ecFlow

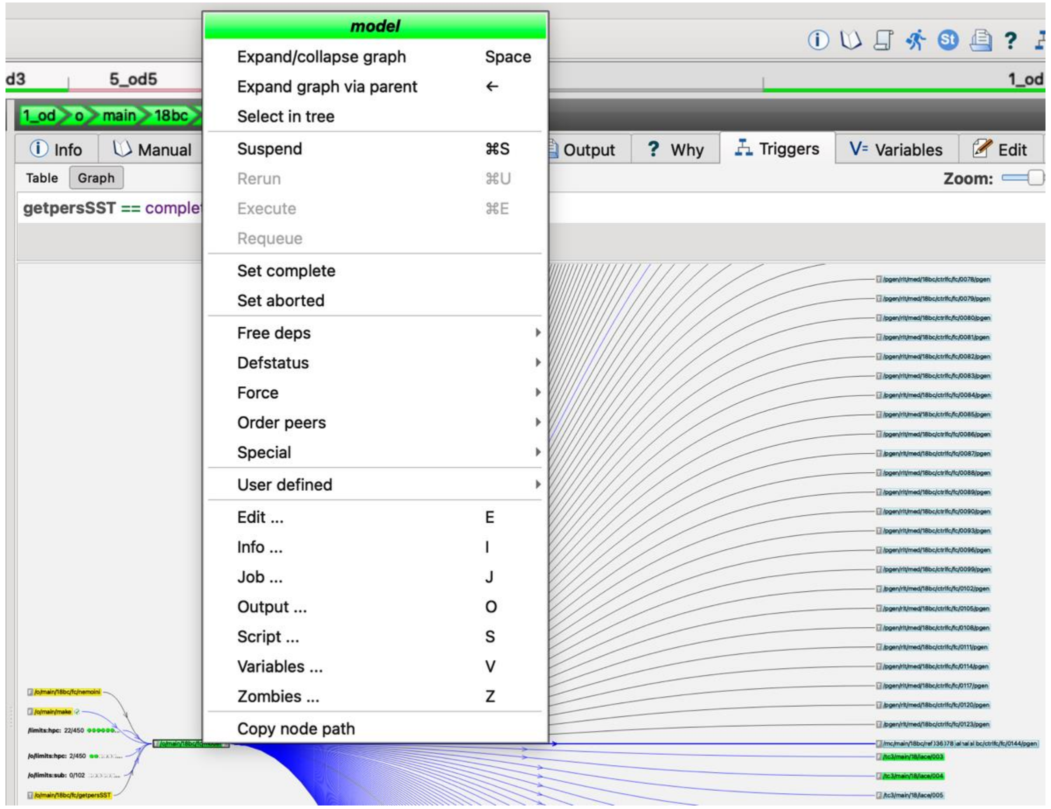
Best Practices and Real-life Workflow examples

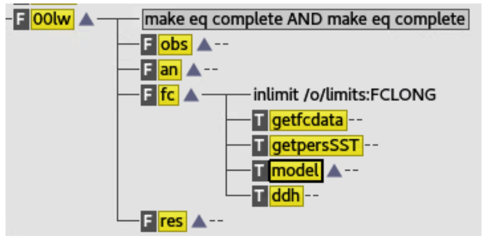
Axel Bonet, Iain Russell, Marcos Bento

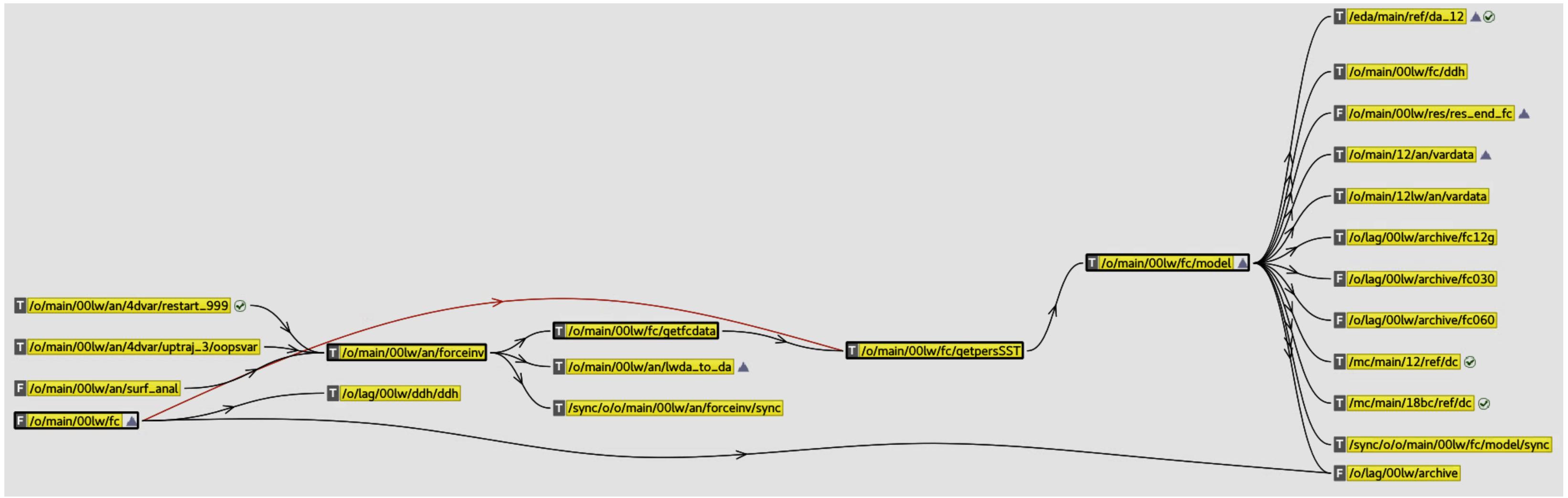
ECMWF

ecFlow: Best Practices?

- Aims
  - How ecFlow can be used
  - Show few ecFlow use cases
  - Advertise suites coding standards manual
    - Hint toward Pyflow ecosystem
    - A glossary
- Overview
  - Architecture
  - Suites design
  - Tasks wrappers design
  - Monitoring & Observability
  - Error Handling
  - Security & Access
  - Operational Tips







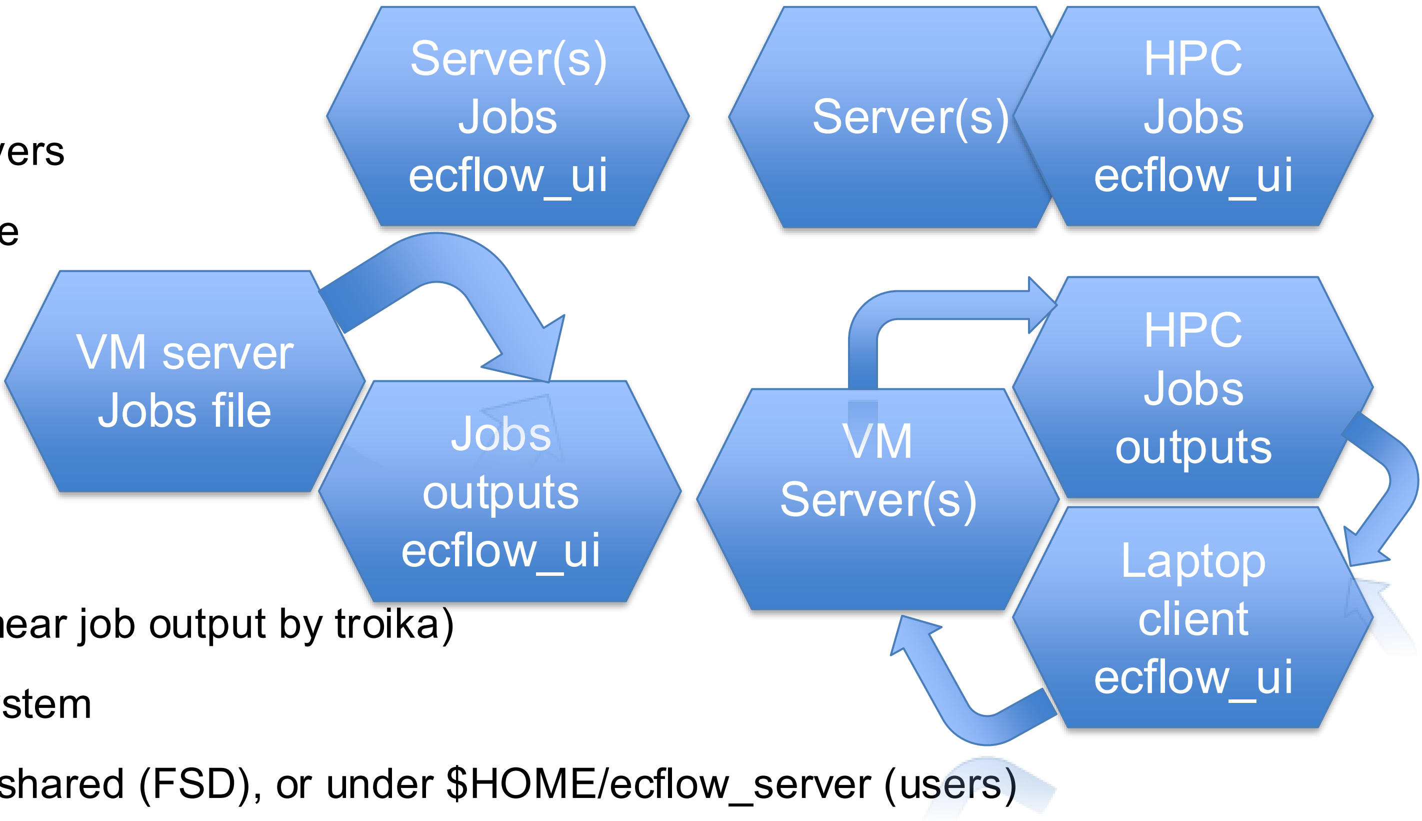
T /o/main/00lw/an/4dvar/restart_999
T /o/main/00lw/an/4dvar/uptraj_3/oopsvar
F /o/main/00lw/an/surf_anal
F /o/main/00lw/fc
T /o/main/00lw/an/forceinv
T /o/lag/00lw/ddh/ddh
T /o/main/00lw/fc/getfcdata
T /o/main/00lw/an/lwda_to_da
T /sync/o/o/main/00lw/an/forceinv/sync
T /o/main/00lw/fc/getpersSST
T /o/main/00lw/fc/model
T /eda/main/ref/da_12
T /o/main/00lw/fc/ddh
F /o/main/00lw/res/res_end_fc
T /o/main/12/an/vardata
T /o/main/12lw/an/vardata
T /o/lag/00lw/archive/fc12g
F /o/lag/00lw/archive/fc030
F /o/lag/00lw/archive/fc060
T /mc/main/12/ref/dc
T /mc/main/18bc/ref/dc
T /sync/o/o/main/00lw/fc/model/sync
F /o/lag/00lw/archive

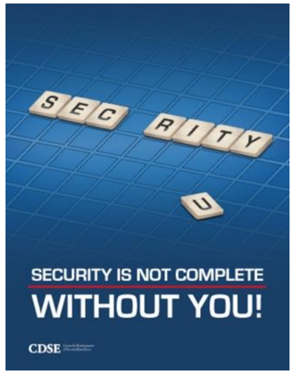
ECMWF

EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

ecFlow best practices – servers' configuration

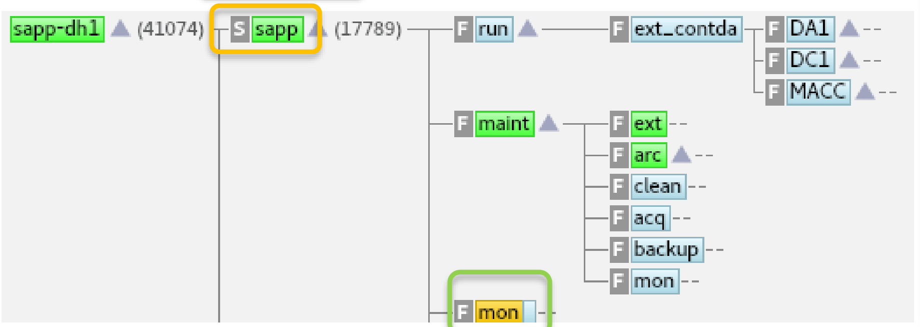
- From introduction, we know:
  - Multiple servers per user is possible on one CPU
  - Multiple users can share same CPU for their servers
  - Multiple users sharing the same server is possible
  - Setup a **whitelist** file to restrict access
    - use **TLS (SSL)** mode where needed,
    - use **password** for best security
- Commissioning **one VM per server**
  - **Jobs files** are local, transients (copied to HPC near job output by troika)
  - **Tasks wrappers** on a mounted persistent file system
  - **Checkpoint file** and **server log** are local (RD), shared (FSD), or under $HOME/ecflow_server (users)
  - VM and server health under the watch of **observability tools** (Splunk, Opsview, Etcd, Grafana, …)
  - Server **automatic restart** (RD, users) vs server restart in **halted mode** (production)
  - A **logserver** is part of ecFlow sources, to run on the target host, to make output visible to the GUI
  - UDP server on the same VM, REST/API on the same VM with memory requirements

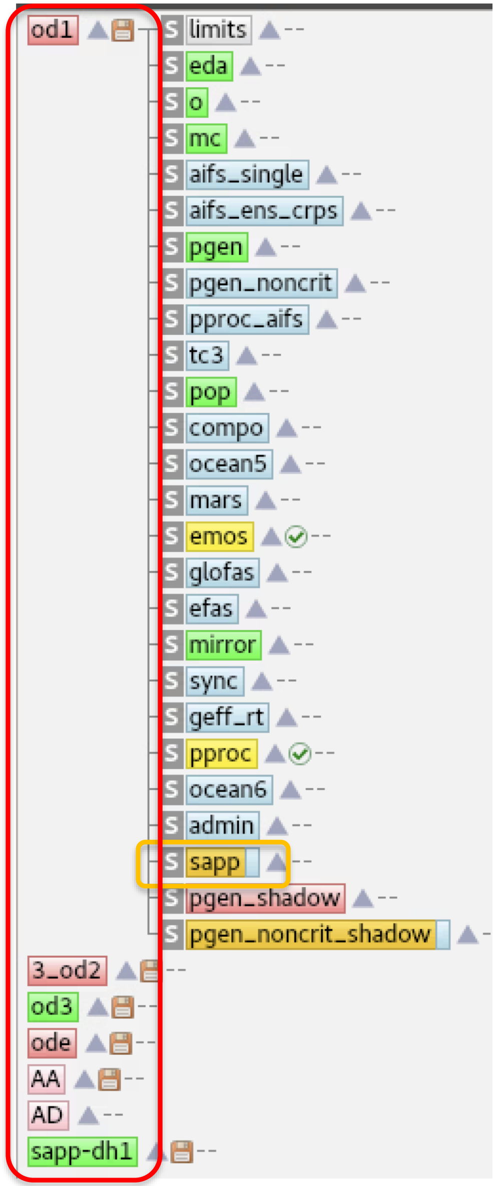


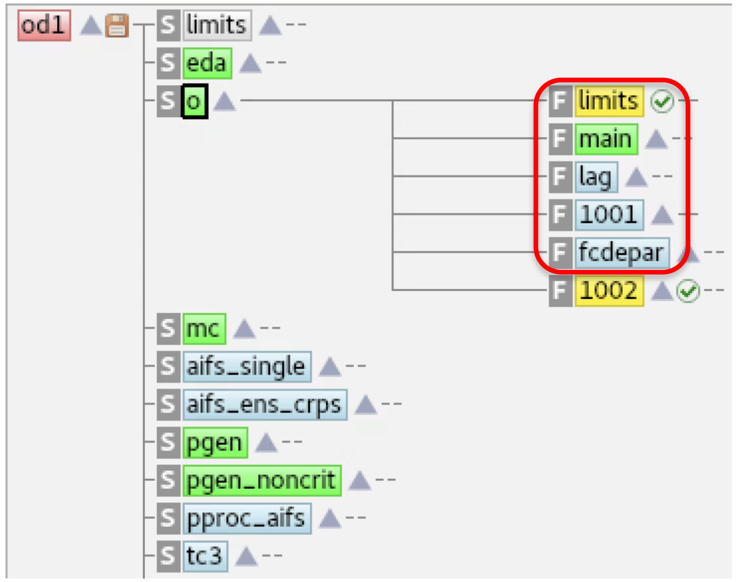


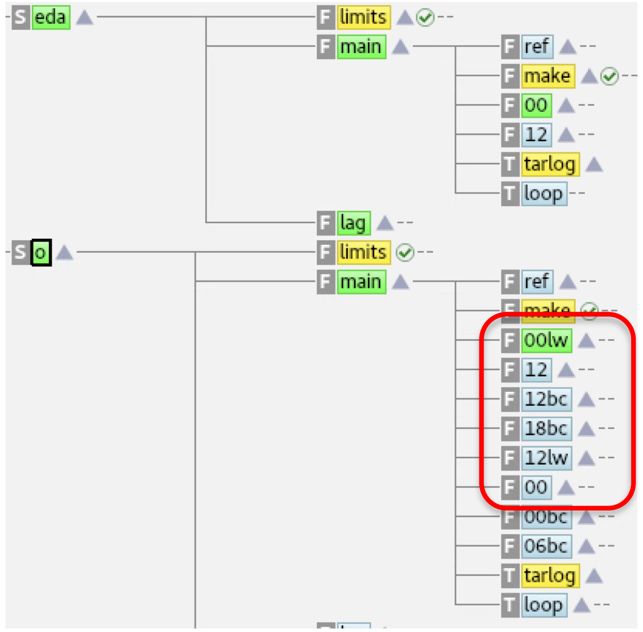
ecFlow in Operation: EMOS operational servers

- EMOS servers:
  - Display criticality: 1_od, 3_od2, 5_od3, 9_ode
- Suite structure reflects on-call criticality:
  - Main: critical path
  - Lag: archive, slow postprocessing
  - Other postprocessing families
  - Inner vs outer watchdog
- Suspended suites may be lively entities:
  - ECF_PASS: FREE, monitoring mode
  - Mirror suite: reflecting a suite on another server



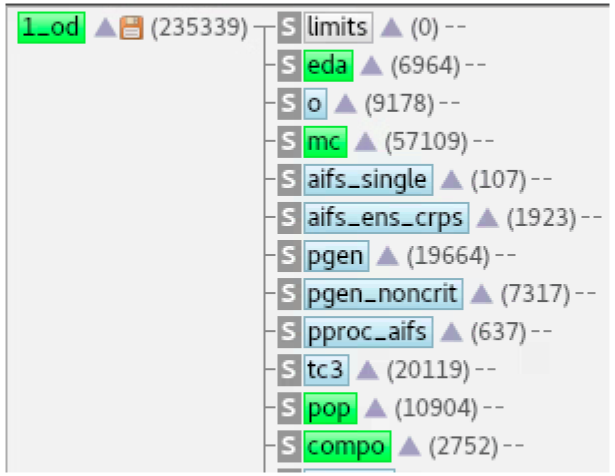




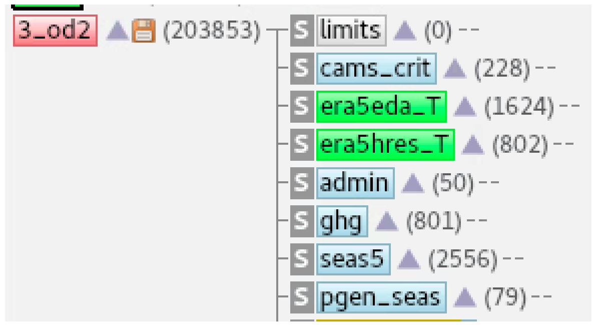


ecFlow best practices – servers' configuration - management

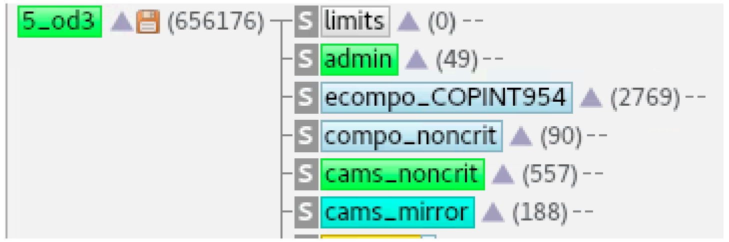
- A federation of servers working together:
  - visual **hierarchy**: top has priority
  - **load balance suites**:
    - od1 (daily run) od2 (monthly) od3 (special projects)
    - Ode (aka emos-0) for e-suites
  - Main and **backup servers** od1, AA, AD



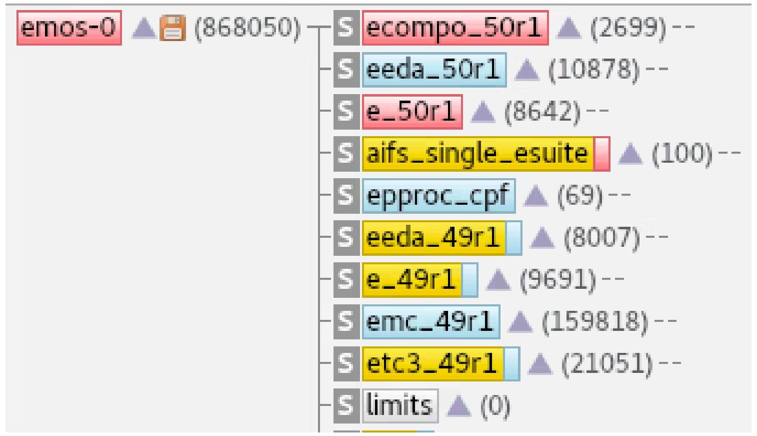
```
1_od (235339) ─ S limits (0) --
               S eda (6964) --
               S o (9178) --
               S mc (57109) --
               S aifs_single (107) --
               S aifs_ens_crps (1923) --
               S pgen (19664) --
               S pgen_noncrit (7317) --
               S pproc_aifs (637) --
               S tc3 (20119) --
               S pop (10904) --
               S compo (2752) --
```



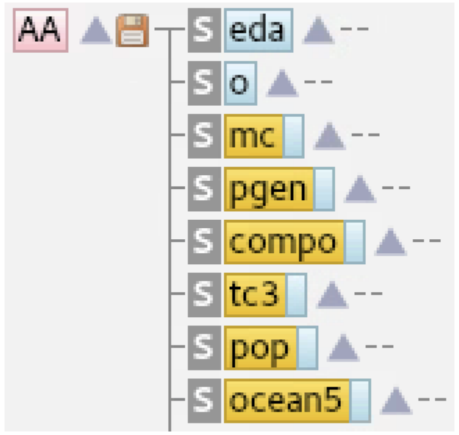
```
3_od2 (203853) ─ S limits (0) --
                 S cams_crit (228) --
                 S era5eda_T (1624) --
                 S era5hres_T (802) --
                 S admin (50) --
                 S ghg (801) --
                 S seas5 (2556) --
                 S pgen_seas (79) --
```



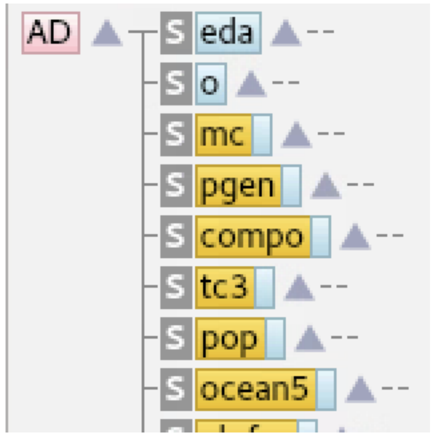
```
5_od3 (656176) ─ S limits (0) --
                 S admin (49) --
                 S ecompo_COPINT954 (2769) --
                 S compo_noncrit (90) --
                 S cams_noncrit (557) --
                 S cams_mirror (188) --
```



```
emos-0 (868050) ─ S ecompo_50r1 (2699) --
                  S eeda_50r1 (10878) --
                  S e_50r1 (8642) --
                  S aifs_single_esuite (100) --
                  S epproc_cpf (69) --
                  S eeda_49r1 (8007) --
                  S e_49r1 (9691) --
                  S emc_49r1 (159818) --
                  S etc3_49r1 (21051) --
                  S limits (0)
```



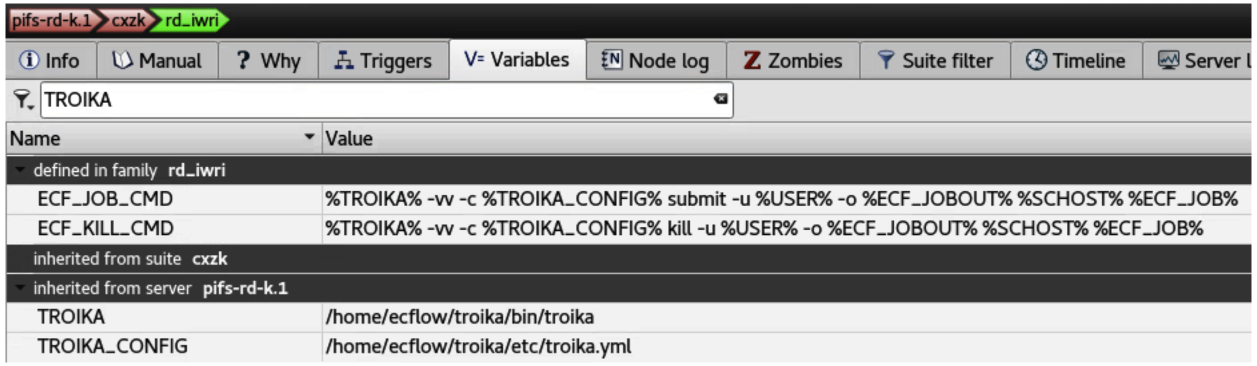
```
AA ─ S eda --
     S o --
     S mc --
     S pgen --
     S compo --
     S tc3 --
     S pop --
     S ocean5 --
```



```
AD ─ S eda --
     S o --
     S mc --
     S pgen --
     S compo --
     S tc3 --
     S pop --
     S ocean5 --
```

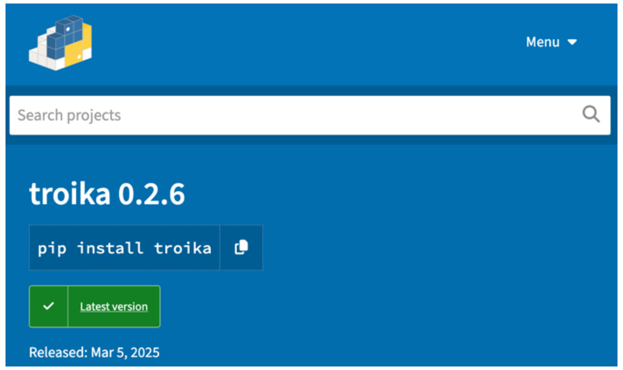
ecFlow: best practices - troika, a fine-tuning jobs submitter

- Troika is open-source, developed at ECMWF
- A system description with a Yaml file
- To Interact with remote queueing system
- Extra jobs tuning (MEM, THREADS, NPES)
- Run hooks (pre / post action)
- Allow deterministic + load balancing submit
- Troika is used in FD/RD/CD/MS workflows
- Extensible: connections (ssh, local), queuing system (Slurm, PBS, …), hooks
- https://github.com/ecmwf/troika


pifs-rd-k.1 > cxzk > rd_iwri
Info | Manual | Why | Triggers | Variables | Node log | Zombies | Suite filter | Timeline | Server l

TROIKA

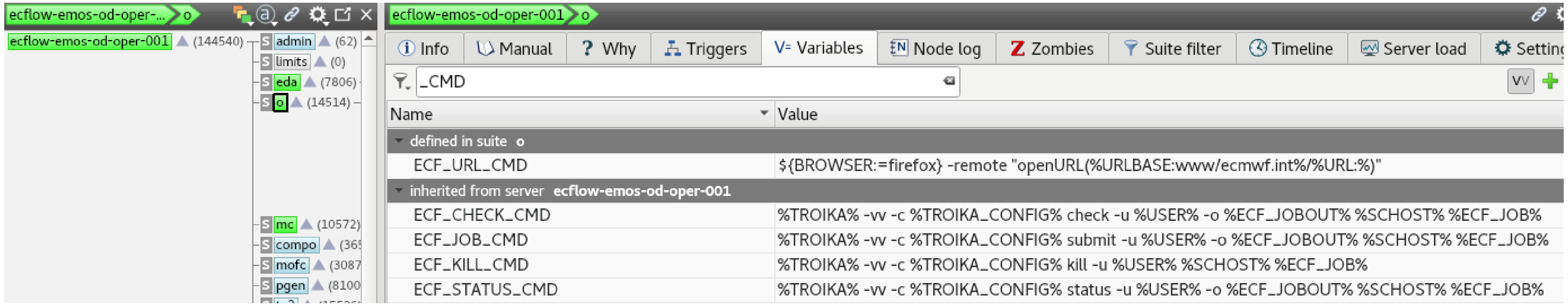
Name	Value
defined in family **rd_iwri**	
ECF_JOB_CMD	%TROIKA% -vv -c %TROIKA_CONFIG% submit -u %USER% -o %ECF_JOBOUT% %SCHOST% %ECF_JOB%
ECF_KILL_CMD	%TROIKA% -vv -c %TROIKA_CONFIG% kill -u %USER% -o %ECF_JOBOUT% %SCHOST% %ECF_JOB%
inherited from suite **cxzk**	
inherited from server **pifs-rd-k.1**	
TROIKA	/home/ecflow/troika/bin/troika
TROIKA_CONFIG	/home/ecflow/troika/etc/troika.yml



troika 0.2.6

pip install troika

Released: Mar 5, 2025

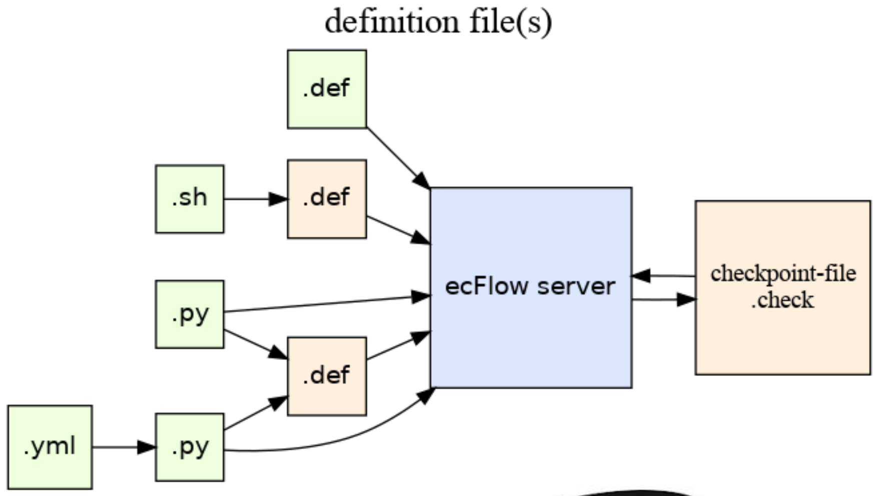

ecflow-emos-od-oper-... > o
ecflow-emos-od-oper-001 (144540)
S admin (62)
S limits (0)
S eda (7806)
S o (14514)
S mc (10572)
S compo (36
S mofc (3087
S pgen (8100
ecflow-emos-od-oper-001 > o
Info | Manual | Why | Triggers | Variables | Node log | Zombies | Suite filter | Timeline | Server load | Setting

_CMD

Name	Value
defined in suite **o**	
ECF_URL_CMD	${BROWSER:=firefox} -remote "openURL(%URLBASE:www/ecmwf.int%/%URL:%)"
inherited from server **ecflow-emos-od-oper-001**	
ECF_CHECK_CMD	%TROIKA% -vv -c %TROIKA_CONFIG% check -u %USER% -o %ECF_JOBOUT% %SCHOST% %ECF_JOB%
ECF_JOB_CMD	%TROIKA% -vv -c %TROIKA_CONFIG% submit -u %USER% -o %ECF_JOBOUT% %SCHOST% %ECF_JOB%
ECF_KILL_CMD	%TROIKA% -vv -c %TROIKA_CONFIG% kill -u %USER% %SCHOST% %ECF_JOB%
ECF_STATUS_CMD	%TROIKA% -vv -c %TROIKA_CONFIG% status -u %USER% -o %ECF_JOBOUT% %SCHOST% %ECF_JOB%

ecFlow: Best Practices – Suites' design

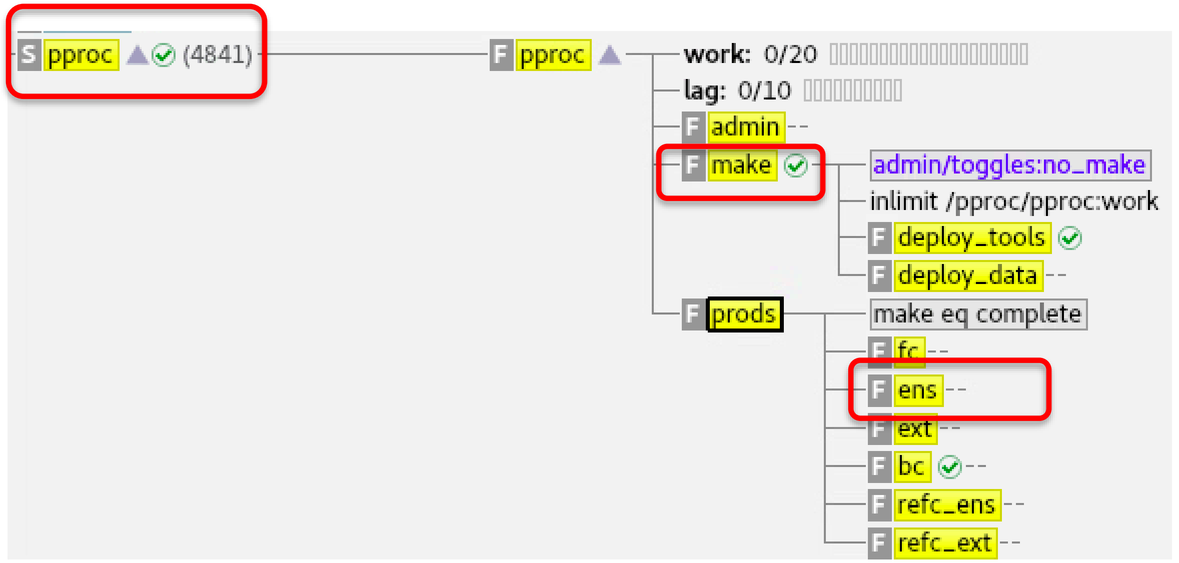
- KISS: Keep it simple
- Pure text definition-file may be all you need
  - Download an existing suite: ecflow_client –get
- Shell suite definition, aka 'stream like definition' may be enough
- Python API: when definition file is no longer needed, yet…
  - Functional programming: no temporary objects, list comprehension
  - We can load/replace a node directly
- Python API:
  - Native ecflow API, ecf API
  - pyflow: **Config as code design** with YAML file turned into a suite
  - Wellies: DRY-design suites from YAML config
  - PySuite (OOD)
  - Check jobs creations from client side
  - Simulate: validate there is no deadlock from suite definition



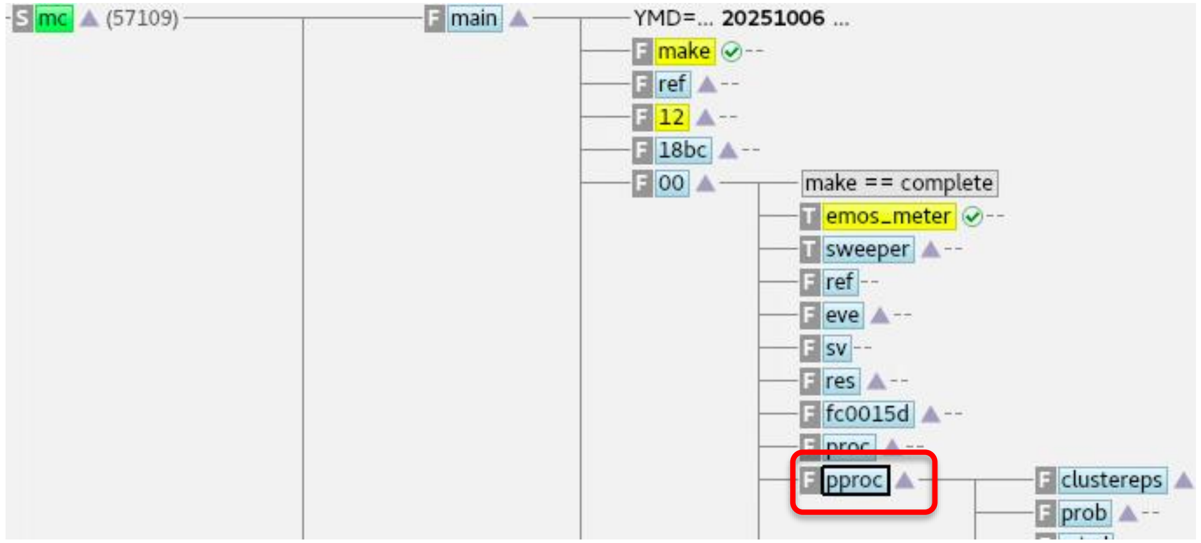
```
NAME = os.getenv("SUITE", "elearning")
DEFS = Defs()
SUITE = create(NAME)
DEFS.add_suite(SUITE)
DEFS.generate_scripts()
DEFS.simulate()
CLIENT = Client(os.getenv("ECF_HOST", "localhost"),
                os.getenv("ECF_PORT", 3141))
CLIENT.replace(f"/{NAME}", DEFS)
```

ecFlow: Best Practices – Suites' design - modularity

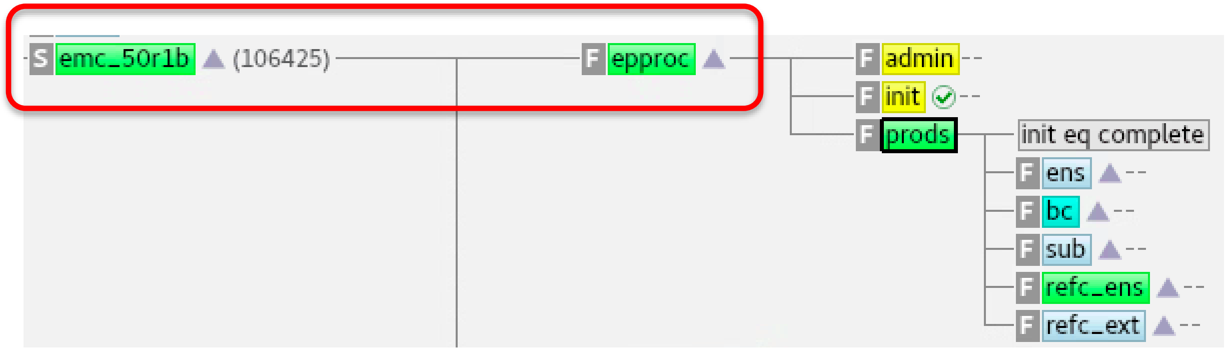
- **Modular design:**
  - Design as **suite**, as **standalone family** (repeat inside), as **embedded family** (inherited repeat, variables): ex. pproc node
  - Make/init/setup family v using binaries from a module
  - We can move suite from server to server once loaded
    - ecflow_client –plug (provided it is ok with ECF_FILES/INCUDE)
    - In fact, any node can be moved (! No active task within)



```
S pproc (4841) ── F pproc ──┬─ work: 0/20
                            ├─ lag: 0/10
                            ├─ F admin --
                            ├─ F make ──┬─ admin/toggles:no_make
                            │           ├─ inlimit /pproc/pproc:work
                            │           ├─ F deploy_tools
                            │           └─ F deploy_data --
                            └─ F prods ─┬─ make eq complete
                                        ├─ F fc --
                                        ├─ F ens --
                                        ├─ F ext --
                                        ├─ F bc --
                                        ├─ F refc_ens --
                                        └─ F refc_ext --
```



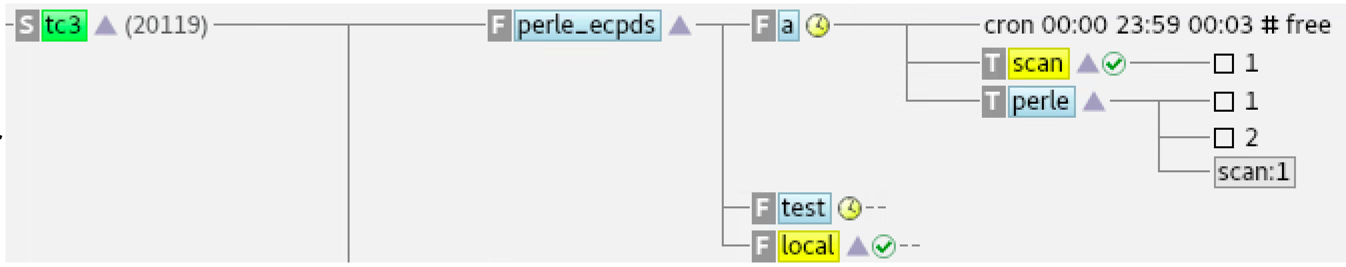
```
S mc (57109) ── F main ──┬─ YMD=... 20251006 ...
                         ├─ F make --
                         ├─ F ref --
                         ├─ F 12 --
                         ├─ F 18bc --
                         └─ F 00 ──┬─ make == complete
                                   ├─ T emos_meter --
                                   ├─ T sweeper --
                                   ├─ F ref --
                                   ├─ F eve --
                                   ├─ F sv --
                                   ├─ F res --
                                   ├─ F fc0015d --
                                   ├─ F proc --
                                   └─ F pproc ──┬─ F clustereps
                                                └─ F prob --
```



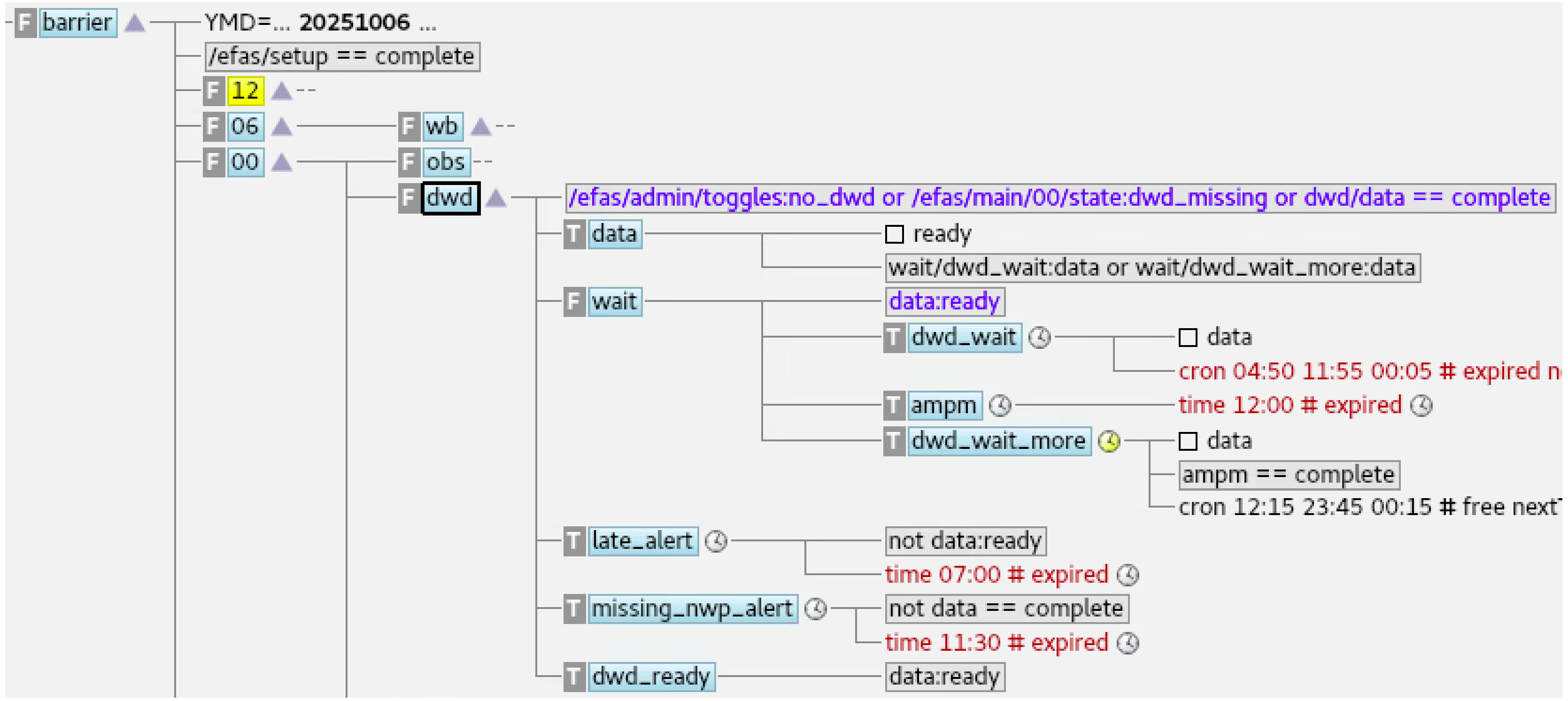
```
S emc_50r1b (106425) ── F epproc ──┬─ F admin --
                                   ├─ F init --
                                   └─ F prods ─┬─ init eq complete
                                               ├─ F ens --
                                               ├─ F bc --
                                               ├─ F sub --
                                               ├─ F refc_ens --
                                               └─ F refc_ext --
```

ecFlow: Best Practices – Suites' design – start processing

- **Waterfall** design vs **query** design
  - Event sent by third party to start processing + trigger
    - Ecpds can be configured to do that on file reception
  - **Barrier** family to query data availability in a time window
    - A combination of
      - Event, Trigger, Complete, Cron
    - It may raise multi-levels alarms
- ⚠ a time attribute?
- ✅ an aviso attribute



```
S tc3 (20119) ── F perle_ecpds ── F a ── cron 00:00 23:59 00:03 # free
                                     ├── T scan ── □ 1
                                     └── T perle ── □ 1
                                                 ├── □ 2
                                                 └── scan:1
                                  ├── F test --
                                  └── F local --
```



```
F barrier ── YMD=... 20251006 ...
          ├── /efas/setup == complete
          ├── F 12 --
          ├── F 06 ── F wb --
          └── F 00 ── F obs --
                   └── F dwd ── /efas/admin/toggles:no_dwd or /efas/main/00/state:dwd_missing or dwd/data == complete
                            ├── T data ── □ ready
                            │          └── wait/dwd_wait:data or wait/dwd_wait_more:data
                            ├── F wait ── data:ready
                            │          ├── T dwd_wait ── □ data
                            │          │              └── cron 04:50 11:55 00:05 # expired n
                            │          ├── T ampm ── time 12:00 # expired
                            │          └── T dwd_wait_more ── □ data
                            │                             ├── ampm == complete
                            │                             └── cron 12:15 23:45 00:15 # free next
                            ├── T late_alert ── not data:ready
                            │                └── time 07:00 # expired
                            ├── T missing_nwp_alert ── not data == complete
                            │                       └── time 11:30 # expired
                            └── T dwd_ready ── data:ready
```

ECMWF EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

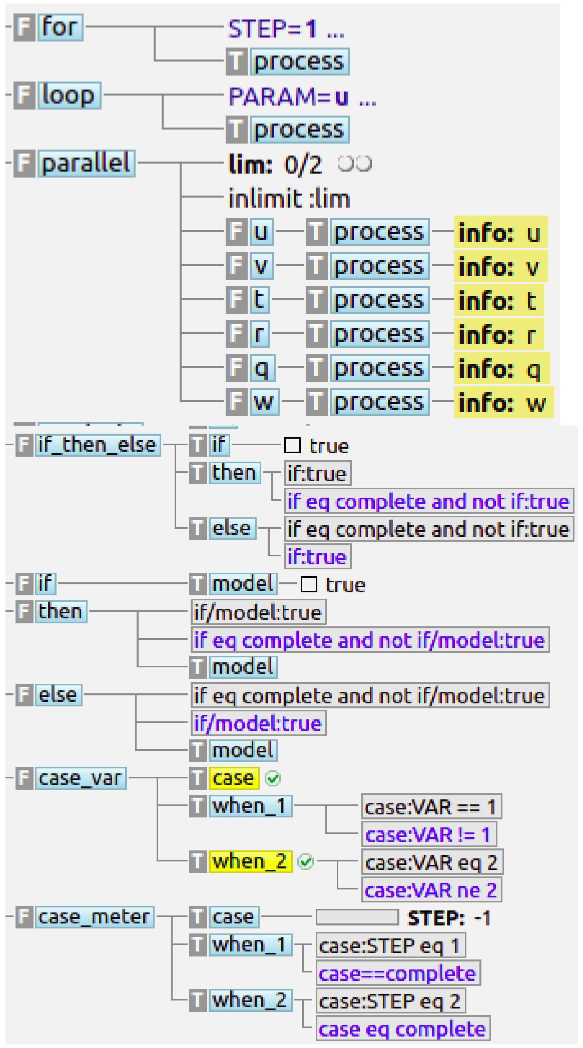
ecFlow: Best Practices – Suites' design – preliminary

- Once a preliminary suite is created:
  - Check jobs creations from client side
    - Detect issue with variables used in wrappers, yet may be undefined in the suite (or without default value)
    - Check all include files are present
    - Include loops can be avoided using %**includeonce**
- Simulate the suite from the client side
  - validate there is no deadlock from suite definition
- Check runtimes for the jobs
  - Find the right balance, number of parallel jobs vs user visible runtime
  - Use bin packing and/or gnu-parallel
- Add **limits** to prevent flooding with too many jobs
  - On top suite node: add **defstatus suspended**

ecFlow: Best Practices – Suites' design –a visual language

- Ecflow suite as a visual language (text definition example)
- ”can you understand easily” ?
  - Suite functionality
  - System solutions for suite handling / robustness
  - Vs a functionality hidden deep into a task
  - One wrapper called multiple time (if model)
  - Vs multiple wrappers
    - If then else block
- Case block with a
  - Rigid selector (defstatus)
  - vs dynamic branch selection
- For block
  - Inner loop
  - Repeat outer loop
  - ‘exploded’

```
family case_var
  task case
    defstatus complete
    edit VAR '1'
  task when_1
    complete case:VAR != 1
    trigger case:VAR == 1
  task when_2
    complete case:VAR ne 2
    trigger case:VAR eq 2
endfamily
family case_meter
  task case
    meter STEP -1 48 48
  task when_1
    complete case==complete
    trigger case:STEP eq 1
  task when_2
    complete case eq complete
    trigger case:STEP eq 2
endfamily
```



```
F for ──────────── STEP=1 ...
                   T process
F loop ─────────── PARAM=u ...
                   T process
F parallel ─────── lim: 0/2
                   inlimit :lim
                   F u ── T process ── info: u
                   F v ── T process ── info: v
                   F t ── T process ── info: t
                   F r ── T process ── info: r
                   F q ── T process ── info: q
                   F w ── T process ── info: w
F if_then_else ─ T if ──── ☐ true
                 T then ── if:true
                           if eq complete and not if:true
                 T else ── if eq complete and not if:true
                           if:true
F if ─────────── T model ─ ☐ true
F then ───────── if/model:true
                 if eq complete and not if/model:true
                 T model
F else ───────── if eq complete and not if/model:true
                 if/model:true
                 T model
F case_var ───── T case ✓
                 T when_1 ──── case:VAR == 1
                               case:VAR != 1
                 T when_2 ✓ ── case:VAR eq 2
                               case:VAR ne 2
F case_meter ─── T case ───── STEP: -1
                 T when_1 ─── case:STEP eq 1
                              case==complete
                 T when_2 ─── case:STEP eq 2
                              case eq complete
```

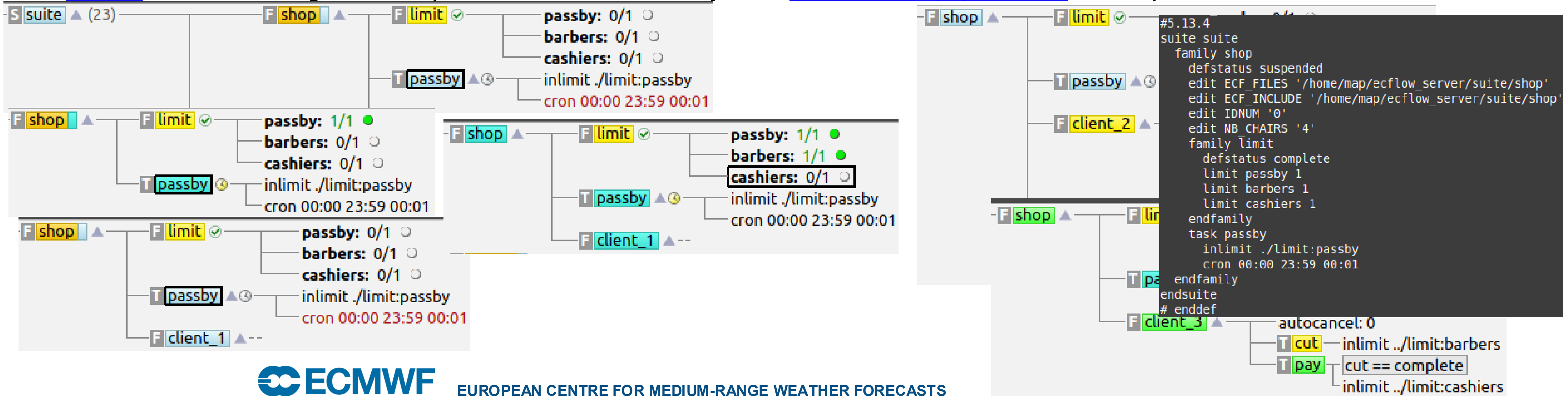
```
family if_then_else
  task if
    event 0 true
  task then
    complete if eq complete and not if:true
    trigger if:true
  task else
    complete if:true
    trigger if eq complete and not if:true
endfamily
family if
  task model
    event 0 true
endfamily
family then
  complete if eq complete and not if/model:true
  trigger if/model:true
  task model
endfamily
family else
  complete if/model:true
  trigger if eq complete and not if/model:true
  task model
endfamily
family for
  repeat integer STEP 1 240 3
  task process
endfamily
family loop
  repeat string PARAM "u" "v" "t" "r" "q" "w"
  task process
endfamily
```

ecFlow: Best Practices – Suites' design – dynamic suites

- **Iterative design**: a suite as a lively entity where we add new families and task, and prune old branches
  - Only consistency is required in the definition file
  - A suite can be defined from multiple definition files
  - Ecflow_client –replace <node> <server>
  - Ecflow_client –delete <node> <server>
  - Provided no active / submitted tasks lie below to prevent zombies

```
#!%SHELL:/bin/bash%
%manual
%end
%include <%QSUB_H:pure%>
%include <%HEAD_H:trap.h%>
cd ${TMPDIR:=/tmp}
IDNUM=%IDNUM:0%
ECF_INCLUDE=%ECF_INCLUDE%
export PYTHONPATH=${ECF_INCLUDE}:${PYTHONPATH:=}
python3 $ECF_INCLUDE/shop.py -n $((IDNUM+1))
%include <%TAIL_H:endt.h%>
```

Tasks can be designed to update their own family: the barber shop problem example



```
#5.13.4
suite suite
  family shop
    defstatus suspended
    edit ECF_FILES '/home/map/ecflow_server/suite/shop'
    edit ECF_INCLUDE '/home/map/ecflow_server/suite/shop'
    edit IDNUM '0'
    edit NB_CHAIRS '4'
    family limit
      defstatus complete
      limit passby 1
      limit barbers 1
      limit cashiers 1
    endfamily
    task passby
      inlimit ./limit:passby
      cron 00:00 23:59 00:01
  endfamily
endsuite
# enddef
```

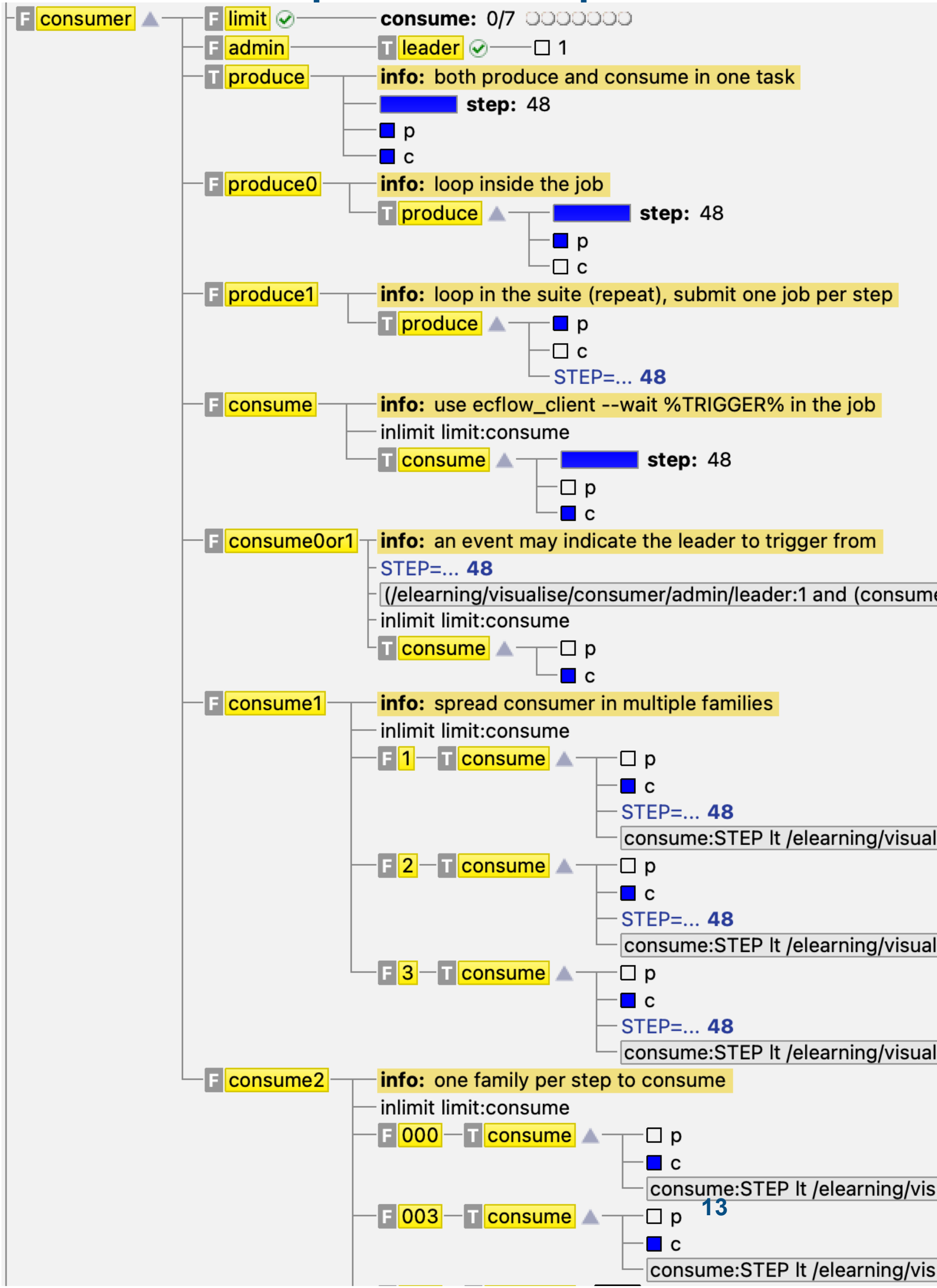
ECMWF EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

ecFlow: Best Practices – Suites' design –consumer-producer pattern

This example is part of the ecFlow eLearning repo
Among the multiple solutions we can find in ecFlow suites:

- a single task, do it all
  - Inner loop
  - Inner checkpointing to restart from previous state
- Separation of concern: producer vs consumer
  - Inner loop (produce0) vs outer loop (using repeat: produce1)

- Different approaches for a consumer:
  - We prefer to expand the loop with one family per step (consume2)
    - A limit to control the load
    - A failure won't affect another STEP
- It depends the project, the criticality, the available resources and support

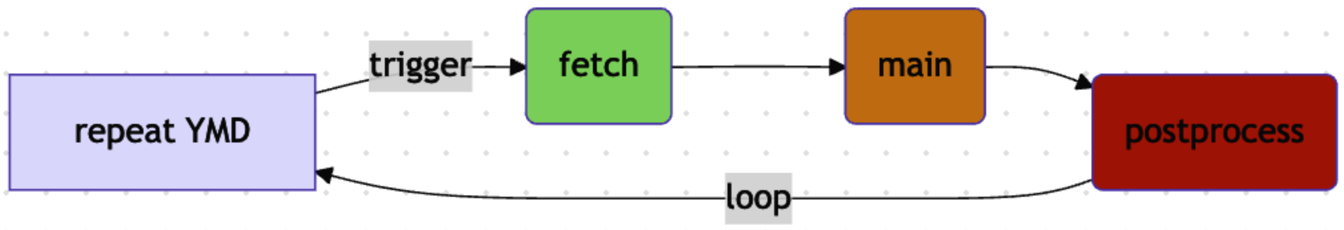


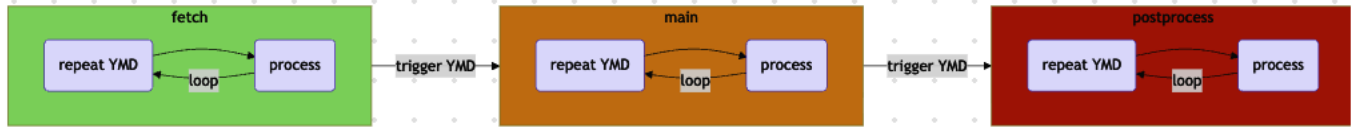
```
F consumer
  F limit        consume: 0/7
  F admin        T leader   □ 1
  T produce      info: both produce and consume in one task
                 step: 48
                 ■ p
                 ■ c
  F produce0     info: loop inside the job
                 T produce   step: 48
                             ■ p
                             □ c
  F produce1     info: loop in the suite (repeat), submit one job per step
                 T produce   ■ p
                             □ c
                             STEP=... 48
  F consume      info: use ecflow_client --wait %TRIGGER% in the job
                 inlimit limit:consume
                 T consume   step: 48
                             □ p
                             ■ c
  F consume0or1  info: an event may indicate the leader to trigger from
                 STEP=... 48
                 (/elearning/visualise/consumer/admin/leader:1 and (consume
                 inlimit limit:consume
                 T consume   □ p
                             ■ c
  F consume1     info: spread consumer in multiple families
                 inlimit limit:consume
                 F 1 — T consume   □ p
                                   ■ c
                                   STEP=... 48
                                   consume:STEP lt /elearning/visual
                 F 2 — T consume   □ p
                                   ■ c
                                   STEP=... 48
                                   consume:STEP lt /elearning/visual
                 F 3 — T consume   □ p
                                   ■ c
                                   STEP=... 48
                                   consume:STEP lt /elearning/visual
  F consume2     info: one family per step to consume
                 inlimit limit:consume
                 F 000 — T consume   □ p
                                     ■ c
                                     consume:STEP lt /elearning/vis
                 F 003 — T consume   □ p
                                     ■ c
                                     consume:STEP lt /elearning/vis
```

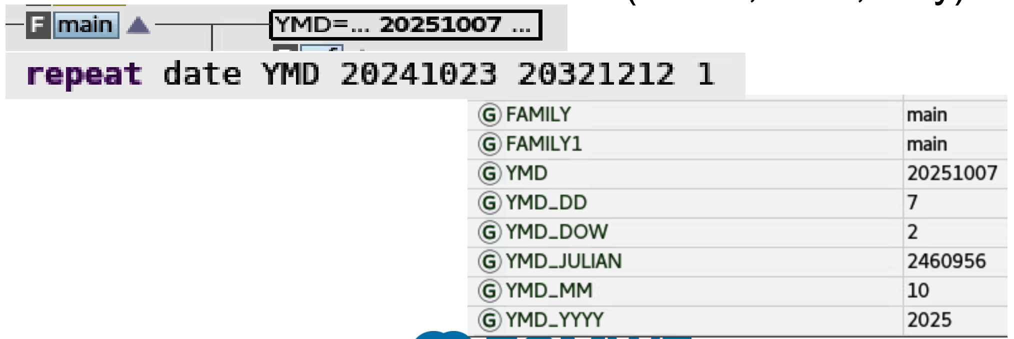
ECMWF EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

ecFlow: Best Practices – Suites' design - repeat

- **Cycling**: shared vs local repeat attribute
- In operations we try to **factorize** YMD
  - Possible in operation
  - Thanks to operators, to analysts to treat concerns
- **Loosely coupling** leads to more complex triggers.
  - Used in research, and few operational suites
  - fetch observations, process, archive
- Repeat
  - date / integer / enumerated / string / datelist
  - Extra variables for date (Julian, Dow, Doy)

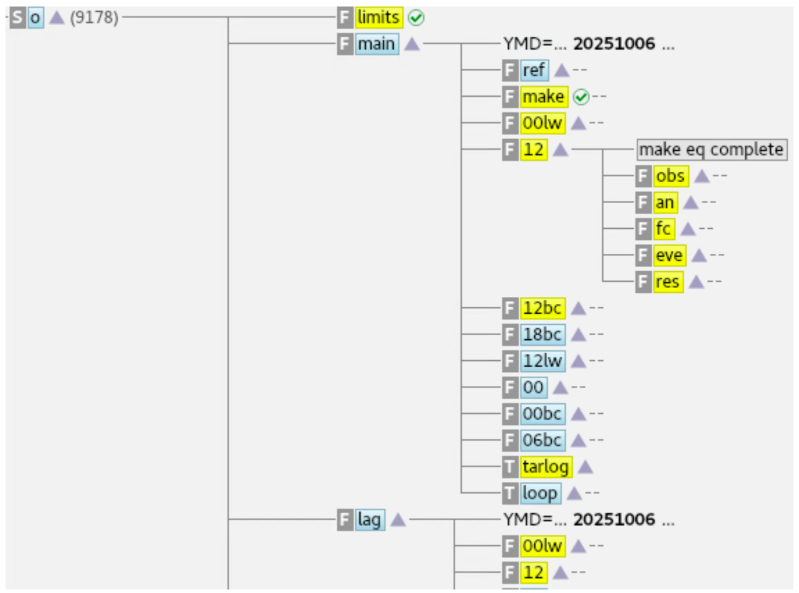




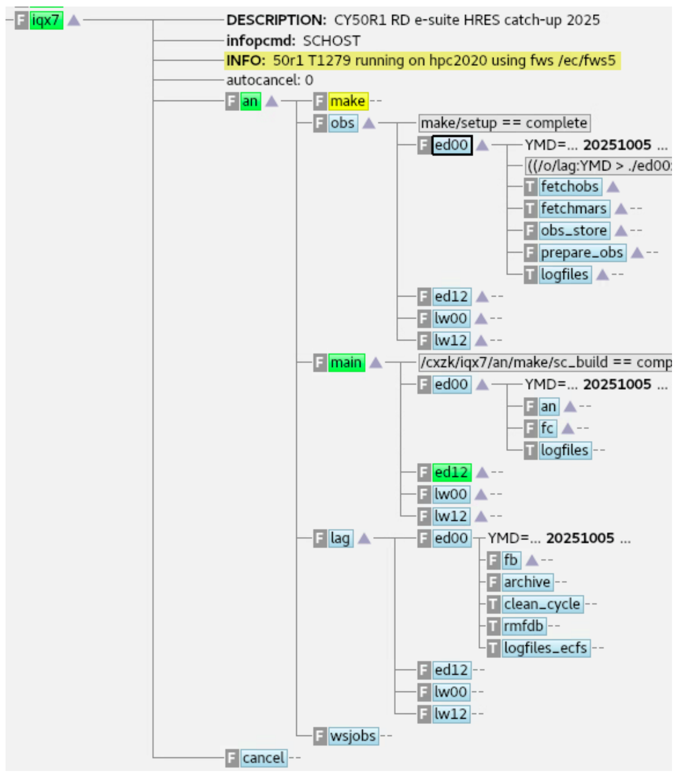


```
F main  YMD=... 20251007 ...
repeat date YMD 20241023 20321212 1
```

FAMILY	main
FAMILY1	main
YMD	20251007
YMD_DD	7
YMD_DOW	2
YMD_JULIAN	2460956
YMD_MM	10
YMD_YYYY	2025



```
S o (9178)
    F limits
    F main   YMD=... 20251006 ...
        F ref --
        F make --
        F 00lw --
        F 12   make eq complete
            F obs --
            F an --
            F fc --
            F eve --
            F res --
        F 12bc --
        F 18bc --
        F 12lw --
        F 00 --
        F 00bc --
        F 06bc --
        T tarlog
        T loop --
    F lag   YMD=... 20251006 ...
        F 00lw --
        F 12 --
```

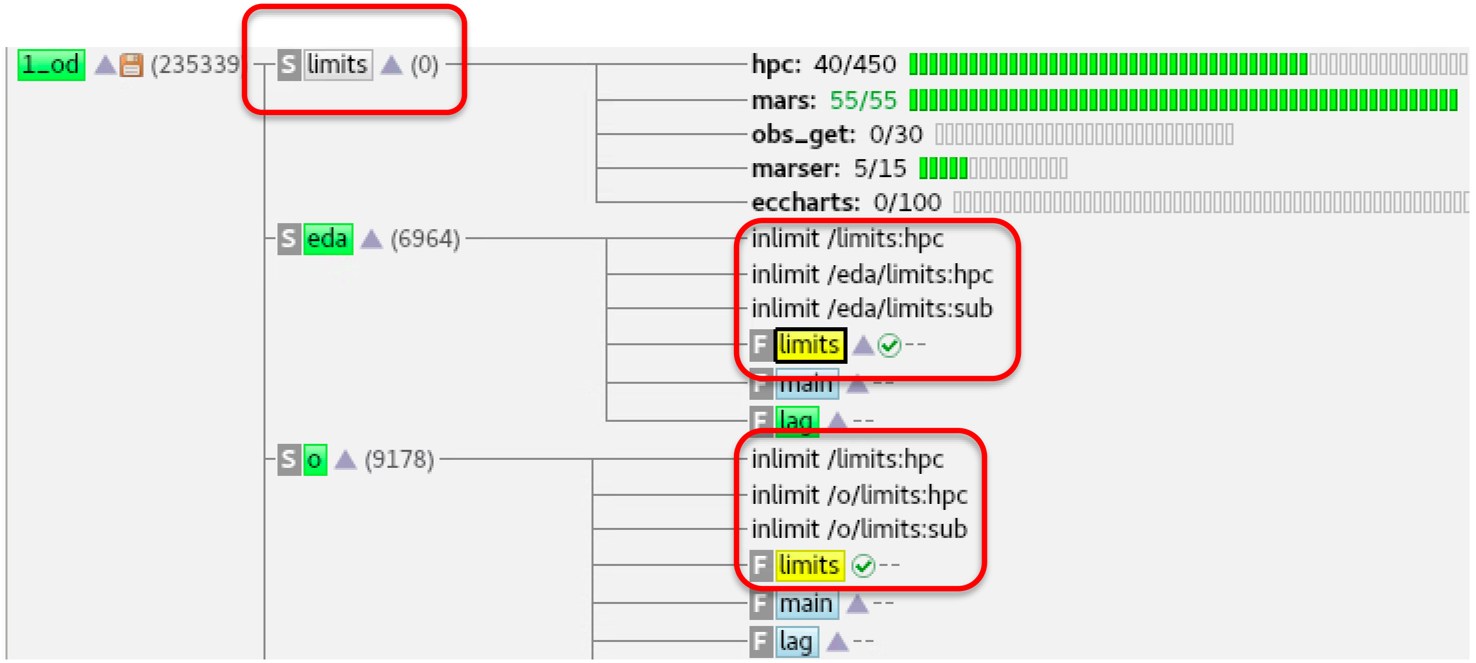


```
F iqx7
    DESCRIPTION: CY50R1 RD e-suite HRES catch-up 2025
    infopcmd: SCHOST
    INFO: 50r1 T1279 running on hpc2020 using fws /ec/fws5
    autocancel: 0
    F an
        F make --
        F obs   make/setup == complete
            F ed00   YMD=... 20251005 ...
                ((/o/lag:YMD > ./ed00
                T fetchobs
                T fetchmars --
                F obs_store --
                F prepare_obs --
                T logfiles --
            F ed12 --
            F lw00 --
            F lw12 --
        F main   /cxzk/iqx7/an/make/sc_build == comp
            F ed00   YMD=... 20251005 ...
                F an --
                F fc --
                T logfiles --
            F ed12 --
            F lw00 --
            F lw12 --
        F lag
            F ed00   YMD=... 20251005 ...
                F fb --
                F archive --
                T clean_cycle --
                T rmfdb --
                T logfiles_ecfs --
            F ed12 --
            F lw00 --
            F lw12 --
        F wsjobs --
    F cancel --
```

ecFlow: Best Practices – Suites' design - limits

– Resources: limit / inlimit / inlimit –s

- Inlimit –s (tasks in submit state only) can refrain submission bursts
- List all active/submit tasks under the limit
- Set limit 0 from the GUI as a 'distributed one click suspend'



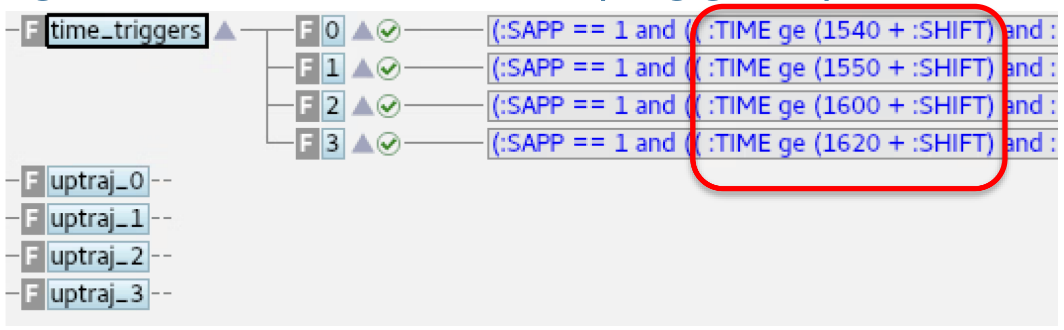
```
1_od (235339) ── S limits (0) ── hpc: 40/450
                                 mars: 55/55
                                 obs_get: 0/30
                                 marser: 5/15
                                 eccharts: 0/100
              ── S eda (6964) ── inlimit /limits:hpc
                                 inlimit /eda/limits:hpc
                                 inlimit /eda/limits:sub
                                 F limits --
                                 F main --
                                 F lag --
              ── S o (9178) ──── inlimit /limits:hpc
                                 inlimit /o/limits:hpc
                                 inlimit /o/limits:sub
                                 F limits --
                                 F main --
                                 F lag --
```

ecFlow: Best Practices – Suites' design – dependencies (triggers)

Group time dependencies in dedicated families + triggers

Easy replacement when schedule changes

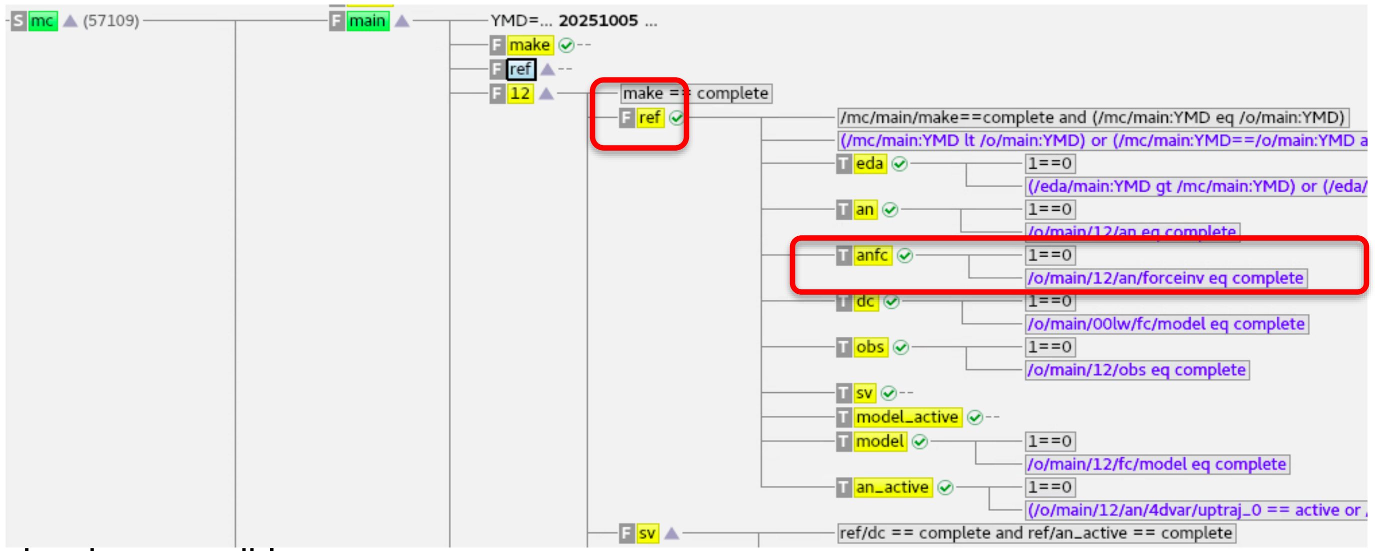
Defstatus complete in catchup-mode



```
time_triggers
  0   (:SAPP == 1 and ( :TIME ge (1540 + :SHIFT) and :
  1   (:SAPP == 1 and ( :TIME ge (1550 + :SHIFT) and :
  2   (:SAPP == 1 and ( :TIME ge (1600 + :SHIFT) and :
  3   (:SAPP == 1 and ( :TIME ge (1620 + :SHIFT) and :
uptraj_0 --
uptraj_1 --
uptraj_2 --
uptraj_3 --
```

Dependencies: simple and explicit triggers (producer-consumer)

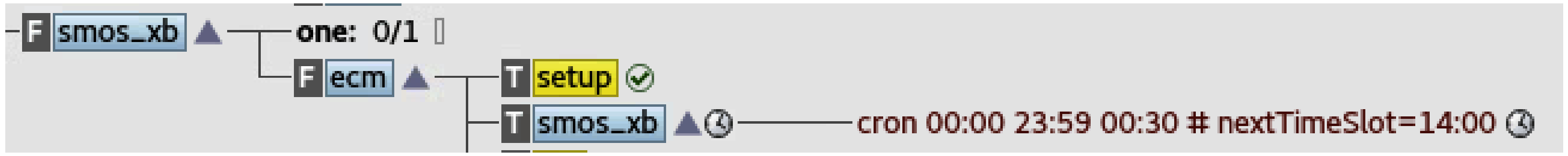
- Triggers can use node states, variables, limits, events, meters, late
- Expressions can use function cal::date_to_julian(/path/to:variable)
- **Group external triggers** in dedicated families
  - Dummy tasks easily replaced
  - Set defstatus complete in standalone mode
- 'fake task' to collect complex triggers:
  - **Trigger impossible** + complete attribute
  - Can be replaced / updated
  - Can be set 'defstatus complete'
  - **'trigger by transitivity**'
- **umbrella triggers**: factorise trigger on parent node when possible
- Absolute vs relative path? Rigid vs fit for 'move family'



```
mc (57109)
  main    YMD=... 20251005 ...
    make --
    ref --
    12
      make == complete
      ref   /mc/main/make==complete and (/mc/main:YMD eq /o/main:YMD)
            (/mc/main:YMD lt /o/main:YMD) or (/mc/main:YMD==/o/main:YMD a
        eda           1==0
                      (/eda/main:YMD gt /mc/main:YMD) or (/eda/
        an            1==0
                      /o/main/12/an eq complete
        anfc          1==0
                      /o/main/12/an/forceinv eq complete
        dc            1==0
                      /o/main/00lw/fc/model eq complete
        obs           1==0
                      /o/main/12/obs eq complete
        sv --
        model_active --
        model         1==0
                      /o/main/12/fc/model eq complete
        an_active     1==0
                      (/o/main/12/an/4dvar/uptraj_0 == active or
      sv    ref/dc == complete and ref/an_active == complete
```

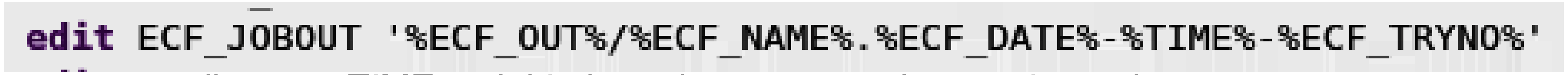
ecFlow: Best Practices – Suites' design – Cron

- We refrain from Cron in operation
  - ”a Cron task never ends”
  - Not compatible with upper repeat attribute, unless … it is managed like with the Barrier pattern
  - A meter, event is reset on the requeue at completion, beware



```
F smos_xb ▲ ─┬ one: 0/1
             └ F ecm ▲ ─┬ T setup ✓
                        └ T smos_xb ▲ ◷ ──── cron 00:00 23:59 00:30 # nextTimeSlot=14:00 ◷
```

- Yet sometimes a Cron is just enough:
  - Error handling: It has to be a very robust task. It would block on abort. designed 'not to fail' (send mail, external QC)
  - Process accumulated input in the interval
  - Minimum/NO operators action requested
  - Can behave on ECF_TRYNO (job occurrence number) value
  - Drawbacks of the Cron:
    - Output would be overwritten on next occurrence: an example where it is worth overwriting ecflow default variable

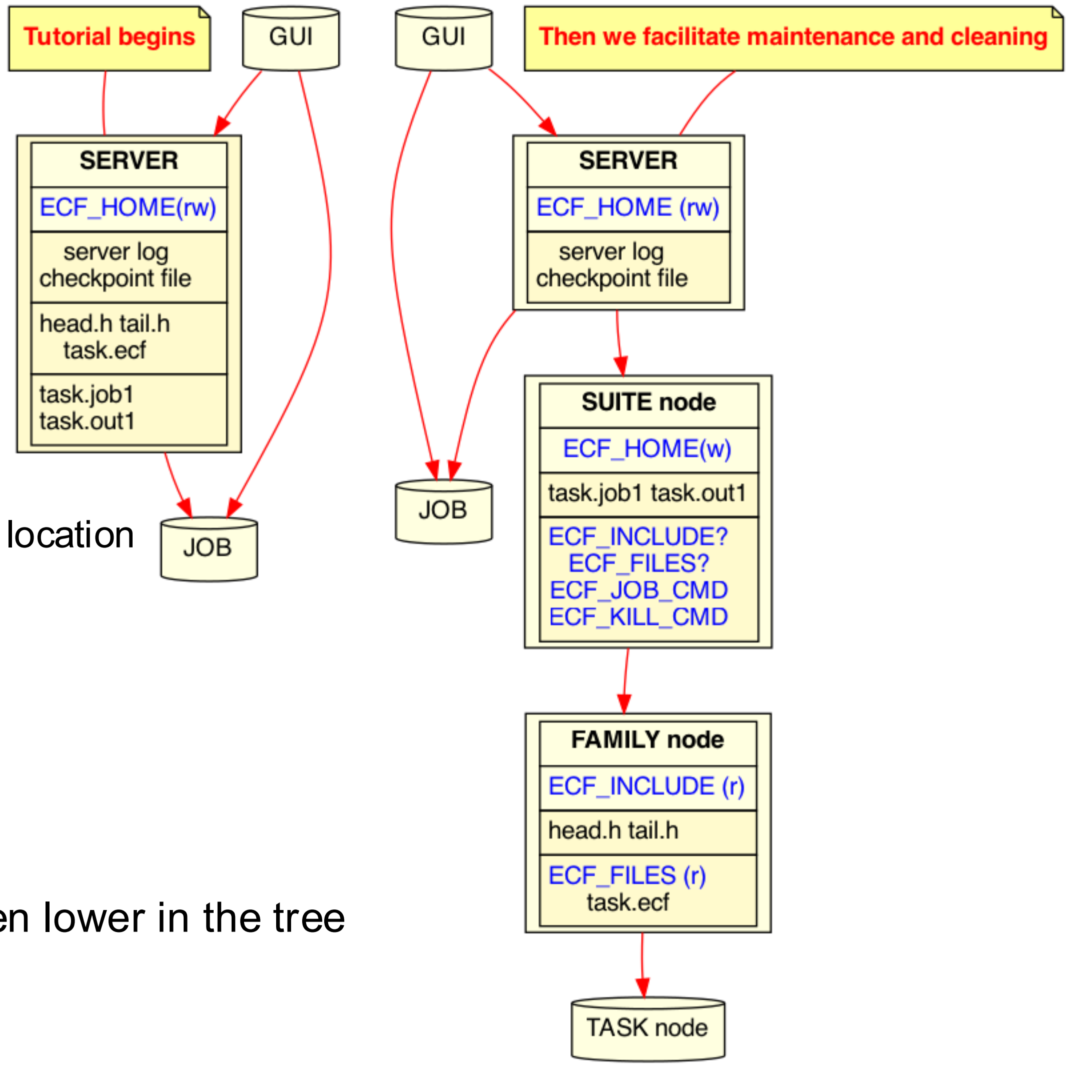


```
edit ECF_JOBOUT '%ECF_OUT%/%ECF_NAME%.%ECF_DATE%-%TIME%-%ECF_TRYNO%'
```

- Use date/time + repeat attribute or :TIME variable in a trigger expression as alternatives
- Cron is fit for simple administrative tasks

ecFlow: Best Practices, files locations, ECF_FILES, ECF_INCLUDE, ECF_HOME

- In the tutorial, ECF_HOME is used for everything:
  - Server logs, checkpoint
  - Tasks wrappers and headers
  - Job and output files
- Later:
  - Use ECF_FILES, ECF_INCLUDE variables (-r):
    - Can be refined lower in the tree for tasks templates and headers' location
    - Store wrappers/headers in read only access
  - ECF_HOME: where job files are (-w)
    - Where local outputs are by default (when ECF_OUT is absent)
  - ECF_OUT: preliminary path for remote jobs outputs
  - ECF_*_CMD variables: can be defined on top, and overwritten lower in the tree



ecFlow: Best Practices – Suites' design – ECF_MICRO

- default micro is %
- it can be changed **globally** from the definition file:

  edit ECF_MICRO $ # change to dollar / .def

helpful when using multiple languages in a suite

helpful when using script from different teams with different convention

- ECF_MICRO can be changed **locally** (and reverted)

  in the wrapper/headers using

  %ecfmicro ~

  ~ecfmicro %

- Change it in specific deliberate, documented cases
- Or **stick to simplicity using %**

```
#!%SHELL:/bin/bash%
%manual
...
%end
%include <%QSUB_H:pure%>
%include <%HEAD_H:trap.h%>
%include <%BODY_H:pure%>
%ecfmicro ^
# block
^ecfmicro %
%include <%TAIL_H:endt.h%>
```

ecFlow: Best Practices – Suites' design – ECF_EXTN

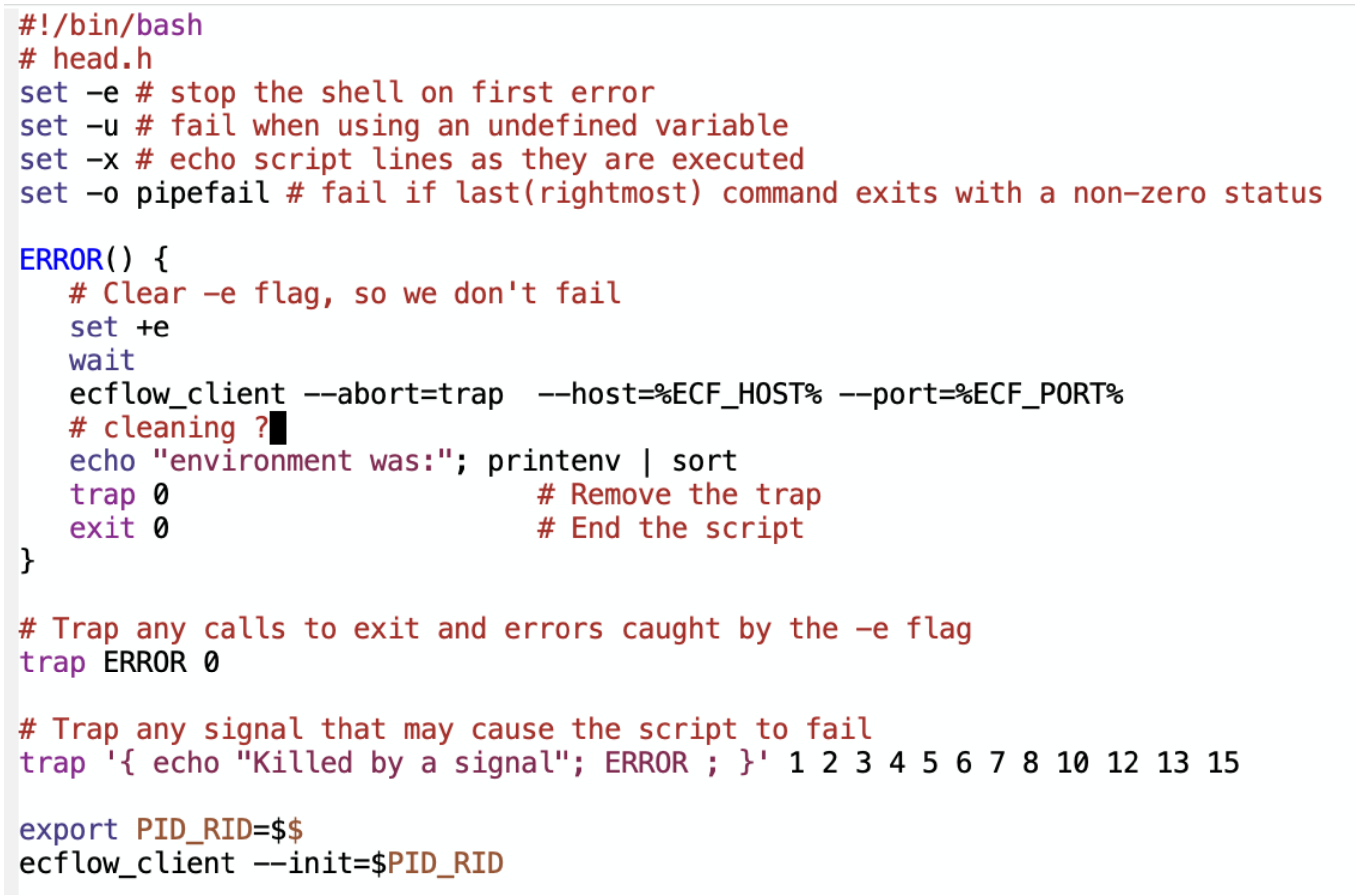
A variable ECF_EXTN: the task wrapper extension

- default is .ecf
- it can be changed in the definition file
  - ex: edit ECF_EXTN '.epy'
- useful when a suite uses multiple languages
- even for shell wrappers it may facilitate a transition (ksh to bash)
- we can design a pure shell/python script to be a valid task wrapper: edit ECF_EXTN .sh
  - (when there is no need for %include)
- we can distinguish source-controlled wrapper (.ecf) vs generated wrappers (.ecg)
- we can extend this practice
  - to distinguish template headers (.h) vs pure shell include (.sh)
  - where no preprocessing is expected (%includenopp <setup.sh>)

```
#!/usr/bin/env $SHELL:python3$
$include <$QSUB_H:pure$>
$include <$HEAD_H:head.epy$>
$includenopp <$SCRIPT:pure$>
$include <$TAIL_H:tail.py$>
```

ecFlow: Best Practices – Wrappers design trapping

- **Robust** jobs: designer duty is to preserve **trapping** (handle early exit and/or external signals reception)
  - Beware shell functions definitions with ksh:
    - x() { cmd; }
    - function x {cmd; }
      - in ksh, this will need to reload trapping
      - %include <trap.h> … trap 0
  - Or use bash

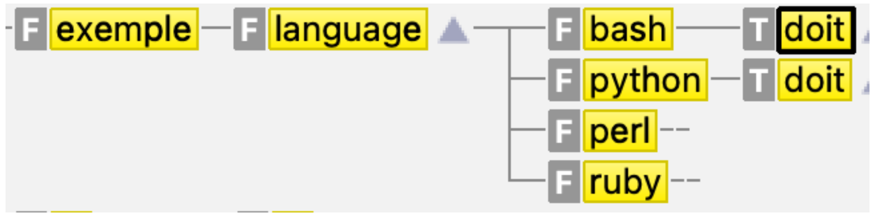


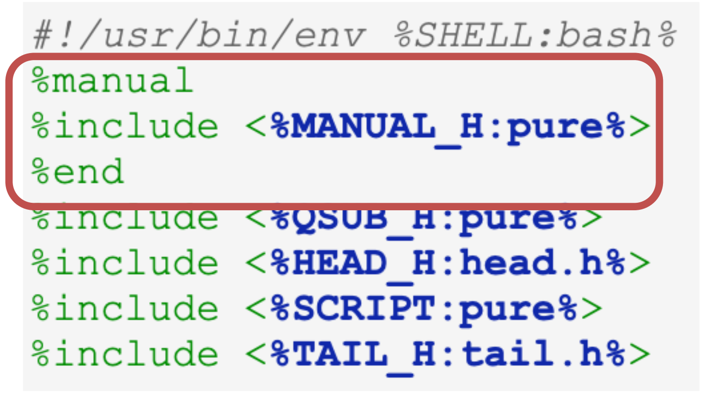
```
#!/bin/bash
# head.h
set -e # stop the shell on first error
set -u # fail when using an undefined variable
set -x # echo script lines as they are executed
set -o pipefail # fail if last(rightmost) command exits with a non-zero status

ERROR() {
   # Clear -e flag, so we don't fail
   set +e
   wait
   ecflow_client --abort=trap  --host=%ECF_HOST% --port=%ECF_PORT%
   # cleaning ?
   echo "environment was:"; printenv | sort
   trap 0                          # Remove the trap
   exit 0                          # End the script
}

# Trap any calls to exit and errors caught by the -e flag
trap ERROR 0

# Trap any signal that may cause the script to fail
trap '{ echo "Killed by a signal"; ERROR ; }' 1 2 3 4 5 6 7 8 10 12 13 15

export PID_RID=$$
ecflow_client --init=$PID_RID
```

- set –eux -o pipefail
  - Fail on error immediately (report abort)
  - Early exit must be trapped (trap ERROR 0)
  - Fail on undefined variables
  - Verbose mode logging each command
  - Set –o pipefail does NOT export in subshell
  - Refrain from using ssh inside, dedicate a task submitted on the other host (or use -o ExitOnForwardFailure=yes)
- **Consistency** and **Maintainability**: same head.h among suite tasks

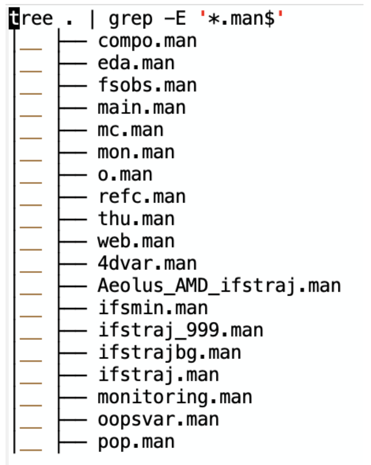
ecFlow: Best Practices – Wrappers design - Manual

- Task documentation: block %manual … %end
  - In the wrapper
  - In any included (pre-processed) script
  - We can delegate a suite variable for the filename for the manual
- A file <node>.man can provide a manual page for a suite or a family node
  - This can be a static or dynamic file
- ECF_URL_CMD is ‘interpreted CMD variable’ to open an URL from GUI
- What, why, When (delay), Where (information), Who
  - Title
  - Description
  - Input/output/variables
  - Procedures
  - Contact points

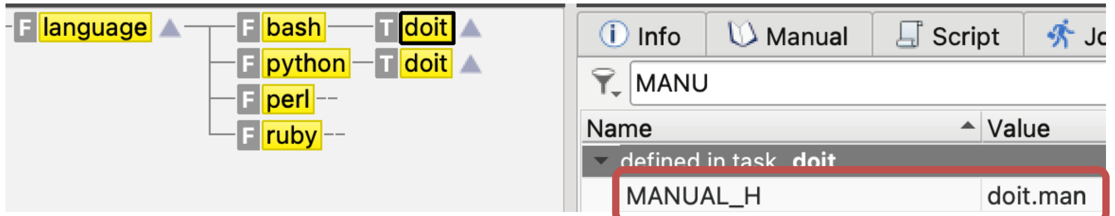




```
#!/usr/bin/env %SHELL:bash%
%manual
%include <%MANUAL_H:pure%>
%end
%include <%QSUB_H:pure%>
%include <%HEAD_H:head.h%>
%include <%SCRIPT:pure%>
%include <%TAIL_H:tail.h%>
```

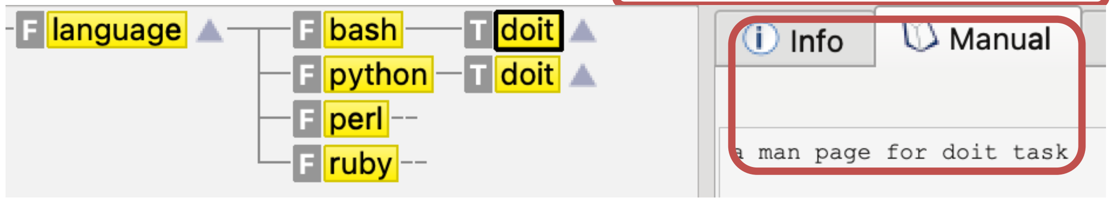


```
tree . | grep -E '*.man$'
├── compo.man
├── eda.man
├── fsobs.man
├── main.man
├── mc.man
├── mon.man
├── o.man
├── refc.man
├── thu.man
├── web.man
├── 4dvar.man
├── Aeolus_AMD_ifstraj.man
├── ifsmin.man
├── ifstraj_999.man
├── ifstrajbg.man
├── ifstraj.man
├── monitoring.man
├── oopsvar.man
├── pop.man
```



MANU

Name	Value
defined in task doit	
MANUAL_H	doit.man



a man page for doit task

ecFlow: Best Practices – Wrappers design - Rerun ability

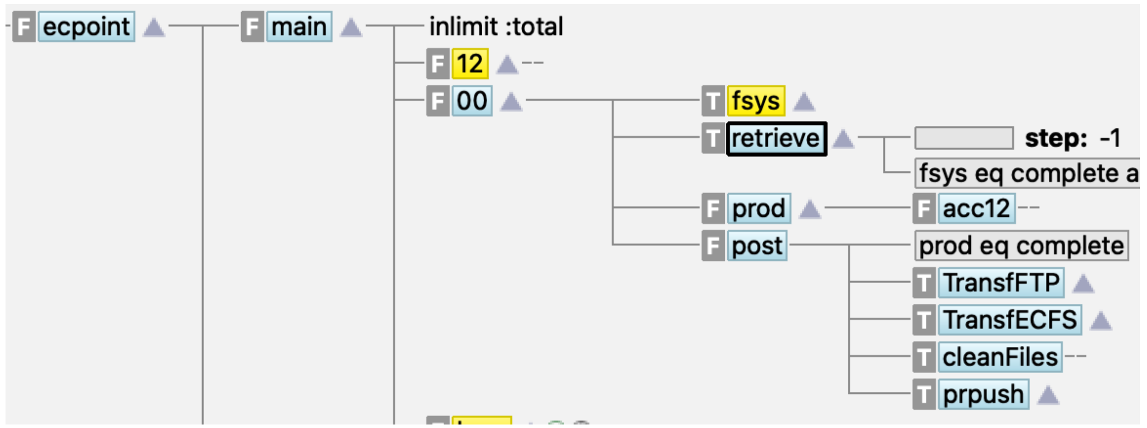
- **Idempotent**: ”a task shall be rerunnable” as part of a robust design
  - Clean ground rerun mode for most tasks
  - Validated in pre-operational mode
- Multiple views: Retrieve – compute – postprocess - push pattern
  - May be one task in development
  - One task per function in operation
- **Checkpointing** for tasks?
  - Minimise rerun time (at the price of changing ETA)
  - Clean ground rerun mode by default vs ‘thanks to a variable’
- Rerun mode: suites variables ECF_TRIES (total attempts) ECF_TRYNO (current occurrence)
  - Set ECF_TRIES to 1 for immediate abort, or increase it to allow automatic re-submit after abort
  - ECF_TRYNO is current try-number: jobs can be ‘self-aware’
  - Jobs can change behaviour: verbosity, debug mode, silent mode, mail, contact alert system

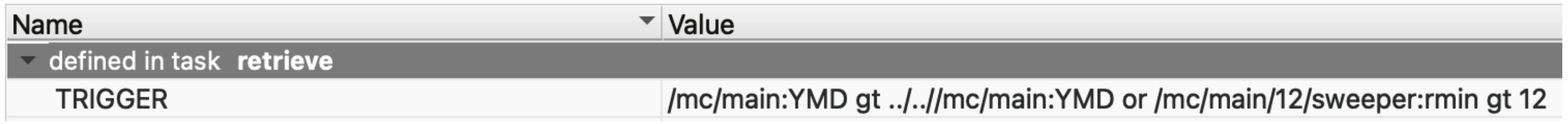
ecFlow: Best Practices – Wrappers design – FAMILY1 TASK

- Anti-pattern ✗
- Refrain from using test in FAMILY, FAMILY1, TASK
  - A dedicated suite variable is better on the long term
- Separation of concern:
  - Running a task vs configuring a task
- Such script would be a generic script,
  - asking for links to provide the different tasks wrappers
  - and more maintenance (rather that focussing on the suite definition)

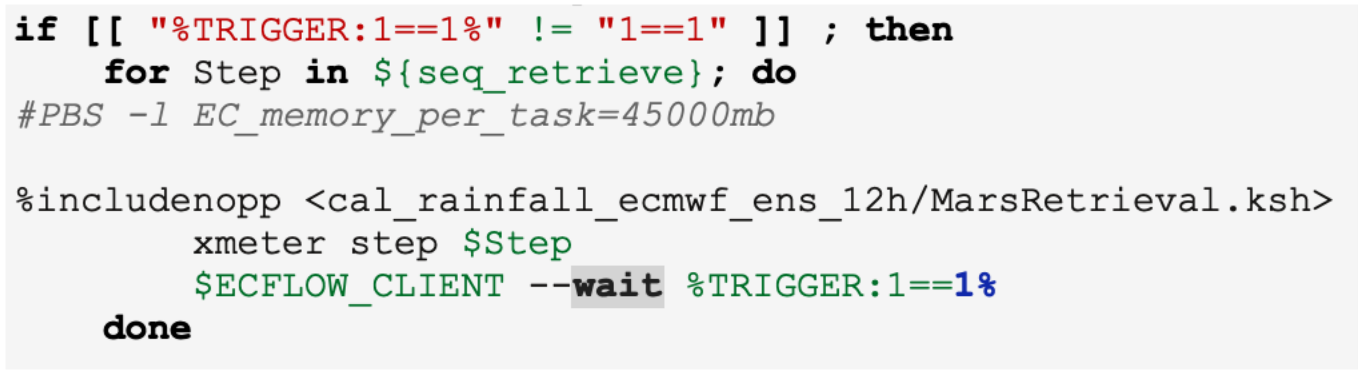
ecFlow: Best Practices – ecflow_client –wait <expression>

- A child command to query a trigger expression from the server to continue
- Aka ’a trigger in a job’
- Blocking: it may cause **live lock**
- **non blocking alternative:** use query
  - not a child command
  - status=$(ecflow_client –query <node>)





Name	Value
defined in task **retrieve**	
TRIGGER	/mc/main:YMD gt ../..//mc/main:YMD or /mc/main/12/sweeper:rmin gt 12



```
if [[ "%TRIGGER:1==1%" != "1==1" ]] ; then
    for Step in ${seq_retrieve}; do
#PBS -l EC_memory_per_task=45000mb

%includenopp <cal_rainfall_ecmwf_ens_12h/MarsRetrieval.ksh>
        xmeter step $Step
        $ECFLOW_CLIENT --wait %TRIGGER:1==1%
    done
```

ecFlow: Best Practices – Wrappers design - Cleaning

- Work in WDIR / TMPDIR
  - Own job cleaning vs admin cleaning task vs system cleaning
  - ecflow_client –complete is not the end
    - Complete for ecflow
    - Cleaning may occur after complete was sent
- In conjunction with **monitoring** tools
  - detect trends, or infringed thresholds
- **Retention** policy:
  - Cleaning can use DELTA_DAY, to keep few days online
  - **Check** validity of archived data before cleaning
  - Move away data from **visibility** before cleaning
  - Roll back capability?
  - documentation
- Test cleaning in **esuite mode**

ecFlow: Best Practices – Monitoring & Observability

Collect

- ecFlow Server log
- Job log
- from a job to write in the server's log: ecflow_client –msg <text>
- Metrics (Json)

Aggregate

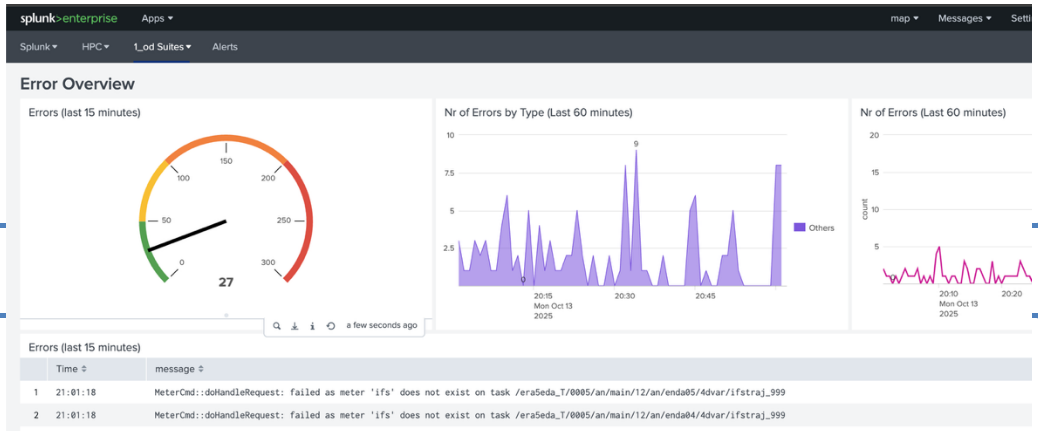
- Telegraf
- Prometheus
- Influx DB

Visualise

- Grafana
- Splunk
- Dashboards

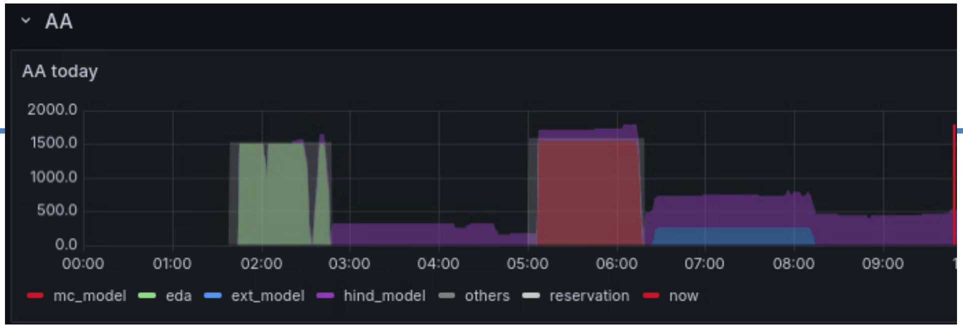
Alert

Opsview
Mail
chat



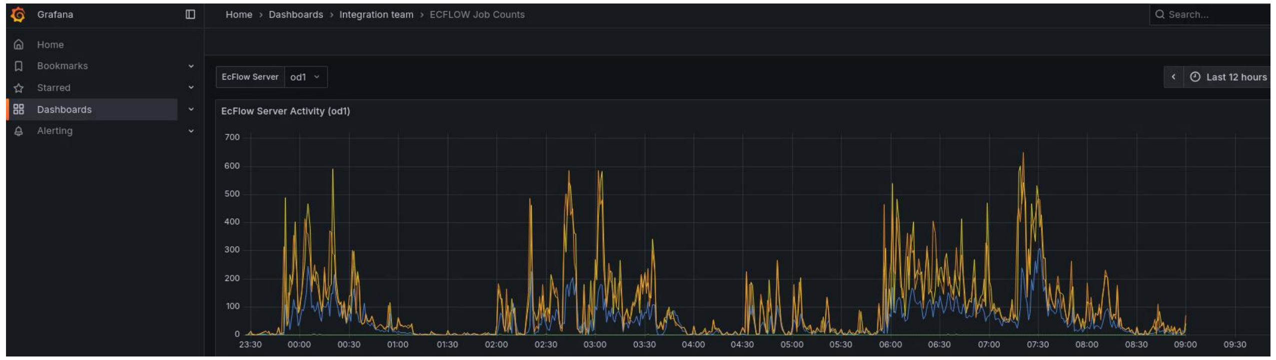
Errors (last 15 minutes)

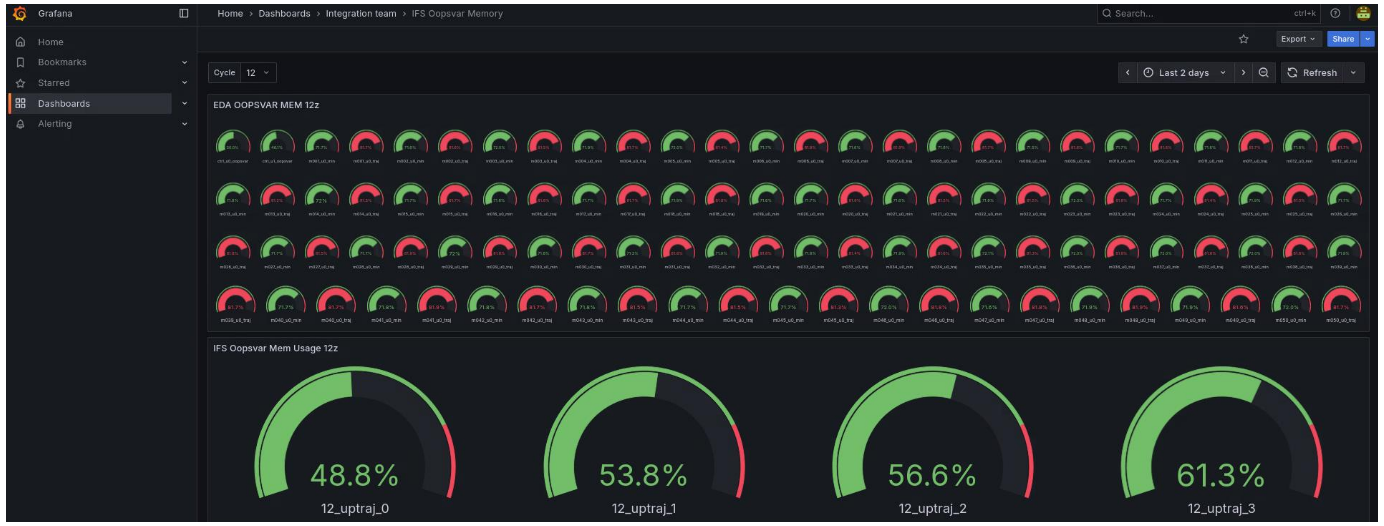
	Time	message
1	21:01:18	MeterCmd::doHandleRequest: failed as meter 'ifs' does not exist on task /era5eda_T/0005/an/main/12/an/enda05/4dvar/ifstraj_999
2	21:01:18	MeterCmd::doHandleRequest: failed as meter 'ifs' does not exist on task /era5eda_T/0005/an/main/12/an/enda04/4dvar/ifstraj_999



ecFlow: Best Practices – Monitoring & Observability

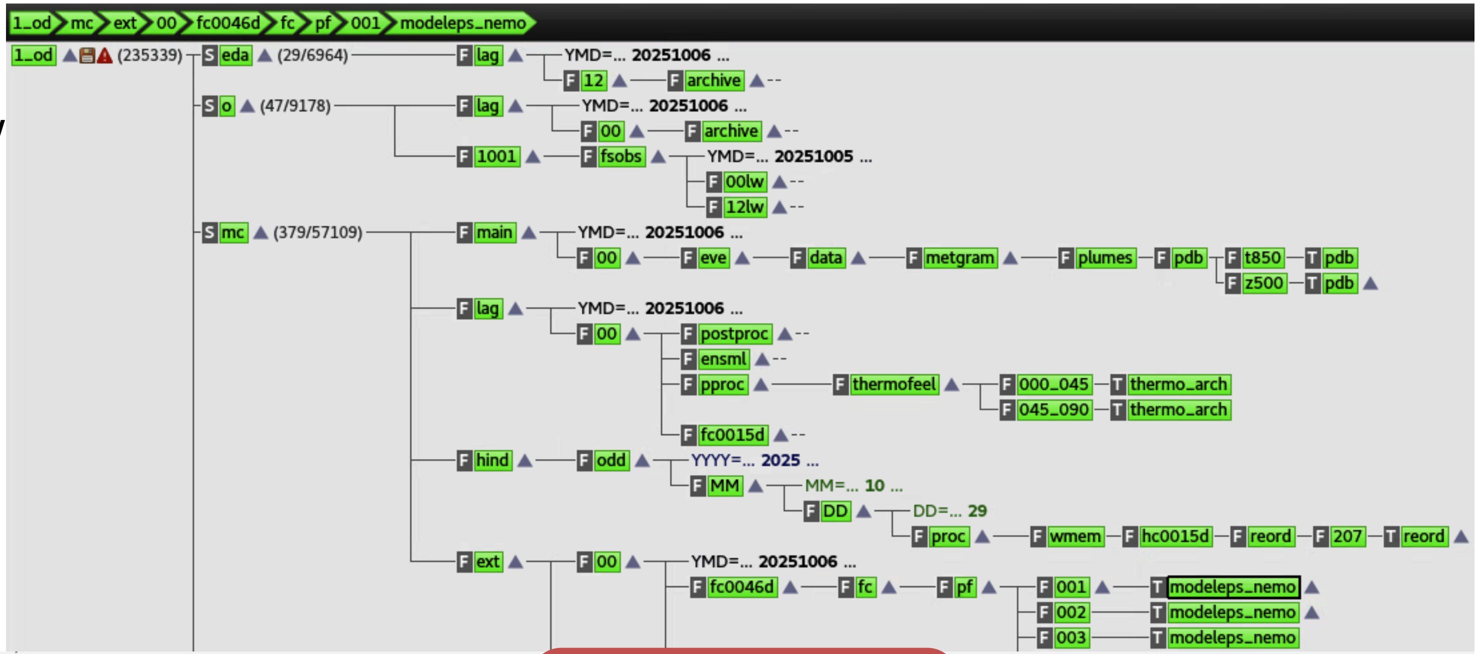
- Contact the server server:
  - ecflow_client –ping
  - ecflow_client –info
  - ecflow_client –query <node>
  - ecflow_client –get
    - to dump server content, may offer status when log is not accessible
- Or better use the **checkpoint** file than hammering a server with too frequent queries

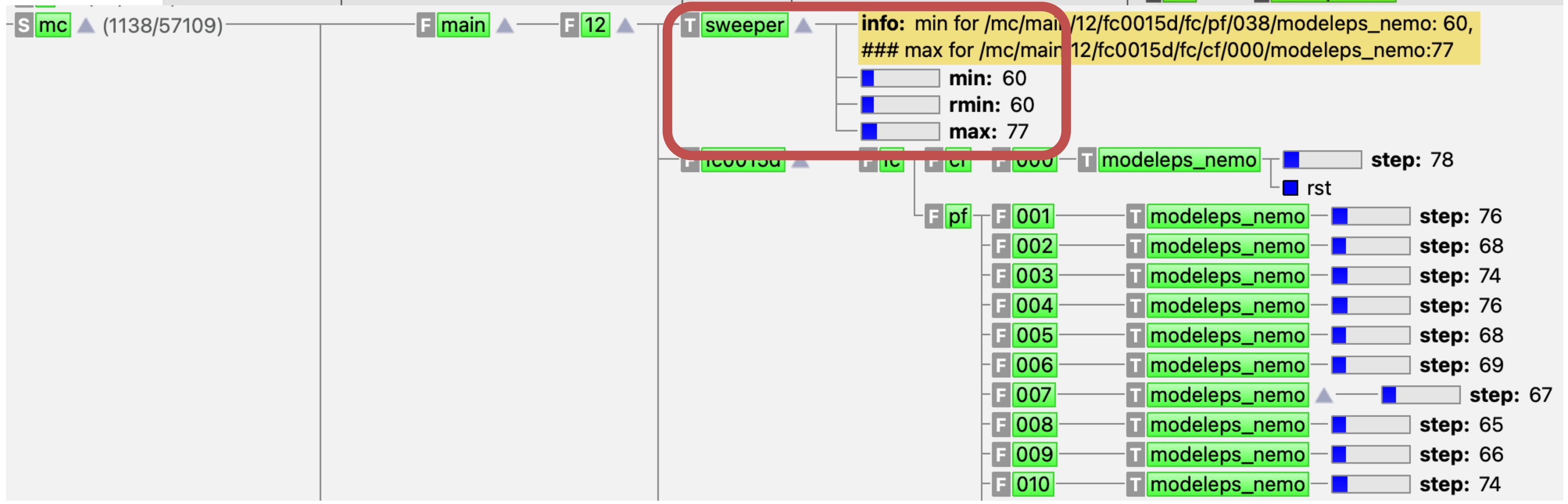




Best practices: Monitoring - Operators view

- Keep in mind **visibility** and **monitoring** purpose from the design phase: Operators' view
  - Operators display only submitted/active/aborted/suspended nodes and few attributes
  - Top is priority for ecflow and for operators
  - Locate up 'summary tasks' (synthetic task): ex. sweeper





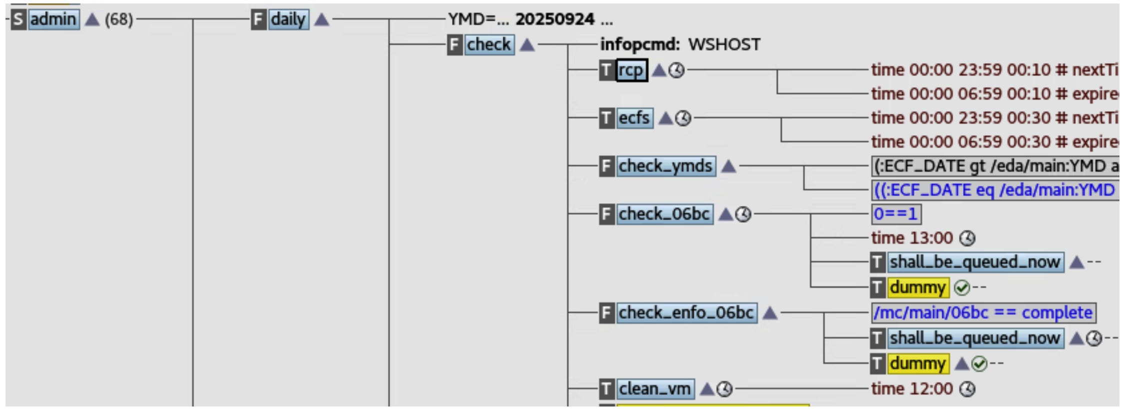
info: min for /mc/main/12/fc0015d/fc/pf/038/modeleps_nemo: 60, ### max for /mc/main/12/fc0015d/fc/cf/000/modeleps_nemo:77

min: 60
rmin: 60
max: 77

step: 78
rst
step: 76
step: 68
step: 74
step: 76
step: 68
step: 69
step: 67
step: 65
step: 66
step: 74

ecFlow: best practices – Monitoring - admin suite

- An admin suite per server
  - Snapshot checkpoint file 10min, 30min
  - Daily looping the server logs + archive
  - Warnings / cleaning for disk space
  - Host ‘outer watch dogs’ for suite



```
S admin (68) ── F daily ── YMD=... 20250924 ...
                         └ F check ── infopcmd: WSHOST
                                   ├ T rcp ──────────── time 00:00 23:59 00:10 # nextTi
                                   │                  └ time 00:00 06:59 00:10 # expire
                                   ├ T ecfs ─────────── time 00:00 23:59 00:30 # nextTi
                                   │                  └ time 00:00 06:59 00:30 # expire
                                   ├ F check_ymds ───── (:ECF_DATE gt /eda/main:YMD a
                                   │                  └ ((:ECF_DATE eq /eda/main:YMD
                                   ├ F check_06bc ───── 0==1
                                   │                  ├ time 13:00
                                   │                  ├ T shall_be_queued_now --
                                   │                  └ T dummy --
                                   ├ F check_enfo_06bc ─ /mc/main/06bc == complete
                                   │                  ├ T shall_be_queued_now --
                                   │                  └ T dummy --
                                   └ T clean_vm ─────── time 12:00
```

ecFlow: Best Practices – Errors handling

Detect

- Exit code (trap)
- Timeout: **late attribute**
- **Watch dogs**: inner, outer dedicated tasks to turn abort when a problem is detected
- Active watchdog (sweeper) vs passive watchdog (dummy task with trigger and complete attribute)
- Exception ?
- Monitoring the monitor

classify

- Light v critical:
  - **Transient:** managed setting ECF_TRIES > 1 to overcome glitches
  - **Resource:** adjust a limit, set limit 0 to 'suspend service'
  - **Critical**: disk switch, cluster switch, handover ecflow server
- Track error patterns

Take action

- Retry: **Automatic rerun** (ECF_TRIES)
- Automatic (task managed) **change of job behaviour** (according to its ECF_TRYNO)
- **Compensation**: family handover, server handover
- **Escalation**: Warn with mail, chat, Opsview
- Incident **response plan**: STHOST, SCHOST, FDB config, checkpoint recovery, server switch

ecFlow: Best Practices – Security & access

Authentication

- Certificates
- SSL/TLS communications ENABLE_SSL compilation option
- Password file

Authorisation

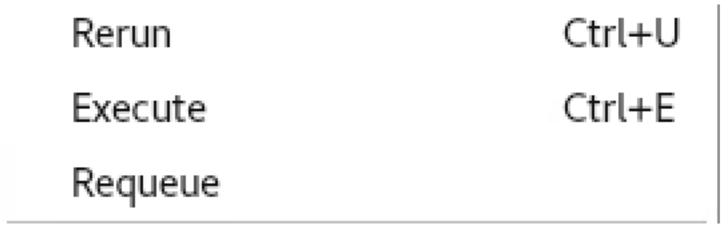
- RBAC: Whitelist for server read vs read-write access
- ACLs with ecFlow in preparation
- Hardening ecFlow server: VM with local logs, checkpoint, config scripts
- Isolation: docker container, network
- Server owner vs job owner
- Communication on a fixed port 3141

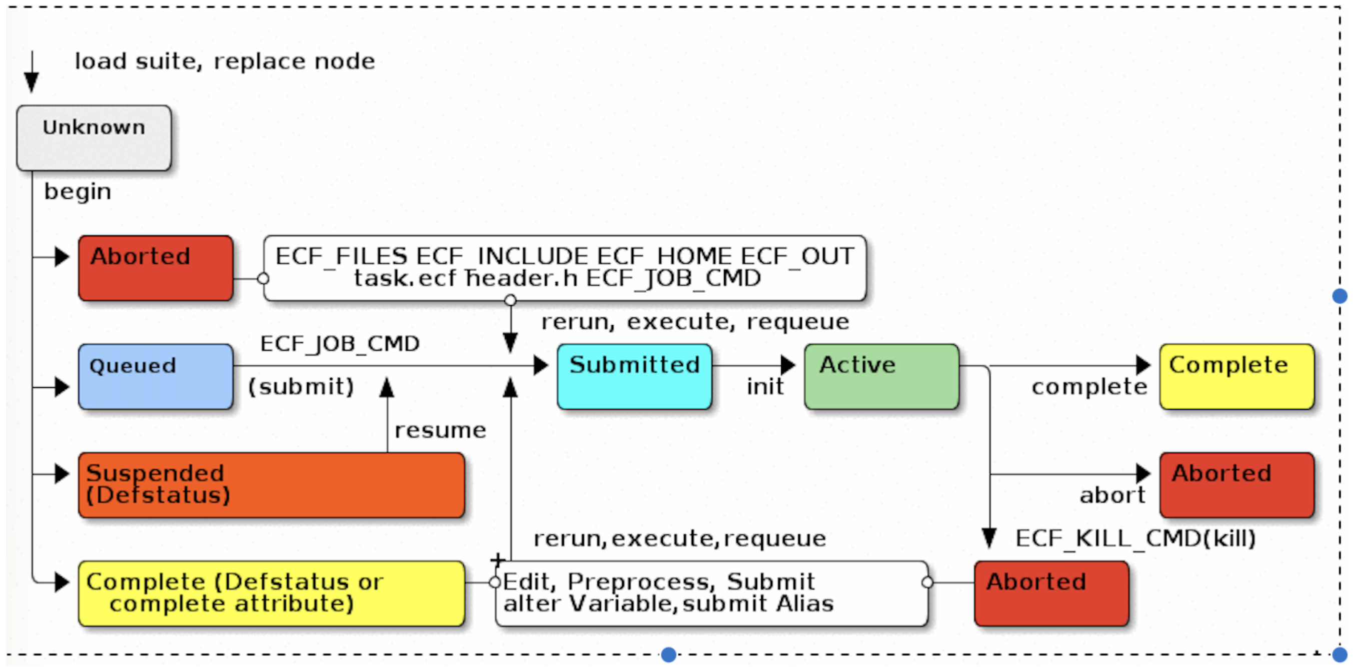
Audit Monitoring

- Logging: Server logs, jobs logs, tar logs into ecfs
- tracing
- Alerts
- Check compliance by peer-review and/or third party
- NEVER RUN ecFlow server AS ROOT

ecFlow: Best Practices – Operational tips – requeue?

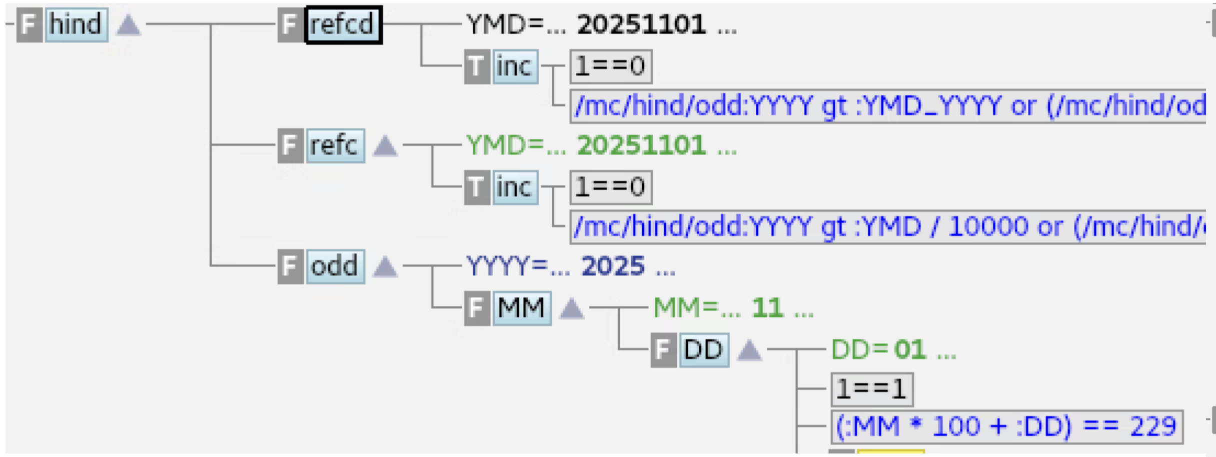
- We distinguish
  - Rerun: aka force queued (suite/family node),
    - ECF_TRYNO **increment** and honour limits, triggers, date/time dependencies
  - Execute: run jobs without condition
    - ECF_TRYNO **increment**
  - Requeue: resets
    - ECF_TRYNO (previous output **overwrite**),
    - repeat, date/time attributes
  - Requeue aborted: menu on Family/Suite node
    - Requeue only aborted tasks below



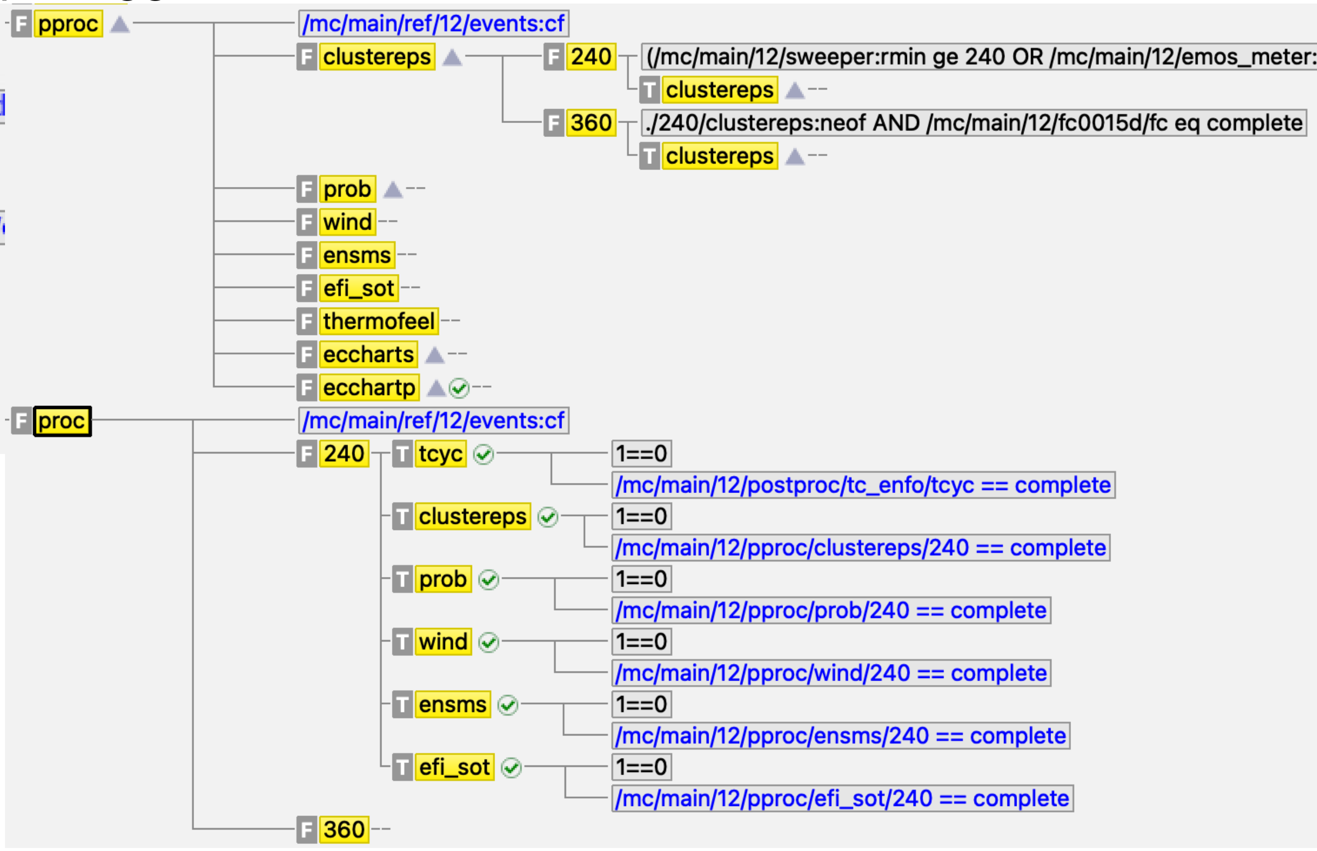


ecFlow: Best Practices – Operational tips

- Fake families/tasks to offer shuffled view for simple triggers



```
F hind
    F refcd   YMD=... 20251101 ...
        T inc   1==0
                /mc/hind/odd:YYYY gt :YMD_YYYY or (/mc/hind/od
    F refc    YMD=... 20251101 ...
        T inc   1==0
                /mc/hind/odd:YYYY gt :YMD / 10000 or (/mc/hind/
    F odd     YYYY=... 2025 ...
        F MM    MM=... 11 ...
            F DD    DD=01 ...
                    1==1
                    (:MM * 100 + :DD) == 229
```



```
F pproc   /mc/main/ref/12/events:cf
    F clustereps
        F 240   (/mc/main/12/sweeper:rmin ge 240 OR /mc/main/12/emos_meter:
            T clustereps
        F 360   ./240/clustereps:neof AND /mc/main/12/fc0015d/fc eq complete
            T clustereps
    F prob
    F wind
    F ensms
    F efi_sot
    F thermofeel
    F eccharts
    F ecchartp
F proc    /mc/main/ref/12/events:cf
    F 240
        T tcyc        1==0
                      /mc/main/12/postproc/tc_enfo/tcyc == complete
        T clustereps  1==0
                      /mc/main/12/pproc/clustereps/240 == complete
        T prob        1==0
                      /mc/main/12/pproc/prob/240 == complete
        T wind        1==0
                      /mc/main/12/pproc/wind/240 == complete
        T ensms       1==0
                      /mc/main/12/pproc/ensms/240 == complete
        T efi_sot     1==0
                      /mc/main/12/pproc/efi_sot/240 == complete
    F 360
```

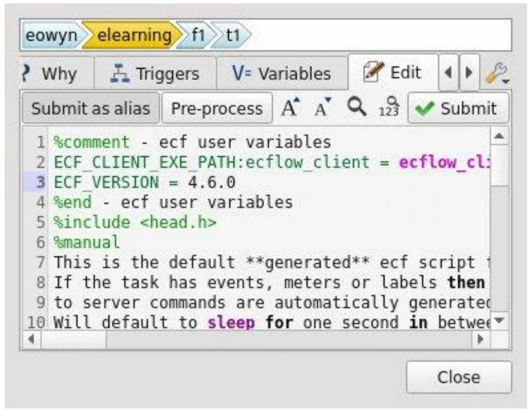
- Work with UTC
- We can change triggers from GUI/CLI:
  - beware drifting away from definition
  - better prefer node replace for valid, source-controlled definition
- Keep a maintenance time window: load balance suites over multiple servers

ecFlow: Best Practices – Operational tips

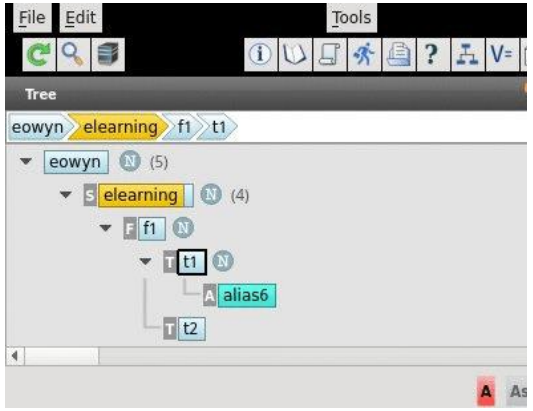
- Wrappers/headers can be updated with no need to replace the suite / node
  - Changing ECF_FILES, ECF_INCLUDE can point to another scripts’ location
  - Atomic wrapper/headers update may occur when the task is queued, away from peak time
  - We deploy the script update form the SCM into ECF_FILES or ECF_INCLUDE directory, logging the change
  - We can run the task as an alias with DELTA_DAY -1

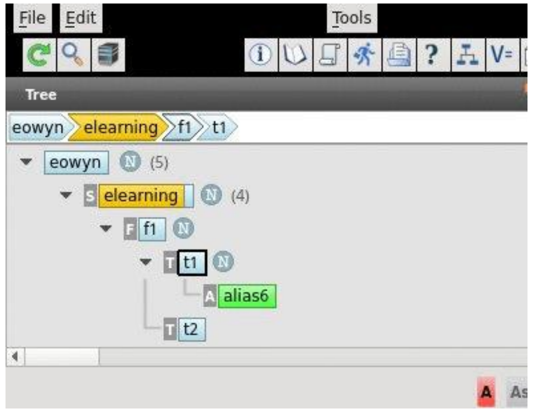
ecFlow: Operational tips - Alias

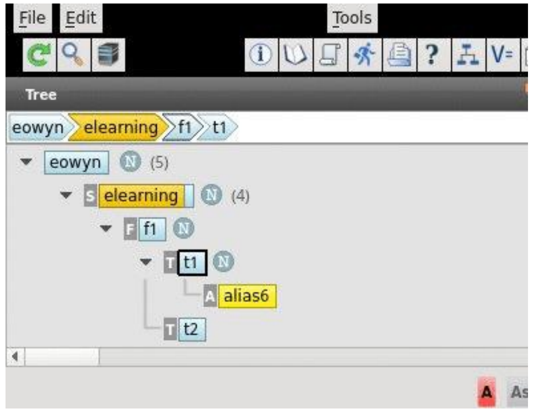
- An Alias can be created from GUI
  - Edit->Submit As Alias
  - Test a variation from the code without scripts alteration
  - Test a variation from variable(s)
  - A .usrN file is created and submitted
  - This file can be modified to submit the same alias again
  - Multiple aliases can be created for the same task



```
%comment - ecf user variables
ECF_CLIENT_EXE_PATH:ecflow_client = ecflow_cl
ECF_VERSION = 4.6.0
%end - ecf user variables
%include <head.h>
%manual
This is the default **generated** ecf script
If the task has events, meters or labels then
to server commands are automatically generate
Will default to sleep for one second in betwe
```

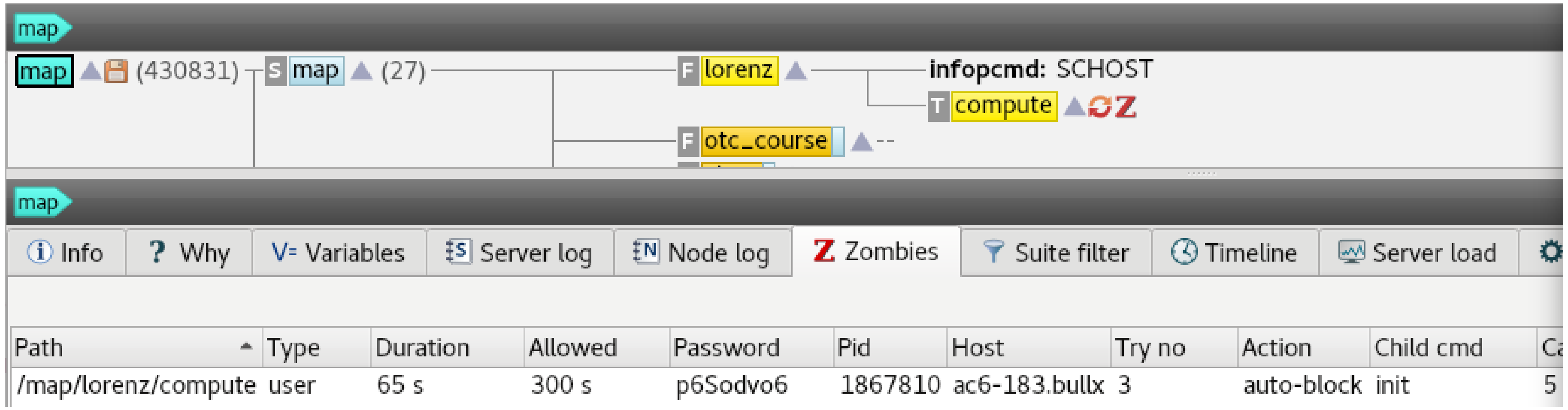






ecFlow: Operational Tips - Zombies

- Jobs are submitted with variable **ECF_PASS** set to pseudo-random value by ecflow server
- ⚠ **ATTENTION**
- Jobs are defined with unique identifiers **ECF_HOST-ECF_PORT-ECF_NAME-ECF_PASS**
  - ECF_PASS **mismatch** leads to a zombie
  - Simple (not best) action: set ECF_PASS FREE from ecflow_ui (Free password) to allow communication
  - Analyse logs, understand the cause (multiple init, child command after complete, system issue …)
  - Clear Flag
  - Terminate / Kill / Fob off / Delete / Rescue Adopt a child command on a backup server (child will look for a list of foster parent thanks to ECF_HOSTFILE.



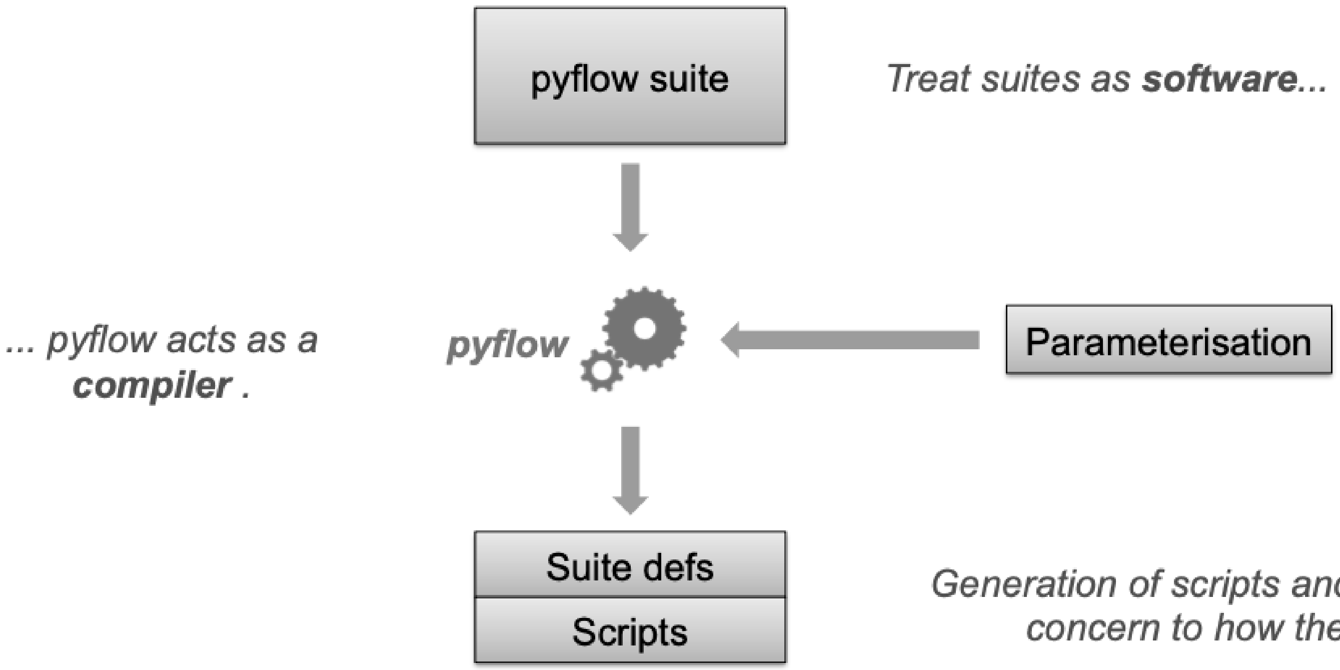
Path	Type	Duration	Allowed	Password	Pid	Host	Try no	Action	Child cmd	Ca
/map/lorenz/compute	user	65 s	300 s	p6Sodvo6	1867810	ac6-183.bullx	3	auto-block	init	5

ecFlow: Best Practices – Operational tips

- Multiple tabs with ecflow_ui vs multiple windows
  - Operators' view tab (only active/submitted tasks)
- Multiple nickname for same server:
  - Different settings: dev/test/prod

Best tools: Pyflow, a pythonic interface to ecFlow

The Tao of Pyflow


pyflow suite
Treat suites as **software**...
... pyflow acts as a **compiler** .
pyflow
Parameterisation
Suite defs
Scripts

Generation of scripts and suites is a separate concern to how they are deployed

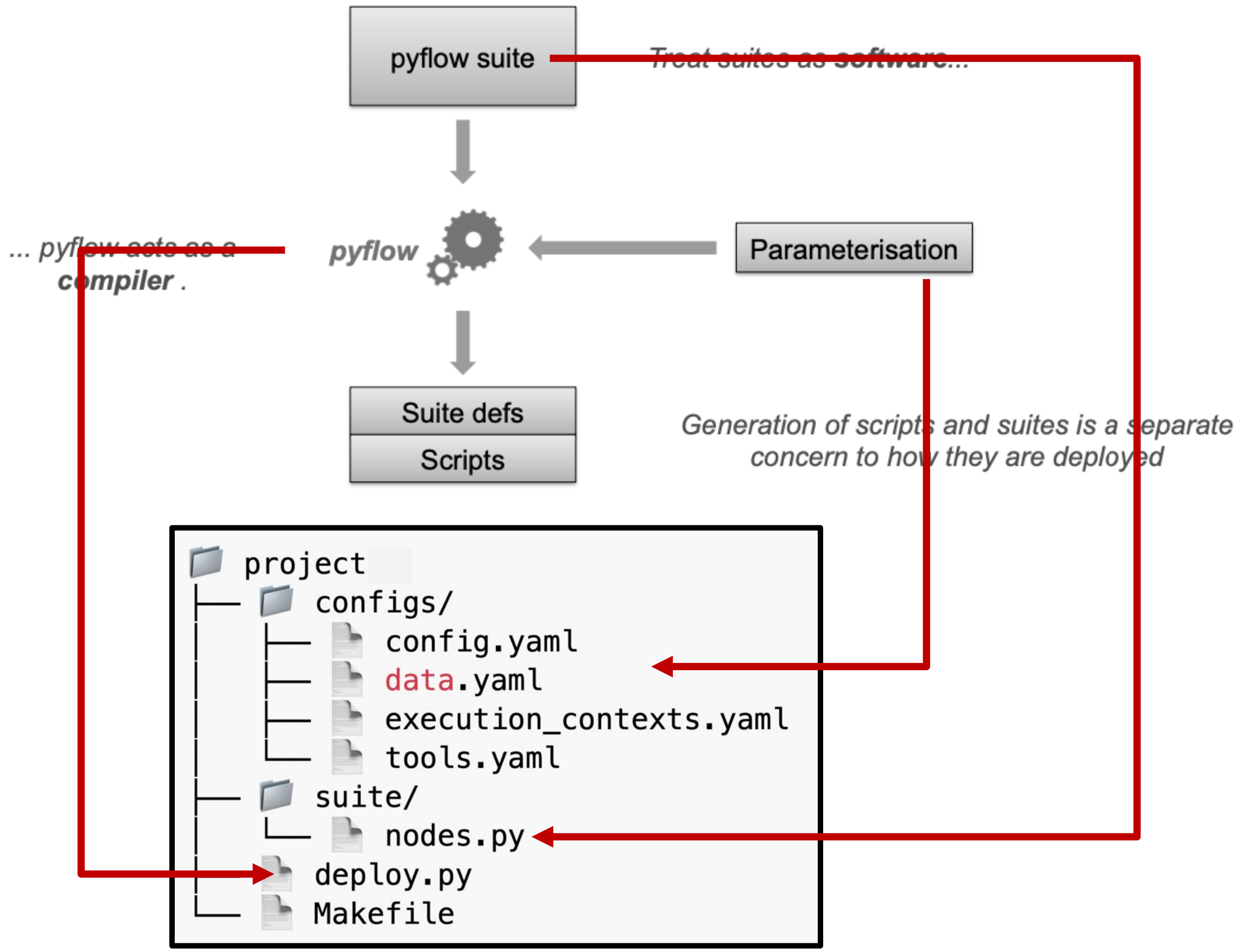
```
with pf.Suite('suite’):

   DeploymentFamily(config)

   with pf.Family(‘tests’, FLAG=123) as f:
     (
       RunUnitTests(config)
       >>
       IntegrationTest1(config, f.FLAG)
       >>
       IntegrationTest2(config, f.FLAG)
     )
```

https://github.com/ecmwf/pyflow

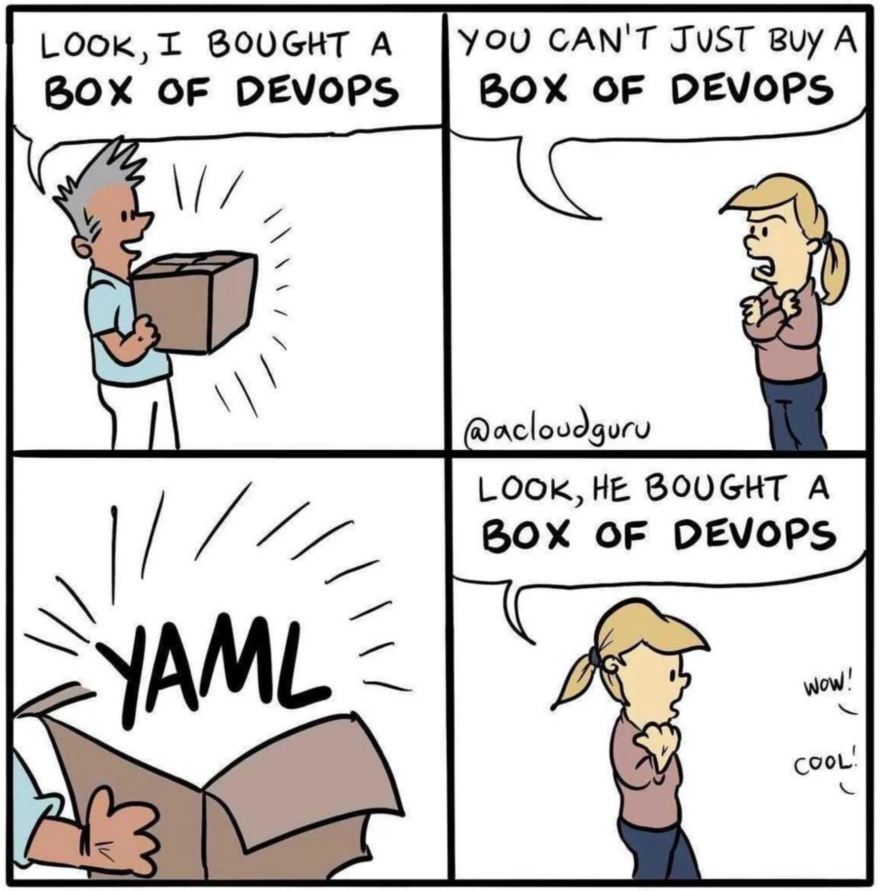
https://pyflow-workflow-generator.readthedocs.io

PYFLOW

ecFlow: best tools, Wellies: build consistent suites through YAML configuration


pyflow suite
Treat suites as software...
... pyflow acts as a compiler .
pyflow
Parameterisation
Suite defs
Scripts
Generation of scripts and suites is a separate concern to how they are deployed

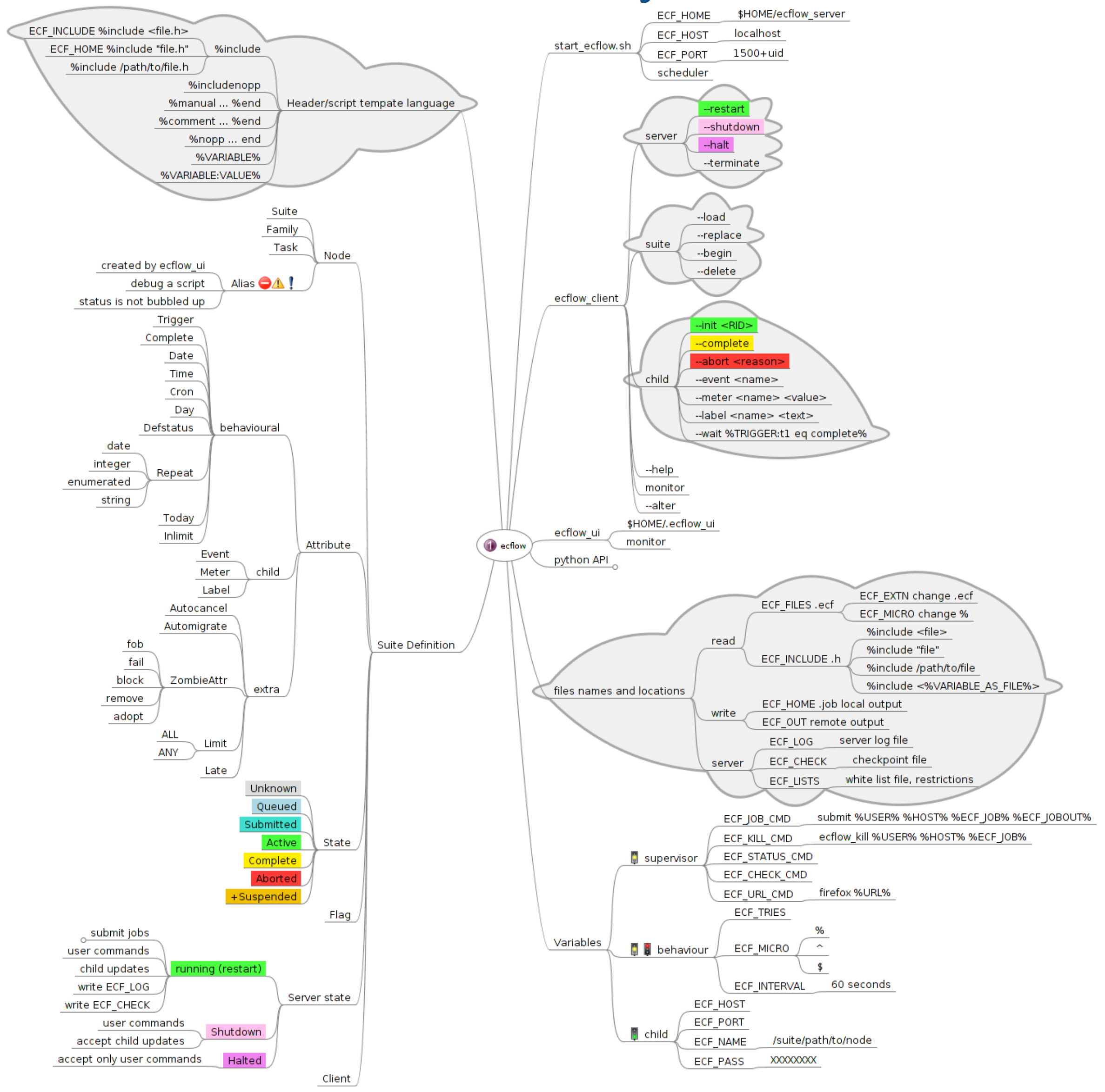
```
project
├── configs/
│   ├── config.yaml
│   ├── data.yaml
│   ├── execution_contexts.yaml
│   └── tools.yaml
├── suite/
│   └── nodes.py
├── deploy.py
└── Makefile
```


LOOK, I BOUGHT A BOX OF DEVOPS
YOU CAN'T JUST BUY A BOX OF DEVOPS
@acloudguru
YAML
LOOK, HE BOUGHT A BOX OF DEVOPS
WOW!
COOL!

ecFlow – best practices – check-list

- Re-runnability
- Look after critical data - HA systems, backups
- Limit number of languages used
- Be careful with error trapping
- All variables should to be set (use default values %VARIABLE:default_value%)
- Use a generic user - identify operations
- Works on multiple systems
  - ECF_JOB_CMD
- Design based on constraints
  - Staff availability
- Avoid accessing off-line data in critical path
- Avoid NFS mounted files or unsafe file-systems (SCRATCH)
- Tasks can be serial or parallel
  - don’t do serial things in parallel tasks
- Use generic directories to simplify cleaning and always clean up!
- Check task runtimes
- Keep output and job files
- Always use a SCM system and test
  - Test ecflow server/suites

Thanks for your attention!

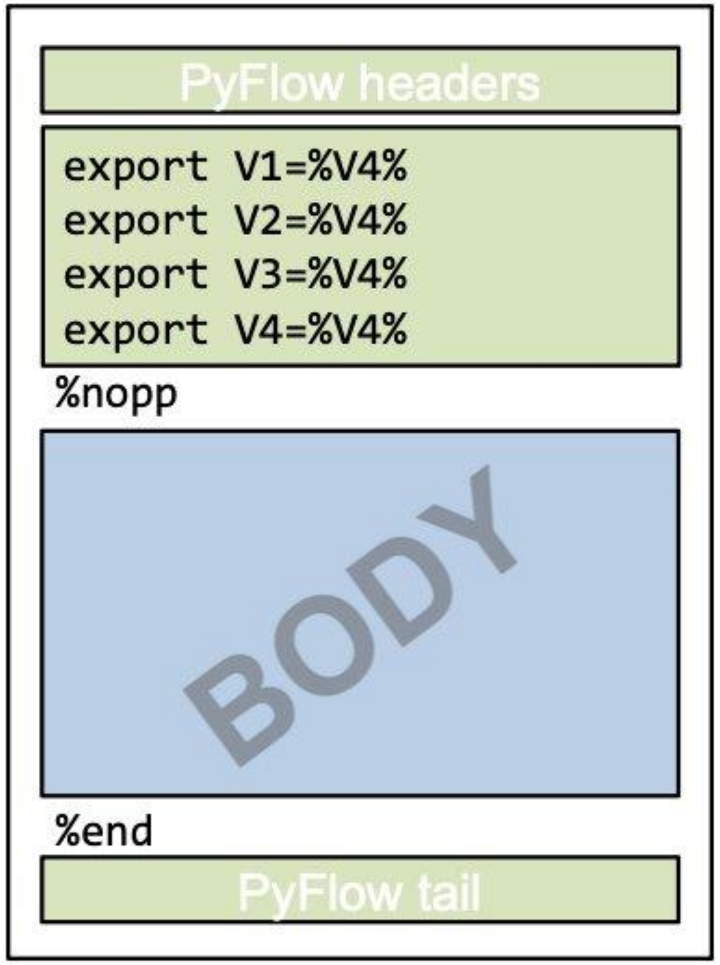


Practice with eLearning Jupiter lab notebook
ecFlow flashcard and Quizz?

Time for Questions?

Gratitude to Corentin Carton de Wiart for pyflow/Wellies aspects,
Gratitude to Christopher Barnard for the monitoring aspects
Image on slide 26 is thanks to Google Gemini

Pyflow task wrapper format



```
export V1=%V4%
export V2=%V4%
export V3=%V4%
export V4=%V4%
%nopp
%end
```

ecFlow: definition-file, checkpoint-file, nodes + attributes

Checkpoint-file written by ecflow_server

- a definition file
- defs, enddef, history additional keywords
- recent values for states and variables, next run time in comment

Nodes:

- suite, family, task
- (endsuite, endfamily, endtask)

Attributes can be classified in multiple ways:

- Active/passive (task requeued)
- Related with child command or not
- Behavioural: defstatus, complete

Looping

- repeat, Cron, time, today, date, day, defstatus, autocancel

Scheduling attributes

- trigger, complete, limit, inlimit

Informational attributes

- label, zombie

used in jobs

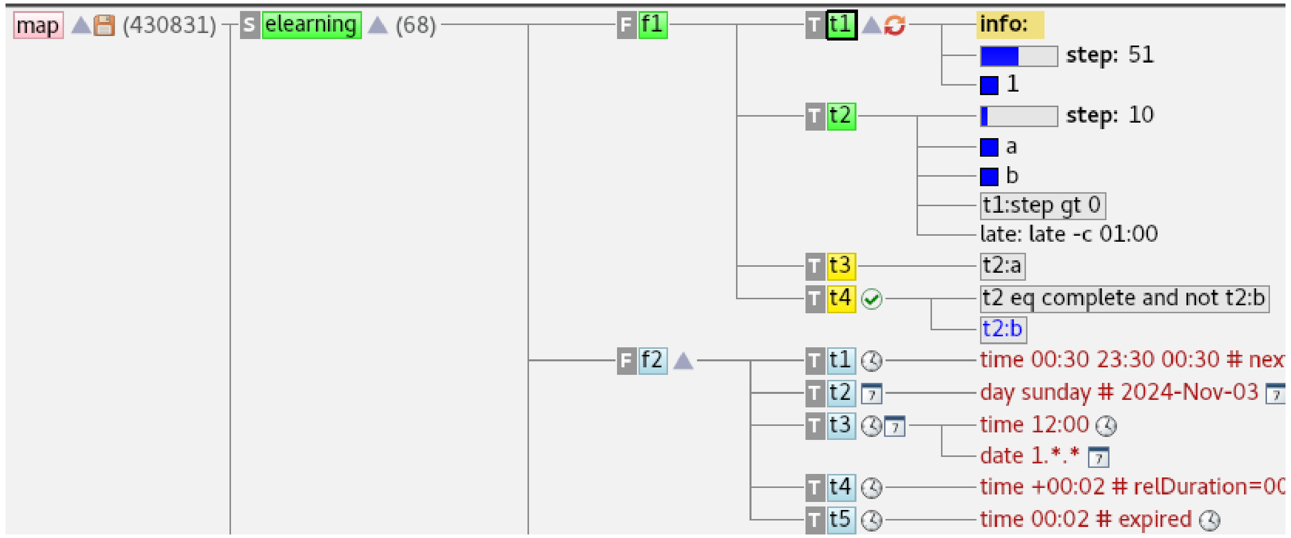
- edit (variable)

used in trigger

- Node status, variable, event, meter, limit, late

ecFlow: Inheritance status v variables v dependencies

- Status inheritance is bubbling up
- A suite or family node reflects most important status
- server node status can be
  - Halted: accept only user commands
  - Shutdown: accept user and child commands
  - Running: additionally, jobs can be submitted



```
map (430831) ─ S elearning (68) ─┬─ F f1 ─┬─ T t1 ─┬─ info:
                                 │        │        ├─ step: 51
                                 │        │        └─ 1
                                 │        ├─ T t2 ─┬─ step: 10
                                 │        │        ├─ a
                                 │        │        ├─ b
                                 │        │        ├─ t1:step gt 0
                                 │        │        └─ late: late -c 01:00
                                 │        ├─ T t3 ──── t2:a
                                 │        └─ T t4 ─┬─ t2 eq complete and not t2:b
                                 │                 └─ t2:b
                                 └─ F f2 ─┬─ T t1 ──── time 00:30 23:30 00:30 # nex
                                          ├─ T t2 ──── day sunday # 2024-Nov-03
                                          ├─ T t3 ─┬─ time 12:00
                                          │        └─ date 1.*.*
                                          ├─ T t4 ──── time +00:02 # relDuration=00
                                          └─ T t5 ──── time 00:02 # expired
```

- variables inheritance is top down
  - A Variable can be redefined lower in the tree
  - Lowest value prevail for jobs creation
- dependencies can be defined on any node
  - Trigger, complete, time, date, Cron attribute
  - All conditions must be true to create a job
  - High dependency will hide the lower
  - Trigger, complete attribute are instantaneous
  - Date, time, Cron attribute have memory

ecFlow: Tasks wrappers / Tasks headers

key variables

- ECF_EXTN: wrapper extension .ecf .sh .py
- ECF_FILES: wrappers location (r)
- ECF_INCLUDE : headers location (r)
- ECF_HOME: where .job are created (w)

Tasks wrappers

- a template script
- describe generic or specific work to do

Tasks headers

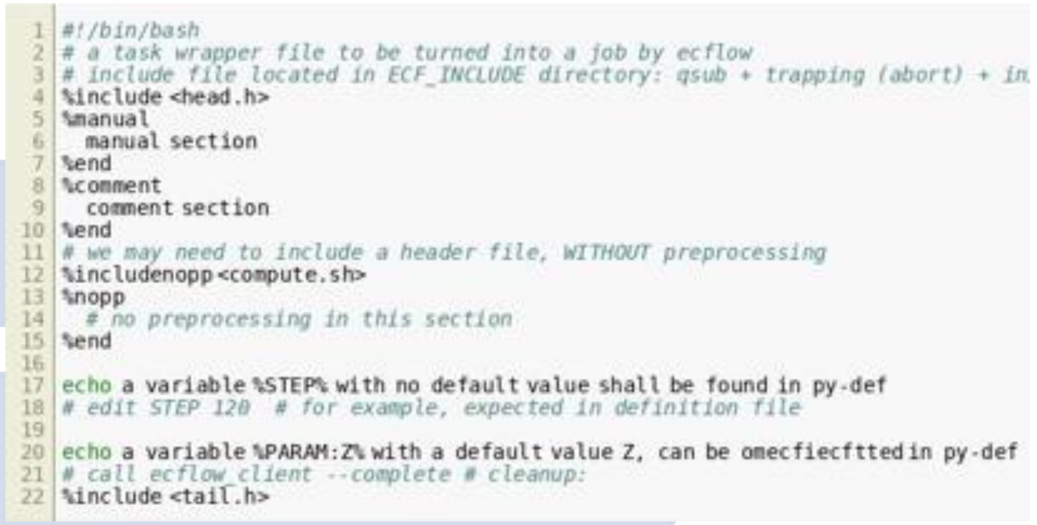
- head.h / qsub.h / tail.h
- %include <%QSUB_H:qsub.h%>

ECF_MICRO % character: variable/block/keyword

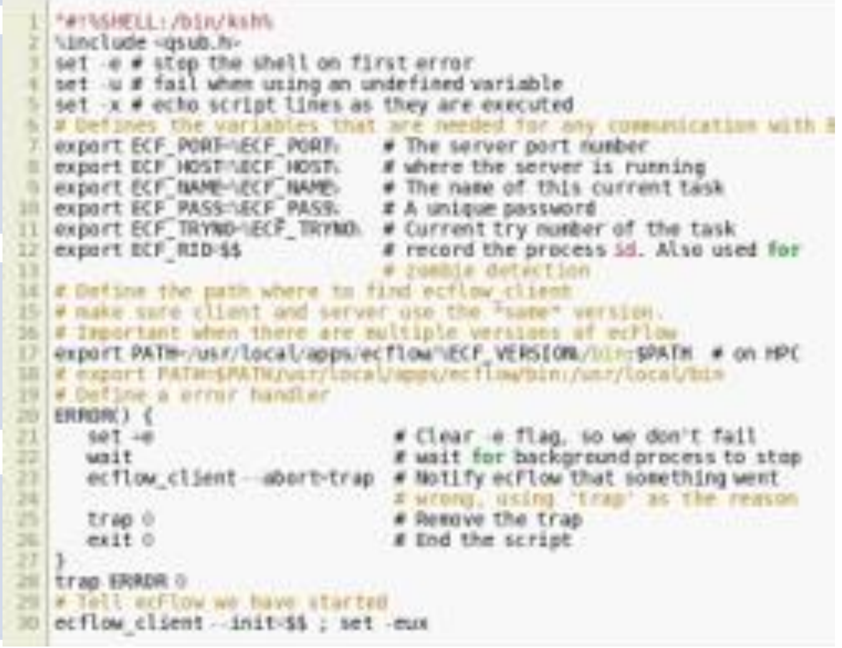
- %VARIABLE:default_value%
- manual, nopp, comment,
- include, includenopp
- global scale or locally in the template script: %ecf_micro $

Tolerance for failures (hardware and software):

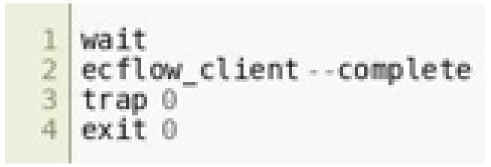
- ECF_TRIES: number of automatic rerun
- ECF_TRYNO: job instance number
- Watchdog task to handle known issues



```
#!/bin/bash
# a task wrapper file to be turned into a job by ecflow
# include file located in ECF_INCLUDE directory: qsub + trapping (abort) + in
%include <head.h>
%manual
  manual section
%end
%comment
  comment section
%end
# we may need to include a header file, WITHOUT preprocessing
%includenopp <compute.sh>
%nopp
  # no preprocessing in this section
%end

echo a variable %STEP% with no default value shall be found in py-def
# edit STEP 120  # for example, expected in definition file

echo a variable %PARAM:Z% with a default value Z, can be omecfiecftted in py-def
# call ecflow_client --complete # cleanup:
%include <tail.h>
```



```
#!%SHELL:/bin/ksh%
%include <qsub.h>
set -e # stop the shell on first error
set -u # fail when using an undefined variable
set -x # echo script lines as they are executed
# Defines the variables that are needed for any communication with E
export ECF_PORT=%ECF_PORT%      # The server port number
export ECF_HOST=%ECF_HOST%      # where the server is running
export ECF_NAME=%ECF_NAME%      # The name of this current task
export ECF_PASS=%ECF_PASS%      # A unique password
export ECF_TRYNO=%ECF_TRYNO%    # Current try number of the task
export ECF_RID=$$               # record the process id. Also used for
                                # zombie detection
# Define the path where to find ecflow_client
# make sure client and server use the *same* version.
# Important when there are multiple versions of ecFlow
export PATH=/usr/local/apps/ecflow/%ECF_VERSION%/bin:$PATH  # on HPC
# export PATH=$PATH/usr/local/apps/ecflow/bin:/usr/local/bin
# Define a error handler
ERROR() {
   set +e                        # Clear -e flag, so we don't fail
   wait                          # wait for background process to stop
   ecflow_client --abort=trap    # Notify ecFlow that something went
                                 # wrong, using 'trap' as the reason
   trap 0                        # Remove the trap
   exit 0                        # End the script
}
trap ERROR 0
# Tell ecFlow we have started
ecflow_client --init=$$ ; set -eux
```



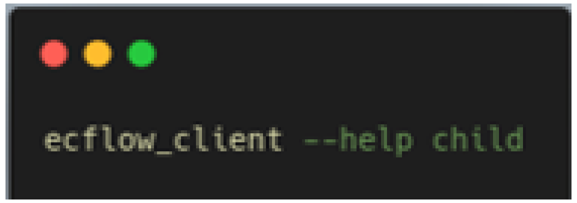
```
wait
ecflow_client --complete
trap 0
exit 0
```

EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

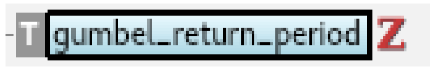
ecFlow: child commands

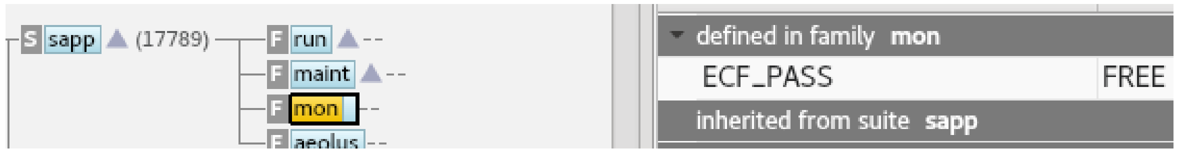
- ecflow_client called from a job

- 4 variables:
  - ECF_NAME: path for the node in the definition tree
  - ECF_HOST,
  - ECF_PORT,
  - ECF_PASS:
    - unique pseudorandom key for current job.
    - Zombie flag is raised when incorrect.
    - set to FREE to rescue a child, or in monitoring mode



```
ecflow_client --help child
```





ECF_PASS	FREE

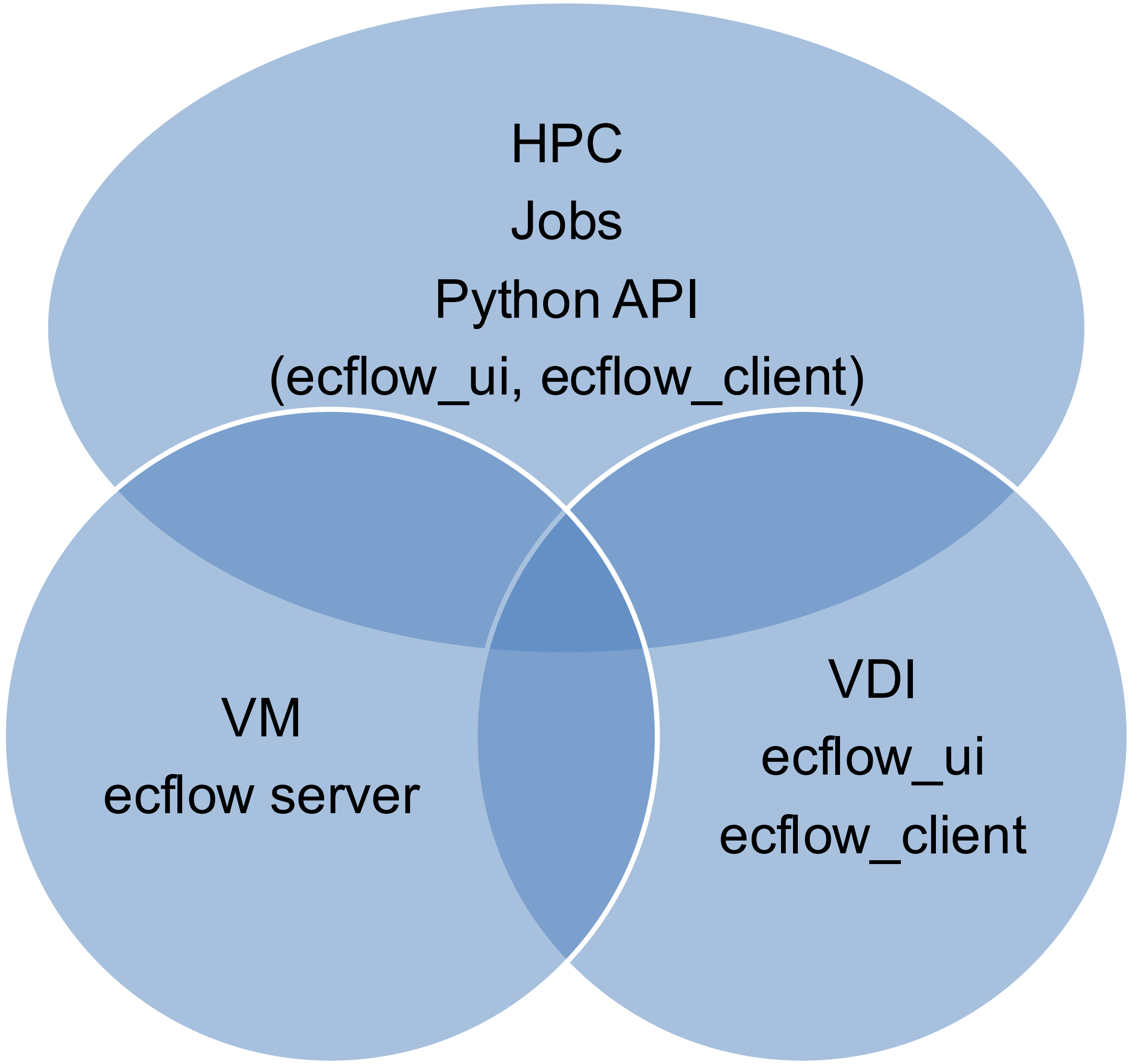
Update status:	Update attribute:	Embedded trigger:	Write into server log:	Get an item from a list: queue
• init <jid> • complete • abort <reason>	• meter <name> <value> • event <name> • label <name> <msg>	• wait <expression>	• msg <text>	• queue <name> <list> # def-file

ecflow_client: child

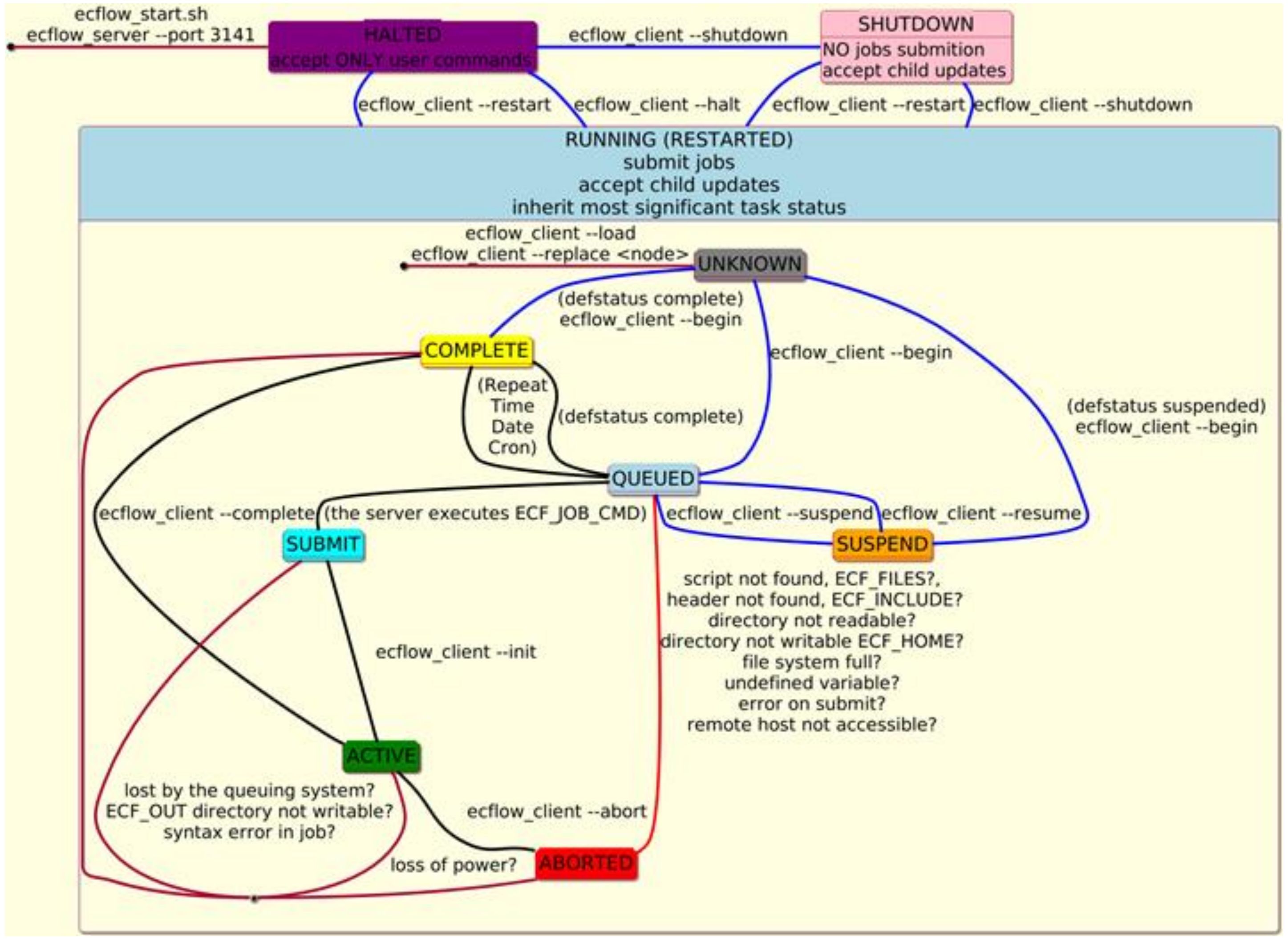
- Update status: --**init** $JOBID –**abort** “$REASON” –**complete**
- Attribute update: --**event** <name> --**meter** <name> <value> --**label** <name> ”$message”
- Child commands are ‘privileged’ i.e. blocking
  - Some may prefer UDP (for labels, meter, event when not critical)
  - --**wait** $expression
    - Is blocking embedded trigger, that may cause a live lock
    - Some will prefer the non blocking –query $node to check status or a variable
  - token=$(ecflow_client --**queue** <name>)
    - get a token, from a queue attribute (definition)
- When parent server is not responding:
  - **ECF_HOSTFILE**=/path/to/file
  - The child will look if this variable is present, if the file is accessible, and will contact the listed alternative servers for adoption. Backup server shall already have a (suspended) suite matching for this job, zombie **icon** shall be displayed, and the task shall be listed in the **zombies**’ **panel**

ecFlow: users use case

- ecFlow server is hosted in a dedicated VM
  - ping ecflow-gen-${USER}-001
- ecflow_ui is run on VDI (or laptop, or HPC)
- Jobs are submitted on HPC
- $HOME is common between VM, HPC, VDI
  - .check, .log under $HOME/ecflow_servers
  - File ecf.lists to grant or refrain access (rw/r/none)

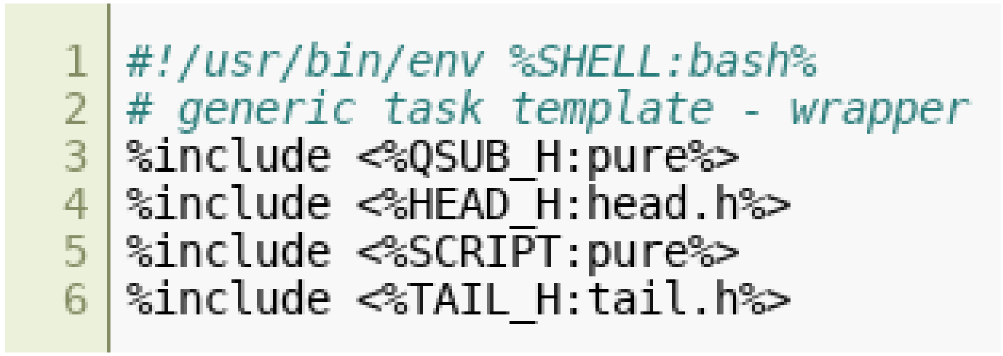


ecFlow: statuses diagram



Ecflow: python – Perl –ruby – bash – na(t)ive example

- A generic wrapper: body.ecf
- **Headers** included are defined **as variables** in the definition
- **ECF_MICRO** may change (% default, used for ksh/bash)
  - ^ for Perl
  - $ for python
- Script extension can change using **ECF_EXTN**:
  - .ecf (default), .ecg (generated template), .ech (preprocess to shell), .erb (to ruby), epy, epl
  - .rb (pure ruby), .pl, .py, .sh
- **Payload** defined as SCRIPT variable
  - To be pre-processed with %include
  - Or not, using %includenopp
- An empty file (touch): **pure**
- QSUB_H empty for local submit, or can be defined as slurm.h, pbs.h, qsub.h provided as needed



```
#!/usr/bin/env %SHELL:bash%
# generic task template - wrapper
%include <%QSUB_H:pure%>
%include <%HEAD_H:head.h%>
%include <%SCRIPT:pure%>
%include <%TAIL_H:tail.h%>
```

Ecflow: python – na(t)ive example

- Python def / head / body / tail

```
#5.14.1
suite starter
  defstatus suspended
  family example
    edit ECF_MICRO '%'
    family language
      edit ECF_FILES '/path/to/example/lang'
      edit ECF_INCLUDE '/path/to/example/test/lang'
      edit BBIN '/home/linuxbrew/.linuxbrew/bin/'
      family python
        edit SHELL 'python3'
        edit ECF_MICRO '$'
        edit ECF_EXTN '.epy'
        edit HEAD_H 'head.epy'
        edit TAIL_H 'tail.py'
        edit ECF_JOB_CMD '$ECF_JOB$ 1> $ECF_JOBOUT$ 2>&1'
        task doit
          edit SCRIPT 'hello_world.py'
      endfamily

```

```
#!/usr/bin/env $SHELL:python3$
$include <$QSUB_H:pure$>
$include <$HEAD_H:head.epy$>
$includenopp <$SCRIPT:pure$>
$include <$TAIL_H:tail.py$>
```

```
#!/usr/bin/env python3
print("Hello world");
```

```
# managed by atexit
```

```
child.report("complete")
```

```
class Child(object):
    # ...
    def signal_handler(self, signum, frame):
      """ catch signal """
      print("Aborting: Signal handler called with signal ", signum)
      self.report("abort", "Signal handler called with signal " + str(signum))

    def __exit__(self, exc_type, exc_value, traceback):
      self.report("abort", "__exit__")

    def report(self, msg, meter=None):
        """ communicate with ecFlow server """
        if msg in ("stop", "complete"):
            self.client.child_complete()
            self.client = None
            sys.stdout.flush()
            sys.stderr.flush()
            print("#MSG: stop")
            sys.exit(0)
        elif msg in ("abort",):
            self.client.child_abort()
            self.client = None
            raise Exception(msg)
        elif meter:
            self.client.child_meter(msg, meter)
        else:
            self.client.child_label("info", msg)

```

```
#!/usr/bin/env $SHELL:python3$
def excepthook(exctype, value, traceback):
    if exctype == KeyboardInterrupt:
        if child:
            child.report("abort", "keyb")
    else:
        sys.__excepthook__(exctype, value, traceback)
        if child:
            child.report("abort", "gen")

class Child(object):
    def __init__(self):
        import signal
        print("#MSG: kill: ssh %s kill -15 %d" % (os.uname()[1], os.getpid()))
        for sig in (
            signal.SIGINT,
            signal.SIGHUP,
            signal.SIGQUIT,
            signal.SIGILL,
            signal.SIGTRAP,
            signal.SIGIOT,
            signal.SIGBUS,
            signal.SIGFPE,
            signal.SIGUSR1,
            signal.SIGUSR2,
            signal.SIGPIPE,
            signal.SIGTERM,
            signal.SIGXCPU,
            signal.SIGPWR,
        ):
            signal.signal(sig, self.signal_handler)
        self.set_client()

    def set_client(self):
        self.client = ecflow.Client()
        host = "$ECF_HOST:$"
        self.client.set_host_port(host, int("$ECF_PORT:0$"))
        self.client.set_child_pid(os.getpid())
        self.client.set_child_path("$ECF_NAME$")
        self.client.set_child_password("$ECF_PASS$")
        self.client.set_child_try_no(int("$ECF_TRYNO$"))
        self.client.child_init()
        self.client.set_child_timeout(20)

```

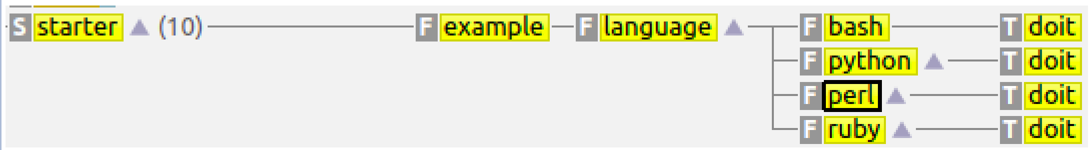
```
child = Child()
$includenopp <$SCRIPT:pure$>
child.report("complete")
```

Ecflow: perl - native example

- Perl

```
#5.14.1
suite starter
  defstatus suspended
  family example
    edit ECF_MICRO '%'
    family language
      edit ECF_FILES '/path/to/example/lang'
      edit ECF_INCLUDE '/path/to/example/test/lang'
      edit BBIN '/home/linuxbrew/.linuxbrew/bin/'
      family perl
        edit SHELL 'perl'
        edit ECF_MICRO '^'
        edit ECF_EXTN '.epl'
        edit HEAD_H 'head.epl'
        edit TAIL_H 'tail.pl'
        edit ECF_JOB_CMD '^ECF_JOB^ 1> ^ECF_JOBOUT^ 2>&1'
        task doit
          edit SCRIPT 'hello_world.pl'
      endfamily
```

```
#!/usr/bin/env ^SHELL:perl^
^include <^QSUB_H:pure^>
^include <^HEAD_H:head.h^>
^includenopp <^SCRIPT:pure^>
^include <^TAIL_H:tail.h^>
```



```
#!/usr/bin/env perl
# head.pl
use strict;

# my $ECF_PORT=^ECF_PORT:0^;
$ENV{'ECF_PORT'} = "^ECF_PORT:0^"; # port
$ENV{'ECF_HOST'} = "^ECF_HOST:0^"; # host
$ENV{'ECF_NAME'} = "^ECF_NAME:0^"; # task path
$ENV{'ECF_PASS'} = "^ECF_PASS:0^"; # password
$ENV{'ECF_TRYNO'} = "^ECF_TRYNO:0^"; # job number
sub xinit() { system("^BBIN:^ecflow_client --init $$"); }
sub xabort() { system("^BBIN:^ecflow_client --abort $$"); }
sub xcomplete() { system("^BBIN:^ecflow_client --complete $$"); }
sub xmeter($$) { my $name=shift; my $value=shift;
      system("^BBIN:^ecflow_client --meter $name $value"); }
sub xevent($)  { my $n=shift;
      system("^BBIN:^ecflow_client --event $n"); }
sub xlabel($$) { my $name=shift; my $value=shift;
      system("^BBIN:^ecflow_client --label $name $value"); }
xinit();
eval '
```

```
#!/usr/bin/env perl
print("Hello World\n");
```

```
# tail.pl
';
if ($@){
    print "# caught signal: $@\n";
    xabort();
    exit;
  }
print "# the job is now complete\n";
xcomplete();
exit;
```

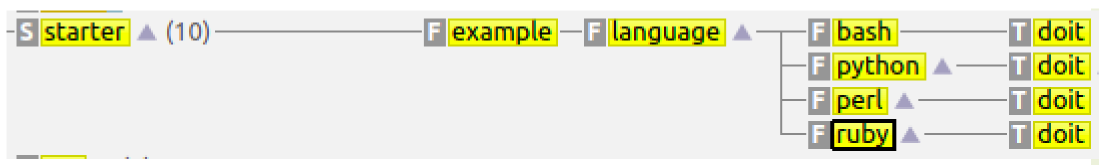
ecFlow: ruby – native example

- Ruby head / body / tail

```
#5.14.1
suite starter
  defstatus suspended
  family example
    edit ECF_MICRO '%'
    family language
      edit ECF_FILES '/path/to/example/lang'
      edit ECF_INCLUDE '/path/to/example/test/lang'
      edit BBIN '/home/linuxbrew/.linuxbrew/bin/'
      family ruby
        edit SHELL 'ruby'
        edit ECF_MICRO '^'
        edit ECF_EXTN '.erb'
        edit HEAD_H 'head.erb'
        edit TAIL_H 'tail.rb'
        edit ECF_JOB_CMD '^ECF_JOB^ 1> ^ECF_JOBOUT^ 2>&1'
        task doit
          edit SCRIPT 'hello_world.rb'
      endfamily
```

```
#!/usr/bin/env ^SHELL:ruby^
^include <^QSUB_H:pure^>
^include <^HEAD_H:head.h^>
^includenopp <^SCRIPT:pure^>
^include <^TAIL_H:tail.h^>
```

```
#!/usr/bin/env ruby
print("Hello World\n");
```



```
# tail.rb
# begin
rescue => e
  puts "caught signal: #{e}"
  xabort
  exit
end

puts "the job is now complete"
xcomplete
exit
```

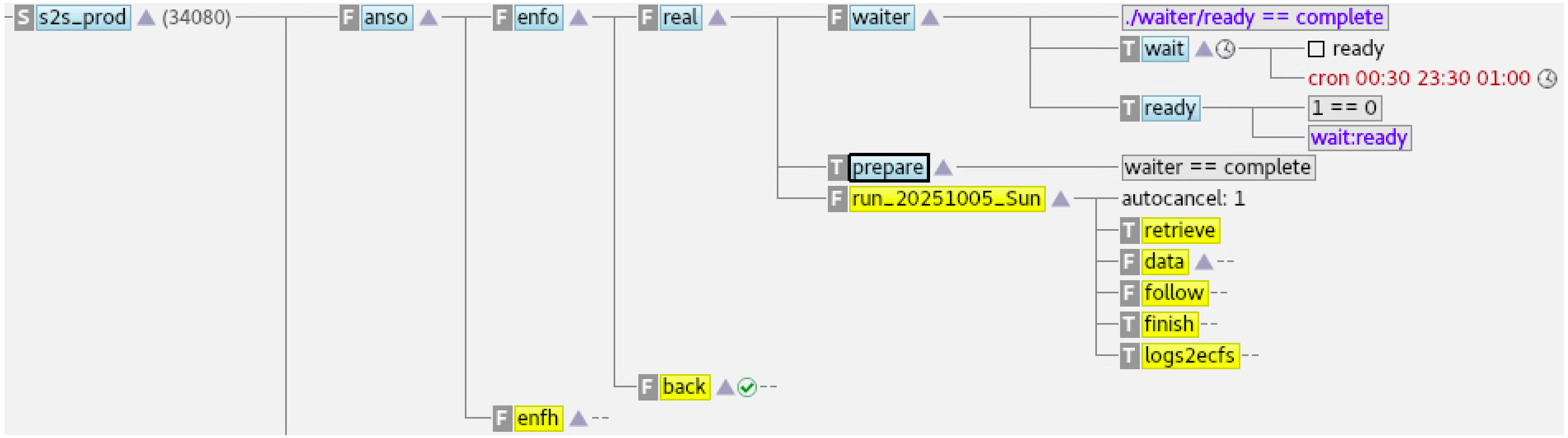
```
#!/usr/bin/env ruby
# coding: utf-8
# head.rb
# env variable for child commands
ENV['ECF_PORT']  = "^ECF_PORT:0^"   # port
ENV['ECF_HOST']  = "^ECF_HOST:0^"   # host
ENV['ECF_NAME']  = "^ECF_NAME:0^"   # task path
ENV['ECF_PASS']  = "^ECF_PASS:0^"   # password
ENV['ECF_TRYNO'] = "^ECF_TRYNO:0^"  # job number

def xinit
  system("^BBIN:^ecflow_client --init #{Process.pid}")
end

def xabort
  system("^BBIN:^ecflow_client --abort #{Process.pid}")
end

def xcomplete
  system("^BBIN:^ecflow_client --complete #{Process.pid}")
end

def xmeter(name, value)
  system("^BBIN:^ecflow_client --meter #{name} #{value}")
end

def xevent(n)
  system("^BBIN:^ecflow_client --event #{n}")
end

def xlabel(name, value)
  system("^BBIN:^ecflow_client --label #{name} #{value}")
end

# init
xinit

begin

```

%include preprocessing directive

- %include <file.h>
  - include a file under ECF_INCLUDE directory
- %include “file.h”
  - Include a file below ECF_HOME directory
- %include /path/to/file
  - a hardcoded location
- %include: NOTE % MUST be first character of the line
  - Avoid complexity, it prevents echo “%include <file>”
  - Avoid ambiguity: # %include <file>
- %include <%FILE_H:pure%>
  - Filename can be provided by a suite variable, here FILE_H
  - Edit({“FILE_H”: “config.oper.h”, })
  - Edit({“FILE_H”: “config.test.h”, })
- %includeonce <%FILE_H:pure%>

ecFlow: Best Practices – Suites' design – dynamic families

- A task to decorate a suite with new families
  - Barber shop example
  - S2S: waiter/prepare + autocancel



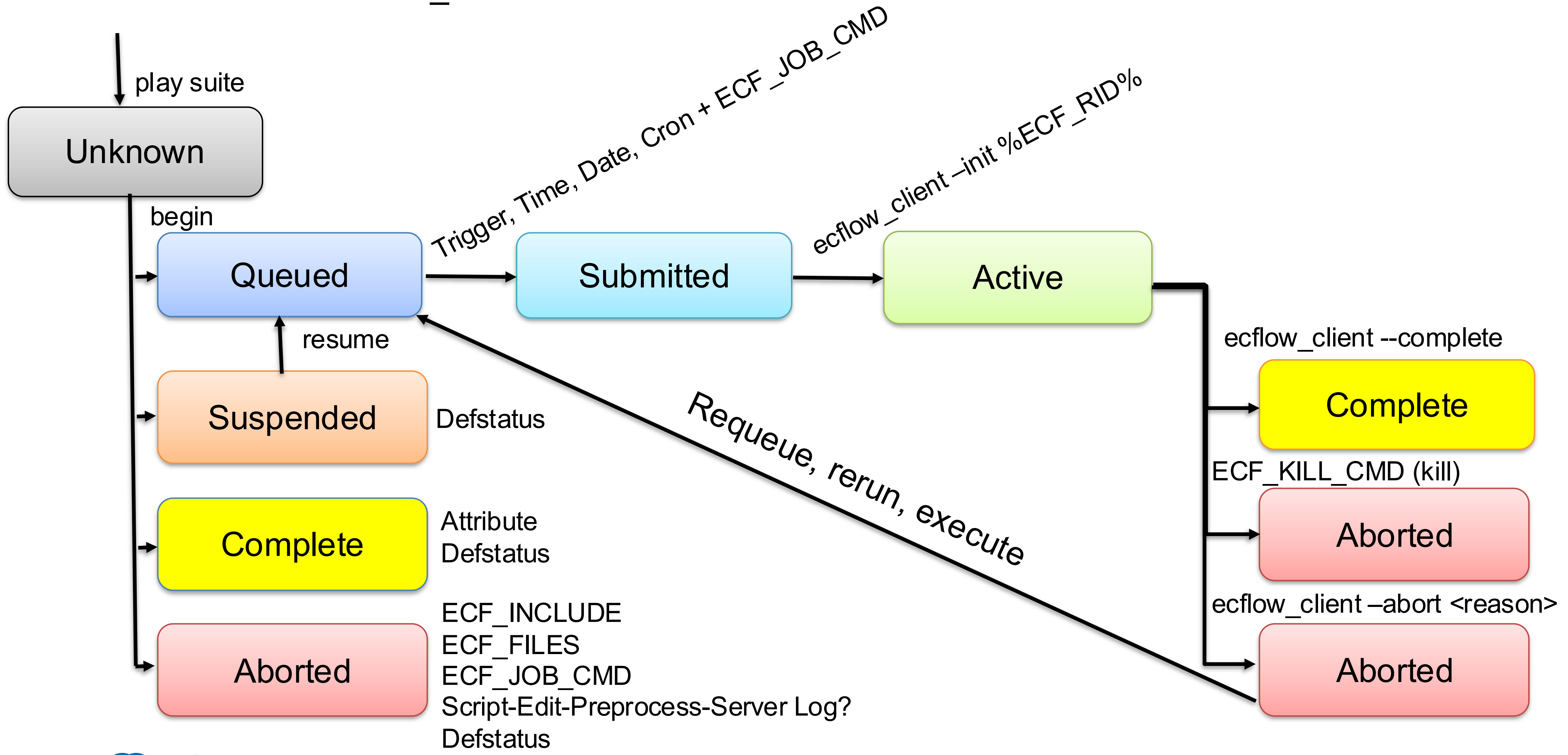
```
S s2s_prod (34080) ── F anso ── F enfo ── F real ── F waiter ── ./waiter/ready == complete
                                                           ├─ T wait ── ☐ ready
                                                           │           └ cron 00:30 23:30 01:00
                                                           └─ T ready ── 1 == 0
                                                                      └─ wait:ready
                                                  ├─ T prepare ── waiter == complete
                                                  └─ F run_20251005_Sun ── autocancel: 1
                                                                        ├─ T retrieve
                                                                        ├─ F data --
                                                                        ├─ F follow --
                                                                        ├─ T finish --
                                                                        └─ T logs2ecfs --
                                        └─ F back --
                              └─ F enfh --
```

ecFlow: Hands-in

- Definition + scripts under SCM (git)
- ECF_FILES/ECF_INCLUDE may refer to the same directory
- One time vs cyclic script
  - Choose ignition strategy: time condition, event set by third party, a wait barrier
  - Choose looping strategy: asap, or delay looping (keep output visible + rerun capable)
- A simple wrapper: lorenz.ecf

ecFlow: Status

- Task v Node: ecflow_client v inheritance



Pyflow ecosystem links

https://github.com/ecmwf/pyflow

https://github.com/ecmwf/pyflow-wellies

https://github.com/ecmwf/tracksuite

https://github.com/ecmwf/troika

https://pyflow-workflow-generator.readthedocs.io

https://pyflow-wellies.readthedocs.io

WELLIES

PYFLOW

module load wellies/new