

# Reproducible workflows for big data with Python

Building reproducible workflows for earth sciences

Iain Russell

Development Section, ECMWF



ecFlowUI (4.7.1) - (menu: User)

File Panels Refresh Servers Tools Help

meta... local avi mdev avi pikachu cassio rdpal odorir metabui...marsflow

make\_build\_dir

- var\_summary
- git\_srcs
- make\_build\_dir
- config
- build
- test
- create\_tarball
- install
- install\_log
- remove\_write\_perm
- module\_sync
- wipe\_build
- bxg
- bx
- leap42
- sappb
- cca
- ccb
- metview

File: /home/ma/deploy/servers/ecflow-metab.5062/metabuilder/suites/ecflow/ecflow/ecgb/make\_build\_dir.job1 Size: 14 KB Modified: 2017-11-07 10:23:36

Source: read from disk

```
4 #SBATCH --get-user-env
5 #SBATCH --job-name=make_build_dir_ecgb
6 #SBATCH --output=/scratch/ma/deploy/ecf_out/ecflow-metab.5062/ecflow/ecgb/make_build_dir.job1
7 #SBATCH --error=/scratch/ma/deploy/ecf_out/ecflow-metab.5062/ecflow/ecgb/make_build_dir.job1
8 #SBATCH --uid=deploy
9 . /home/ma/emos/.profile
10 #!/bin/ksh
11
12 # standard header
13 set -e # stop the shell on first error
14 set -u # fail when using an undefined variable
15 set -x # echo script lines as they are executed
16 set -o pipefail # fail if any part of a shell pipeline fails
17 ELAPSED_TIMING_START=$SECONDS
18
19 echo "HOST = $(hostname)"
20
21 # need at least one qsub for trimurti
22
```

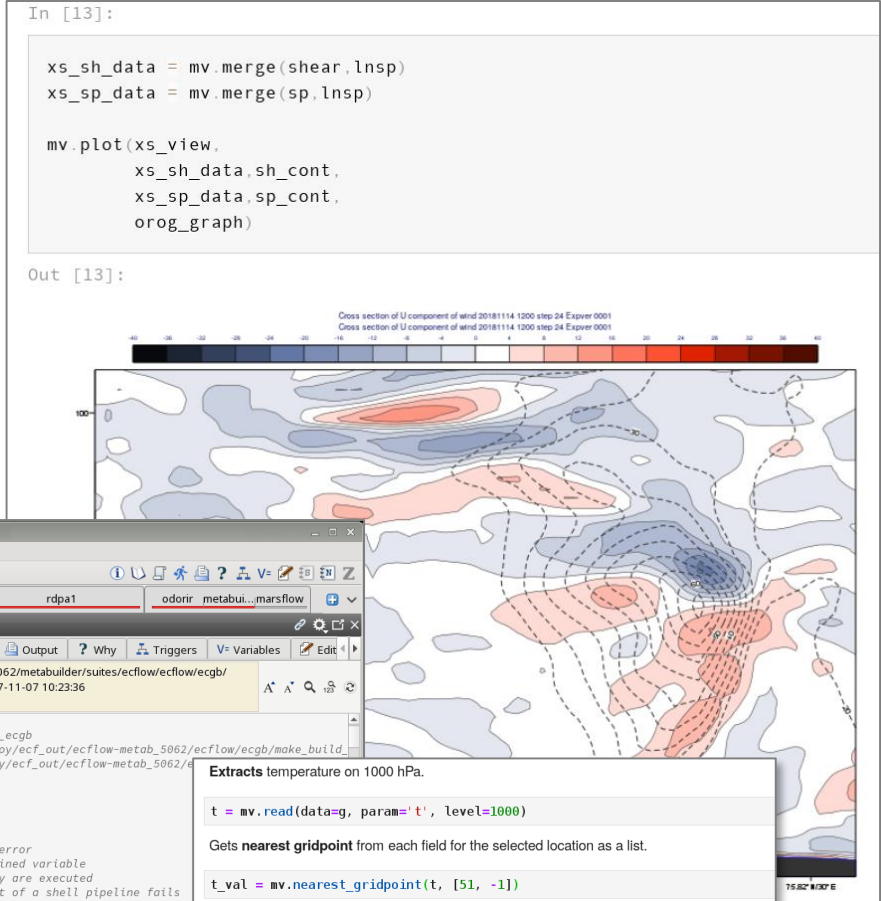
No selection

Filter:

Node	Status	Type	Trigger	Label	Event
/prodgen/cron/cron_monitor	queued	task			
/ecflow/test_local_actions/action1	suspended	task			
/ecflow/test_local_actions	suspended	family			
/ecflow/set_module_new	complete	task			
/ecflow/regenerate	complete	task			
/ecflow/localhost/intel/wipe_build	queued	task	install_log == complete		

date: Tue 7 Nov

Notifications: Aborted Late



Extracts temperature on 1000 hPa.

```
t = mv.read(data=g, param='t', level=1000)
```

Gets nearest gridpoint from each field for the selected location as a list.

```
t_val = mv.nearest_gridpoint(t, [51, -1])
```

Gets valid date from the GRIB headers (it is computed from multiple keys) as a list of Python datetime objects.

```
d_val = mv.valid_date(t)
```

Prints the dates and values together.

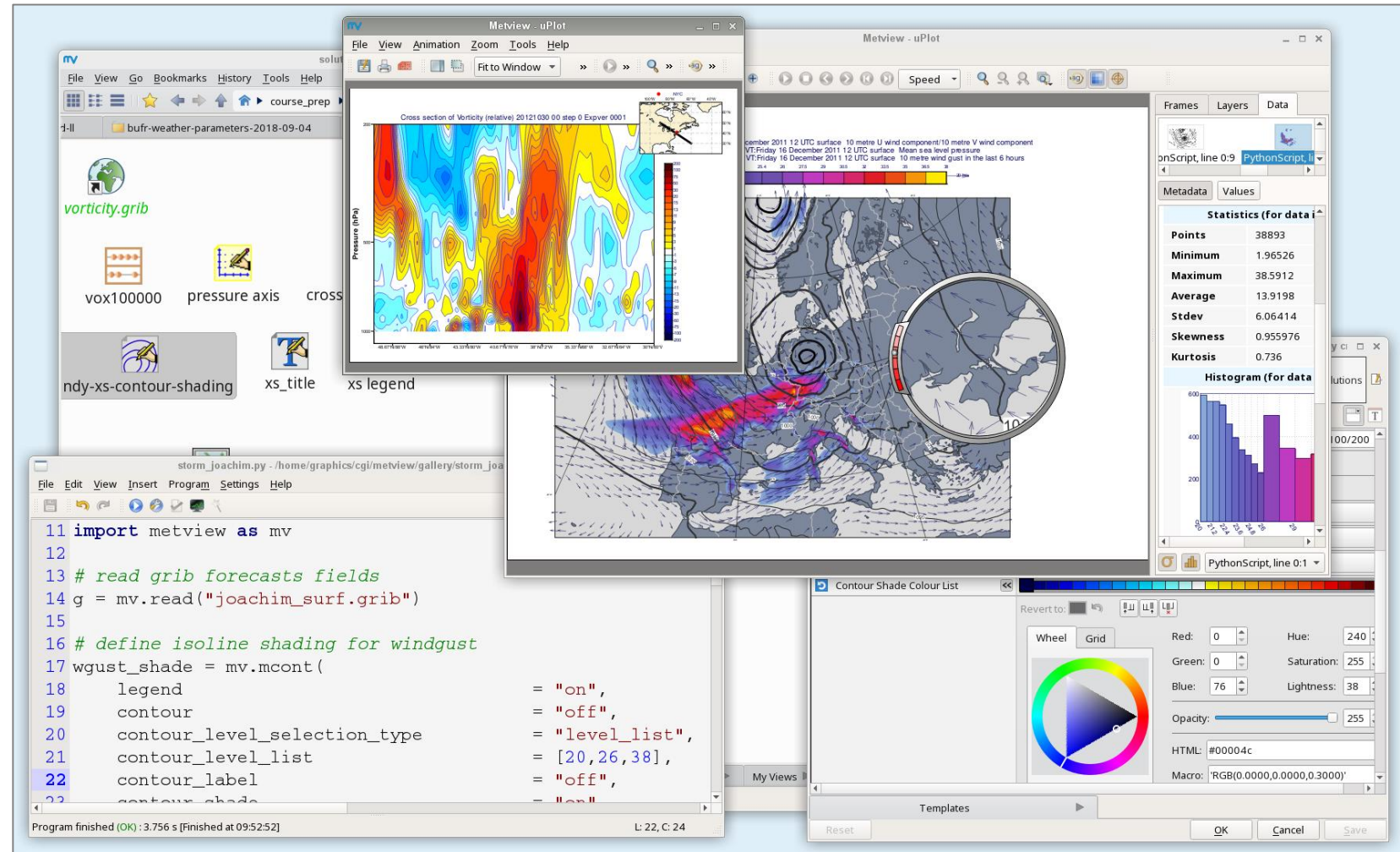
```
for dd,vv in zip(d_val, t_val):
    print('{} h --> {} K'.format(dd,vv))
```

```
2019-05-02 12:00:00 h --> 285.30157470703125 K
2019-05-02 18:00:00 h --> 283.7098083496094 K
2019-05-03 00:00:00 h --> 281.2944030761719 K
2019-05-03 06:00:00 h --> 280.19874572753906 K
```

# Metview



- ECMWF's workstation software for retrieving, processing and plotting meteorological data (UNIX, laptops to HPCs)
- Interactive GUI plus batch processing with Python
- Works with large or small data sets
- Open Source under Apache Licence 2.0
- Metview is a co-operation project with INPE (Brazil)



## High level API means less change in workflow

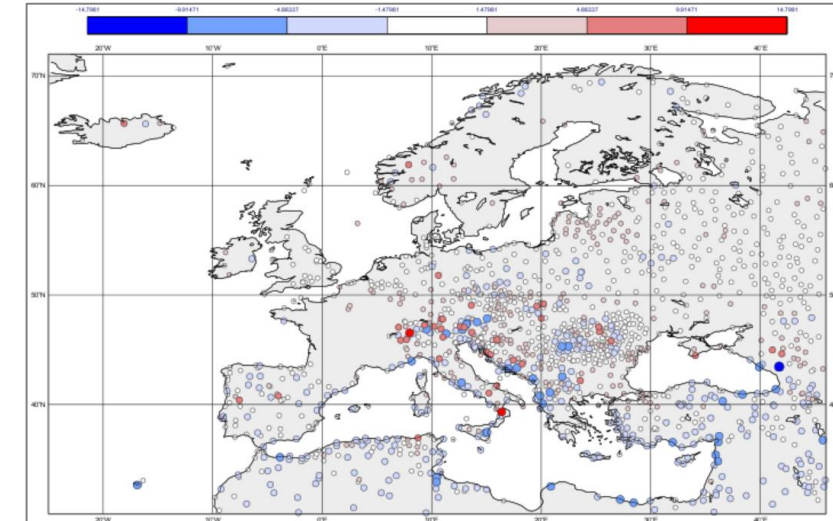
- Example 1: load (gridded) GRIB and (scattered) BUFR files, filter the BUFR data, then compute the difference
- ECMWF has changed both its GRIB and BUFR decoding/encoding libraries (GRIBEX/BUFRDC -> ecCodes)
- Direct users of these libraries had to change a lot of code
- Users of Metview had little or nothing to change
  - The Metview developers did the work!
  - Scientists' scripts can better define the workflow rather than get into details about file formats

```
In [1]: import metview as mv

t2m_grib = mv.read('t2m_grib.grib')
obs_3day = mv.read('obs_3day.bufr')

t2m_gpt = mv.obsfilter(
    data = obs_3day,
    parameter = 'airTemperatureAt2M',
    output = 'geopoints'
)

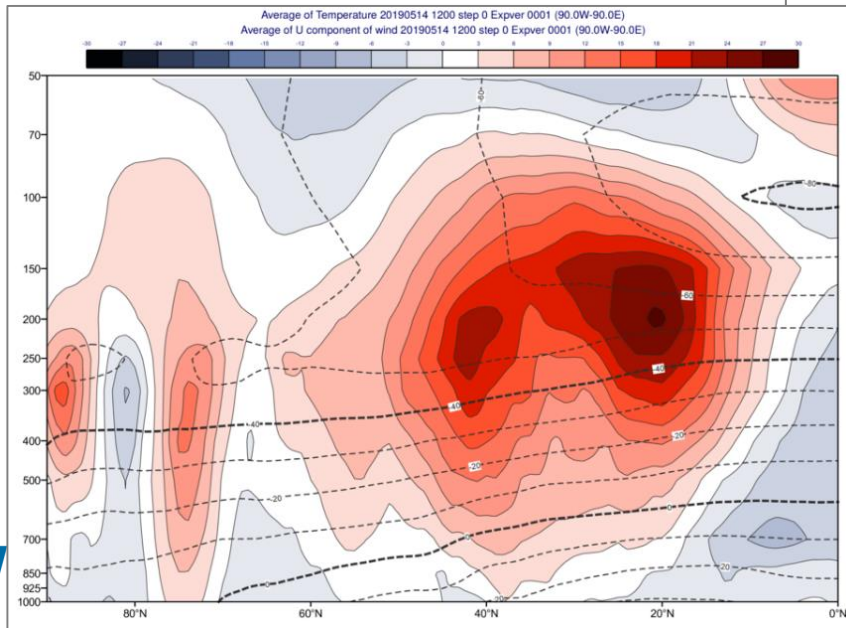
diff = t2m_grib - t2m_gpt
```





## Another case - regridding

- Example 2: filter and regrid GRIB fields onto a 0.5x0.5 degree grid and plot a zonal mean section
- Works regardless of number of fields
- ECMWF has changed its interpolation library (emoslib -> MIR)
- Users of Metview (and MARS) had nothing to change



```
In [ ]: import metview as mv

# read pressure level data
g = mv.read("avg_tuv.grib")

# filter u (zonal wind component) and t
u = mv.read(data=g, param='u', grid=[0.5,0.5])
t = mv.read(data=g, param='t', grid=[0.5,0.5])

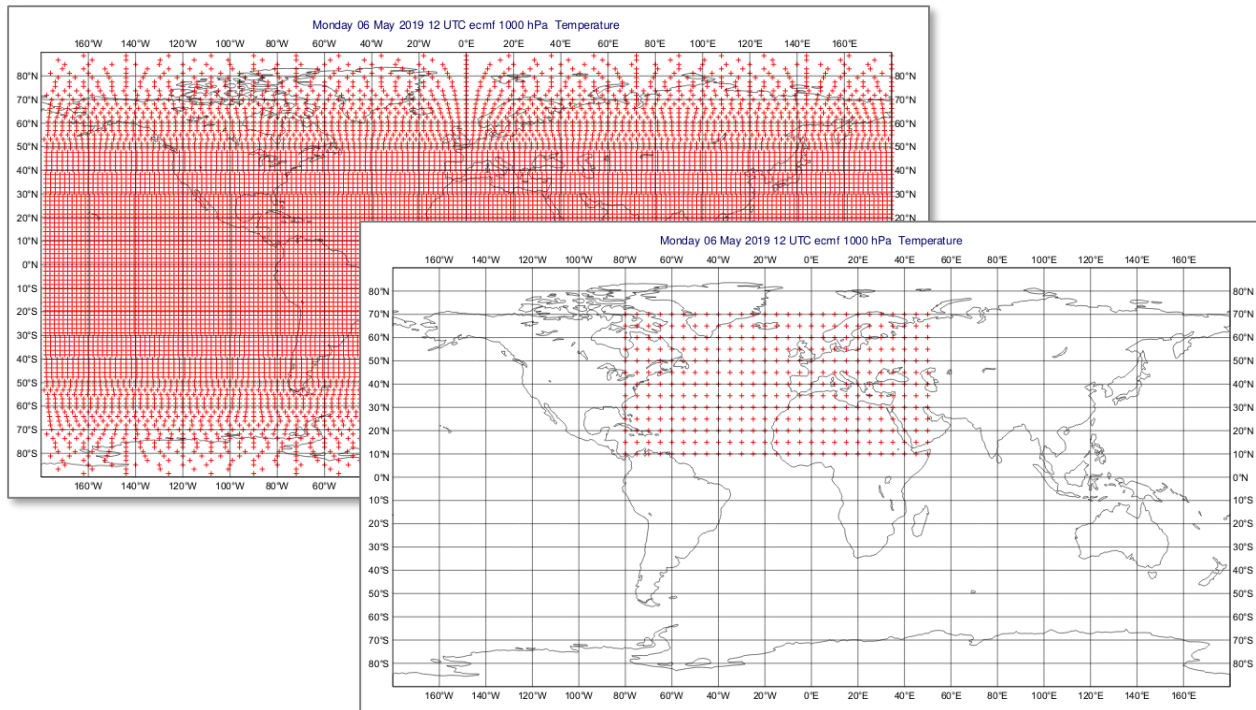
# scale temperature to celsius
t = t - 273.16

# define averaging area - one half of the North
area = [90,-90,0,90] #N,W,S,E

# define average view for zonal mean
view = mv.maverageview(
    top_level          = 50,
    bottom_level       = 1000,
    vertical_scaling    = "log",
    area               = area,
    direction          = "ew",
    horizontal_axis     = horiz_axis,
    vertical_axis      = vertical_axis)
```

## Another case – retrieval and regridding

- MARS retrieval requests that worked 30 years ago should work today
- Regridding techniques may have changed (improved)



```
t2m = mv.retrieve(
    type      = "fc",
    levtype   = "sfc",
    param     = "2t",
    date      = -5,
    step      = 72,
    grid      = [0.5, 0.5]
)
```

```
RETRIEVE,
TYPE      = FC,
LEVTYPE   = SFC,
PARAM     = 2T,
DATE      = -5,
STEP      = 72,
GRID      = 0.5/0.5
```



# ecFlow

- Workflow management software developed at ECMWF (Open Source, Apache licence)

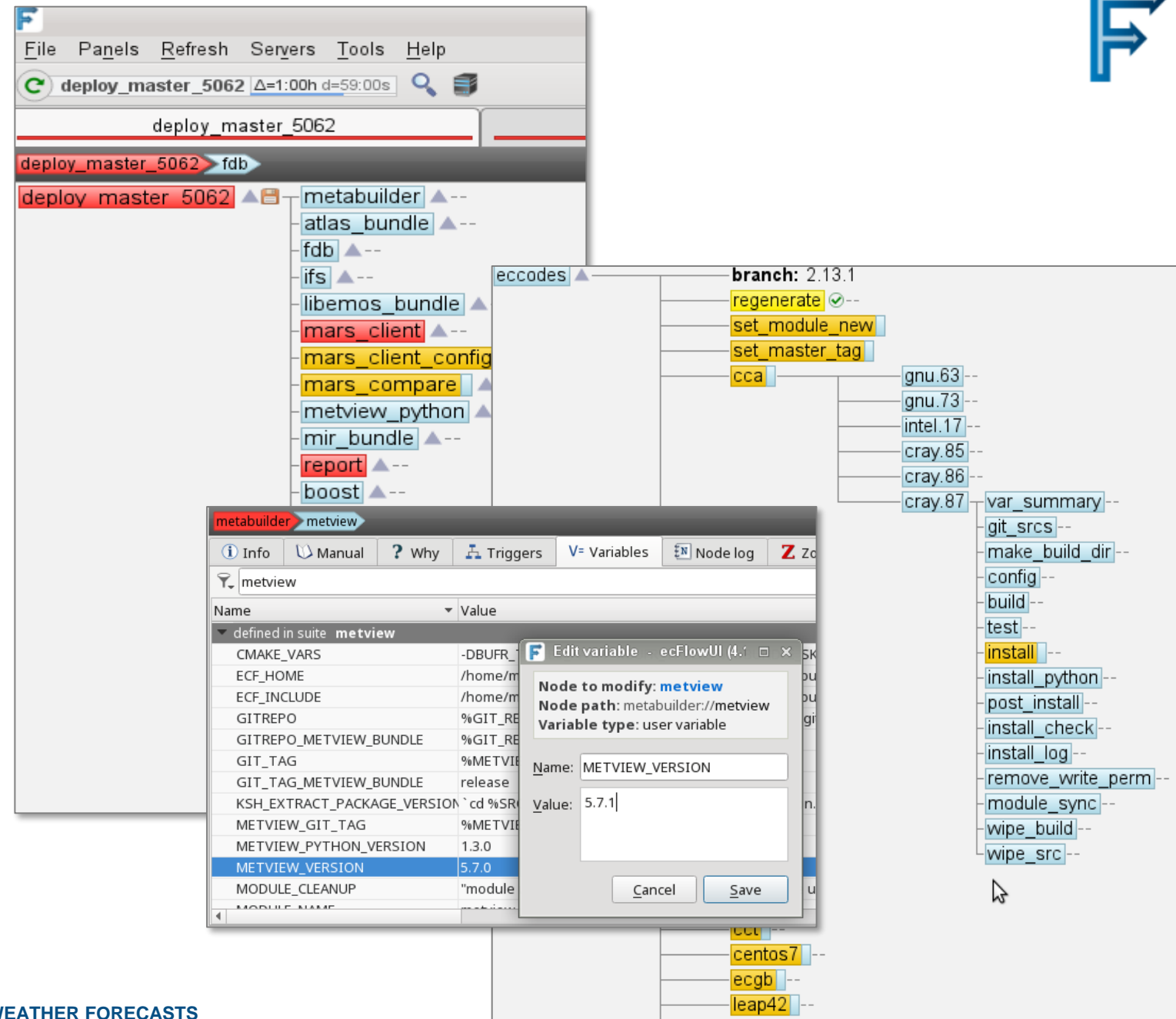
The screenshot displays the ecFlowUI (4.7.1) interface, showing a workflow management system. The main window displays a tree view of workflow components, including 'make\_build\_dir', 'var\_summary', 'git\_srcs', 'config', 'build', 'test', 'create\_tarball', 'install', 'install\_log', 'remove\_write\_perm', 'module\_sync', and 'wipe\_build'. A terminal window shows the execution of 'wab' and 'gtwx' commands. A table lists workflow nodes with their status (active, aborted, queued, suspended) and triggers. The bottom panel shows a list of tasks with their status and completion dates.

Node	Status	Type	Trigger	Label	Event	Meter	Status changed
/wab/gtwx/an/lag	aborted	family					2017-Nov-09 08:33:00
/wab/gtwx/an	aborted	family					2017-Nov-09 08:33:00
/wab/gtwx	aborted	family		infopsc: 'ccb'			2017-Nov-09 08:33:00
/wab/gtwx/an/main/lw12/an/4dvar	active	task	ifstraj == complete				2017-Nov-09 08:47:00
/wab/gtwx/an/main/lw12/an/4dvar	active	family	uptraj_0 == complete				2017-Nov-09 08:47:00
/wab/gtwx/an/main/lw12/an/4dvar	active	family	odb == complete and vardata =				2017-Nov-09 08:33:00
/wab/gtwx/an/main/lw12/an	active	family	/wab/gtwx/an/obs/lw12 == cor				2017-Nov-09 08:33:00
/wab/gtwx/an/main/lw12	active	family					2017-Nov-09 07:54:00
/wab/gtwx/an/main/lw00/fcdong/n	active	task	getpersSST == complete AND info:			step: 120	2017-Nov-09 07:54:00

Status	Type	Trigger	Label	Event	Meter	Status changed
queued	task					2017-Nov-07 21:08
suspended	task					2017-Nov-07 16:24
suspended	family					2017-Nov-07 16:24
complete	task					2017-Nov-07 16:24
complete	task					2017-Nov-07 16:24
complete	task					2017-Nov-07 16:24
queued	task	install_log == complete				2017-Nov-07 16:24

# The Metabuilder

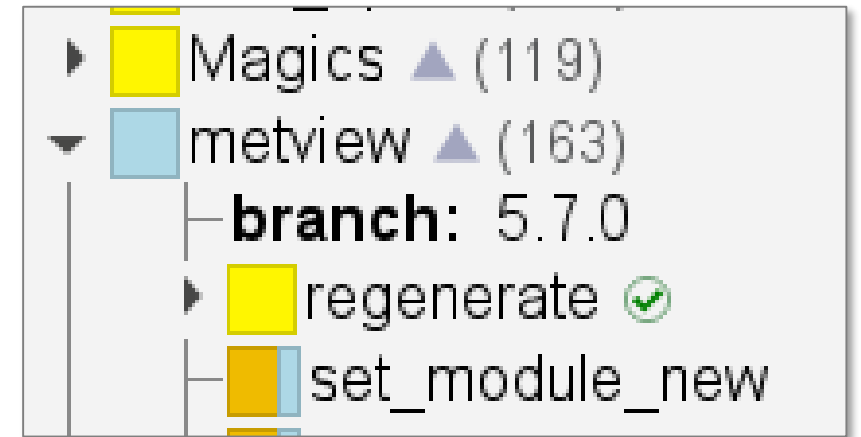
- A set of Python scripts that generate ecFlow suites to **build**, **test** and **deploy** our software packages
- Default scripts for each task, but can be overridden by each package
- Package maintainers can deploy their own packages in isolation 'manually' with the metabuilder
- But automation is better for a reproducible workflow!



The screenshot displays the ecFlowUI interface for the 'deploy\_master\_5062' suite. The main window shows a tree view of the workflow tasks, including 'metabuilder', 'atlas\_bundle', 'fdb', 'ifs', 'libemos\_bundle', 'mars\_client', 'mars\_client\_config', 'mars\_compare', 'metview\_python', 'mir\_bundle', 'report', and 'boost'. A secondary window titled 'metabuilder' shows a list of variables defined in the suite, with 'METVIEW\_VERSION' highlighted. A third window, 'Edit variable', is open, showing the 'Node to modify: metview' and the 'Variable type: user variable'. The 'Name' field is set to 'METVIEW\_VERSION' and the 'Value' field is set to '5.7.1'.

## Running modes

- The Metabuilder very much uses the git branches and tags of our packages
  - Relies on us using the ‘gitflow’ workflow
- ‘CI’ mode – continuous integration
  - runs every day
  - builds the **development branches** of all our packages routinely to test for errors introduced that day
- ‘Release CI’ mode
  - tests the **release branches**
  - do this for a few days before a ‘synchronised release’
- Deployment mode
  - use the **version tags** on the master branches
  - tags should not be moved!
  - build, test and deploy





## Synchronised release

- Every couple of months we do a co-ordinated 'synchronised release' of all our software packages
- The Metabuilder's code and configuration are stored in git, so we tag it after each 'synchronised release'
- The idea is that we could recreate any given synchronised release
- E.g. we get a new platform on which we want to install the stable versions of all our packages

```
22 qt_version = '5.12.3'
23 valgrind_version = '3.14.0'
24
25 ecbuild_version = '3.1.0'
26 eccodes_version = '2.14.0'
27 eccodes_python_version = '0.9.3'
28
29 atlas_version = '0.19.0'
30 mir_version = '1.3.0'
31
32 odb_version = '1.0'
33 odb_api_version =
34 odb_tools_version
35 |
36 magics_version =
37 magics_python_ver
38 metview_version =
39 metview_python_ve
```



ecSDK / metabuilder

### Commits

master ▾ ...

Branches Tags

Enter a tag name

2019.10.0

2019.07.0

2019.05.0

# Summary

- Using high-level APIs clarify workflows and can help reproducibility over time
  - Should ideally be fairly agnostic as to size of data
- Version tracking is important
- User workflow and machine workflow are linked
- Questions?

