Design of a Generic Workflow Generator for the JEDI Data Assimilation System

Mark Olah and Yannick Trémolet
Joint Center for Satellite Data Assimilation (JCSDA)
molah@ucar.edu
ECMWF: Reproducible Workflows Workshop -- Reading UK -- Oct. 15, 2019
# JEDI: Joint Effort In Data Assimilation Integration

## Partner Organizations
- NOAA
- US Navy
- US Air Force
- NASA [EMC]
- NCAR
- UK Met Office

## Agile Development Philosophy
- Git; Github; Git LFS; Git-flow; ZenHub
- Core Languages: C++ and Python
- CMake; ecBuild Bundles
- Automated Testing [CI]
  - CTest, Codecov, CDash, Valgrind
  - TravisCI; AWS CodeBuild
- Containers:
  - Docker, Singularity, Charlie Cloud
- Cloud computing
  - Storage, Compute, Automation, Hosting

## JEDI Core Team
- Lead: Yannick Trémolet
- ~10 FTE Programmers
- Boulder, CO; Greenbelt, MD; France
- Organized < 2 Years Ago
## Reproducibility in Scientific Applications

### Bitwise Reproducibility
- Not generally achievable
- Severe implications for performance
- Inhibits portability
- Requires static, isolated systems
- Slows the pace of development

### Scientifically Meaningful Reproducibility
- Realistic floating point tolerances
- Tolerate benign permutations / ordering
- Handle unexpected events / Avoid failures
- Enhances collaboration
- Aids the pace of development

### Temporal Reproducibility (single system)
- Reproducibility over processor/node counts
- Reproducibility between Machines
- Reproducibility between Compilers / MPI
- Reproducibility between Operating Systems

### Desirable Types of Reproducibility

#### Portability of Runtime Environment
- Reproducibility between Problem Scales
- Reproducibility in Performance Characteristics
- Reproducibility of Algorithms Across Models
- Composability of components and interfaces
- Adaptability to dependency updates
Implications of Full Runtime Portability for JEDI

- Need to move between workstation and Cloud and amongst varying HPC Environments
  - Must package and provide necessary dependencies
- Free and Open-source libraries and tools
  - Proprietary dependencies can be used but not required
- Easily adapt applications to different workflow engines
- Prefer universal, open source data formats.
- Data products must be available to all partners and collaborators
- Build systems must be cross-platform and adapt to wildly different systems
- Generic interfaces
JEDI Runtime Environment Portability

- **Compiler:**
  - GNU 7.4 – 9.2
  - Intel 17 – 19
  - Clang

- **MPI:**
  - OpenMPI
  - Mpich
  - Intel MPI
  - Cray

- **CMake Libraries Build Scripts**
- **autotools Library Build Scripts**

- **JEDI-Stack**

- **Container Image:**
  - Docker
  - Singularity
  - Charlie Cloud

- **Modules (Lmod)**

- **Laptop/Server:**
  - Linux
  - OSX
  - Win64 [WSL]

Reproducibility of runtime environment across systems is accomplished with a common dependency build system with flexibility in input and output configuration.
JEDI Workflow System Structure

- The JEDI system is generic with respect to Model, but presents a common interface
  - Each model produces the same fundamental set of executables
  - Each executable takes a single YAML file as input
  - GOAL: Mirror this structure in overall workflow structure and guidelines
JEDI Generic Executable Interfaces

$ fv3jedi_hofx3D.x config.yaml
JEDI Generic Executable Interfaces

$ fv3jedi_hofx3D.x config.yaml

```
config.yaml

fv3jedi_hofx3D.x
```

```
ml_file: Data/fv3files/fmsmpp.yml
Assimilation Window:
  Begin: '2019-07-09T03:00:00Z'
  Length: PT6H
Geometry:
  ml_file: Data/fv3files/input_gfs_c788.yml
  trc_file: Data/fv3files/field_table
  pathfile_akkb: Data/inputs/gfs_c788/akkb-gfs.nc
Initial Condition:
  filetype: gfs
datapath_tile: Data/inputs/gfs_c788.20190709.00/
filename_core: 20190709.000000.fv_core.res.nc
filename_trcr: 20190709.000000.fv_tracer.res.nc
filename_sfcld: 20190709.000000.sfc_data.nc
filename_sfcm: 20190709.000000.fv_sfc_msl.res.nc
filename_cplm: 20190709.000000.coupler.res
variables: ["T","DELP","sphum","phis"]
Observations:
  ObsTypes:
    - ObsSpace:
      name: GnssroBndROPP2D
      ObsDataIn:
        obsfile: Data/obs/cosmic1/KOMS_20190709_06Z.nc
      ObsDataOut:
        obsfile: Data/hofx/hofx_gfs_c788_gnssro_ropp2d_KOMS_20190709_06Z.nc
      simulate:
        variables: [bending_angle]
      ObsOperator:
        name: GnssroBndROPP2D
      ObsOptions:
      Prints:
        frequency: PT3H
```
JEDI Generic Executable Interfaces

$ fv3jedi_hofx3D.x config.yaml

Observation HofX Output

Observations:
- ObsTypes:
  - ObsSpace:
    - name: GnsrsBndRopp2D
  - ObsDataIn:
    - obsfile: Data/obs/cosmic1/KOM5_20190709_06Z.nc
  - ObsDataOut:
    - obsfile: Data/hofx/hofx_gfs_c768_gnsrs_opp2d_KOM5_20190709_06Z.nc
    - simulate:
      - variables: [bending_angle]
  - ObsOperator:
    - name: GnsrsBndRopp2D
  - ObsOptions:
    - Prints:
      - frequency: PT3H
JEDI Generic Executable Interfaces

$ fv3jedi_hofx3D.x config.yaml

config.yaml  →  fv3jedi_hofx3D.x

Aux model config files
Observation Input Files
Model Background

Observation HofX Output

```
nml_file: Data/fv3files/fmsmpp.nml
Assimilation Window:
  Begin: ['2019-07-09T03:00:00Z']
  Length: PT6H
Geometry:
  nml_file: Data/fv3files/input_gfs_c788.nml
  trc_file: Data/fv3files/field_table
  pathfile_akbk: Data/inputs/gfs_c788/akbk-gfs.nc
Initial Condition:
  filetype: gfs
datapath_tile: Data/inputs/gfs_c788.20190709.00/
filename_core: 20190709.000000.fv_core.res.nc
filename_trcr: 20190709.000000.fv_tracer.res.nc
filename_sfcv: 20190709.000000.sfc_data.nc
filename_sfcw: 20190709.000000.fv_srf_wnd.res.nc
filename_cplr: 20190709.000000.coupler.res
variables: ['T', 'DELP', 'sphum', 'phis']
Observations:
  ObsTypes: [- ObsSpace:  name: GnssrBndRopp2D
                         ObsDataIn:
                           obsfile: Data/obs/cosmic1/KOM5_20190709_06Z.nc
                         ObsDataOut:  obsfile: Data/hofx/hofx_gfs_c768_gnssr_opp2d_KOM5_20190709_06Z.nc
                             simulate:  variables: [bending_angle]
               ObsOperator:  name: GnsssrBndRopp2D
                                ObsOptions:
Prints:
  frequency: PT3H
```
$ fv3jedi_hofx3D.x config.yaml

JEDI Generic Executable Interfaces

Aux model config files

Observation Input Files

Model Background

config.yaml

fv3jedi_hofx3D.x

Observation HofX Output

out.log

nml_file: Data/fv3files/fmsmpp.nml
Assimilation Window:
  Begin: '2019-07-09T03:00:00Z'
  Length: PT6H
Geometry:
  nml_file: Data/fv3files/input_gfs_c788.nml
  trc_file: Data/fv3files/field_table
  pathfile_akbk: Data/inputs/gfs_c788/akbk-gfs.nc
Initial Condition:
  filetype: gfs
datapath_file: Data/inputs/gfs_c788.20190709.00/
filename_core: 20190709.060000.fv_core.res.nc
filename_trcr: 20190709.060000.fv_tracer.res.nc
filename_sfcd: 20190709.060000.sfc_data.nc
filename_sfew: 20190709.060000.fv_srf_wnd.res.nc
filename_cplr: 20190709.060000.coupler.res
variables: ['T', 'DELP', 'sphum', 'phis']
Observations:
  ObsTypes:
  - ObsSpace:
      name: GnsrsOrbRdRopp2D
  ObsDataIn:
    obsfile: Data/obs/cosmic1/KOM5_20190709_06Z.nc
  ObsDataOut:
    obsfile: Data/hofx/hofx_gfs_c768_gnsrs_opp2d_KOM5_20190709_06Z.nc
    simulate:
      variables: [bending_angle]
  ObsOperator:
    name: GnsrsOrbRdRopp2D
  ObsOptions:
Prints:
  frequency: PT3H
JEDI Generic Executable Interfaces

$ fv3jedi_hofx3D.x config.yaml

```
{nl_file: [(app.config_dir)]/fmsmpp.nml
Assimilation Window:
Begin: {{window.begin}}
Length: {{window.length}}
Geometry:
{nl_file: [(app.config_dir)]/input_{(model.desc)}.nml
trc_file: [(app.config_dir)]/field_table_{(obs.class)}
pathfile_akkb: [(app.config_dir)]/akkb-{(model.name)}.nc
Initial Condition:
filetype: {model.name}
datapath_tile: {(app.data_dir)}/{(model.name)}/bg/{(model.date_dir)}
filename_core: {(model.date_file)}.{fvd_core.res.nc
filename_trcr: {(model.date_file)}.{fvd_tracer.res.nc
filename_sfcdata: {(model.date_file)}.{sfc_data.nc
filename_sfcw: {(model.date_file)}.{fvd_srf_wnd.res.nc
filename_cplcr: {(model.date_file)}.{coulper.res
variables: {model.variables}
Observations:
ObsTypes:
- ObsSpace:
  name: GnssrBndRDP2D
  ObsDataIn:
    obsfile: {(app.data_dir)}/{(obs.gnssro.kompasat5.obs_file_path)}
  ObsDataOut:
    obsfile: "{{app.output_dir}}/{(model.name)}/hofx/\n    {{model.descr}}_{(model.name)}_{(model.date_str)}.nc4"
    simulate:
      variables: [bending_angle]
  ObsOperator:
    name: GnssrBndRDP2D
  ObsOptions:
    Prints:
      frequency: {(app.print_frequency)}
```
JEDI-Rapids application design goals

- Python 3.6+
- Version controlled repository of
  - Apps (i.e., Tasks, Nodes)
  - Workflows descriptions (i.e., Suites)
- Configuration of Apps YAML / Jinja2
- Generation of JEDI configurations with YAML / Jinja2
- Inheritance and reuse of Apps

- System-agnostic App configurations
- Capture all relevant App state in a single YAML formatted configuration file (Reproducibility)
- Flexibility in workflow system
- Unified dependency and build system for App executables
JEDI Workflow Axes of Generality

Systems
- HPC
  - NOAA; NASA; NCAR;
  - Met Office; NAVY; US DOE
- Cloud (AWS)
- Workstation
- Laptop (Linux, OSX, Win [WSL])

Models
- MPAS
- FV3-GFS / FV3-GEOS
- MOM6 / SOCA
- Neptune
- LFRic
- ...

Workflow systems
- ecFlow
- Cylc
- Apache Airflow

Environments
- GNU / Intel / Clang
- OpenMPI / mpich / IMPI / Cray
- Modules / Containers / Native Pkg
- FS: Cloud / Distributed / Parallel
# Python vs Shell Scripting for Workflows

## Shell Scripts
- Are not portable
  - Dozens of undefined program dependencies.
  - Standard programs can be incompatible across systems
- Lack true debugging facilities
- Are difficult to reuse and compose into larger programs
- Do not provide data structures
- Lack functional and object-oriented language features
- Are dangerously intertwined with environment variables
- Call programs with external state stored in local config files
  - No universal way to capture full state
- Lack native parallelism and threading facilities
- Rely on clunky command line interface to external tools (No APIs)

## Python
- Cross-platform
- Dependency management
  - pip, virtualenv, conda, …
- Debuggers
- Profilers
- Advanced Data structures
- Object-oriented and functional programming
- Composable and reusable (Modules)
- RE parsing and string formatting
- Datetime manipulations
- pathlib filesystem manipulation
- Logging
- Advanced APIs
- Plotting
- Data format interfaces
- Multiprocessing, threading, futures, …
- Network libraries
JEDI Workflow Features Facilitating Reproducibility

**Portability**
- Apps are Python-3.6+
- JEDI-stack – Portable environment
  - Provides all library dependencies
  - Lua Modules / Containers
  - Intel / GNU / CLANG
  - OpenMPI / mpich / impi / ...

**Data Integrity**
- Universal checksums
- S3 ETags
- Storage:
  - DB / AWS S3: Metadata
  - Filesystem: YAML aux files

**Provenance Tracking**
- System information (Python platform)
- Software GIT tags/versions
- Input product checksum metadata
- Stored as aux files YAML-format

**Portability**
- Apps are Python-3.6+
- JEDI-stack – Portable environment
  - Provides all library dependencies
  - Lua Modules / Containers
  - Intel / GNU / CLANG
  - OpenMPI / mpich / impi / ...

**Provenance Tracking**
- System information (Python platform)
- Software GIT tags/versions
- Input product checksum metadata
- Stored as aux files YAML-format

**Logging / Configuration**
- App YAML Config
  - Python ruamel provides round trip preserving of comments
- JEDI Executable config.yaml
- Logs: App / JEDI
- Runtime statistics (CPU / MEM / IO)
  - Python psutil
JEDI-Rapids Application Performance Monitoring

Based on Python psutil
- Cross-platform
- Free, Open-source
- /proc filesystem-based
- Highly configurable
  - System memory
  - Swap memory
  - Per-Process memory
    - RSS, PSS, USS
- System load-avg
- CPU times (user, system)
- Threads
- I/O
- Context switches
- System calls
- Network stats
- JEDI-Rapids polls at regular intervals
  - Saved as Python pickle
  - plots are post-processed
JEDI-Rapids Application Performance Monitoring

- No proprietary software
- No instrumentation of code
- No recompiling or special flags
- Full operational performance
- Works with any language
  - Fortran, C++, Python, etc.
- Works on any system with no special external libraries
- Works on laptops, in the cloud, and on multi-node MPI jobs (aggregation)
- Comparison between runs, machines, problem sizes, algorithms.
JEDI-Rapids Workflow Development Plans

• Automated Workflow Suite Generation
  • Programmatic open-ended cycling DA generic workflows
• Front-end GUI
• Porting to HPC
  • NASA, NOAA, US Navy, US DOE, Met Office
• Universal Observation Ingest System