
      datapath: /opt/jedi/build/fv3-jedi/test/Data/inputs/gfs_c12/bkg/
      filename_core: 20180414.210000.fv_core.res.nc
      filename_trcr: 20180414.210000.fv_tracer.res.nc
      filename_sfcd: 20180414.210000.sfc_data.nc
      filename_sfcw: 20180414.210000.fv_srf_wnd.res.nc
      filename_cplr: 20180414.210000.coupler.res
      state variables: [ua,va,T,ps,sphum,ice_wat,liq_wat,o3mr,phis,
                        slmsk,sheleg,tsea,vtype,stype,vfrac,stc,smc,snwdph,
                        u_srf,v_srf,f10m]
# [...]
observations:
  - obs space:
      name: AMSUA-NOAA19
      obsdatain:
        obsfile: /opt/jedi/build/fv3-jedi/test/Data/obs/testinput_tier_1/amsua_n19_obs_2018041500_m.nc4
      simulated variables: [brightness_temperature]
      channels: 10
    obs operator:
      name: CRTM
      Absorbers: [H2O,O3]
      obs options:
        Sensor_ID: amsua_n19
```

A taste of what a JEDI configuration file looks like (you'll see more in the other lectures and activities)

Summary

Acquire dependencies through containers, environment modules, or by building them yourself (jedi-stack)

Download and build JEDI using **ecbuild** and **make**

Questions Welcome