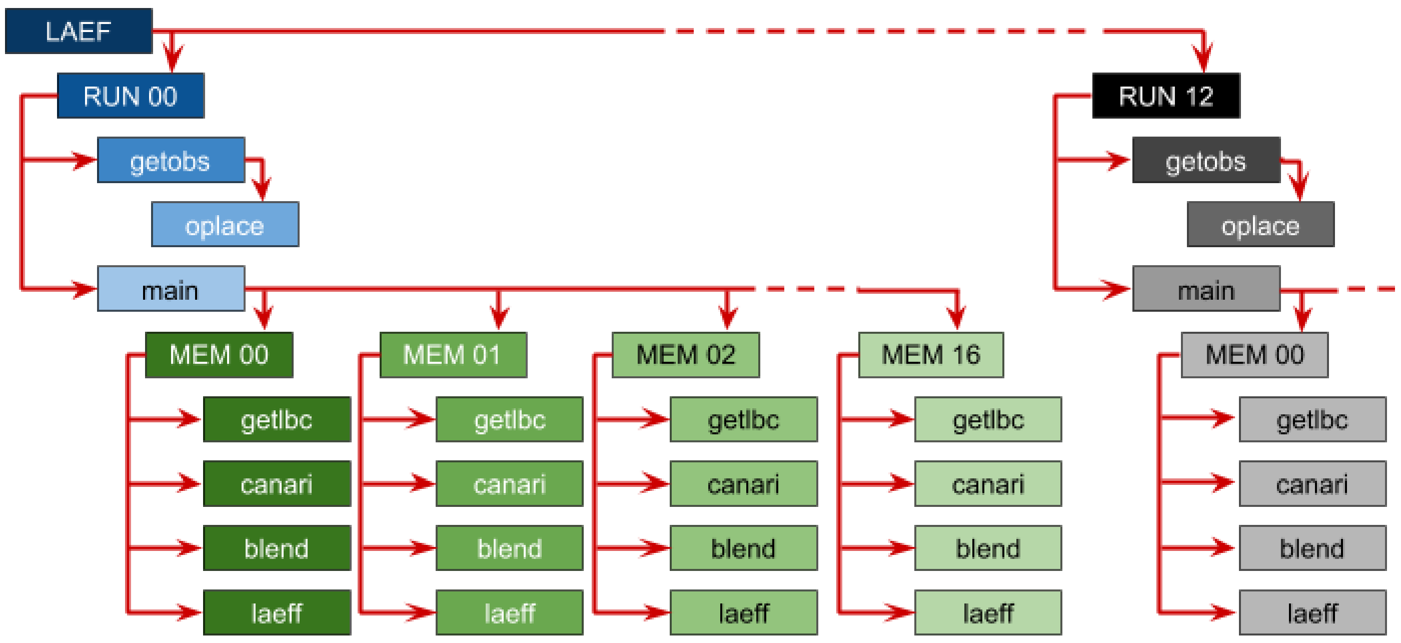
# Report on stay at ZAMG

30/07~24/08, 2018, Vienna, Austria

## ecFlow suite for new ALADIN-LAEF operations



**::Author**
Martin Belluš (SHMU)
martin.bellus@shmu.sk

## ::Table of Contents

## ::Acknowledgement

I very much appreciate the help from Alena Trojakova and Florian Weidle. I am also grateful to the whole ZAMG team for their hospitality and friendly atmosphere.

## ::Foreword

Enhanced ALADIN-LAEF system on 5 km grid and 60 vertical levels with recent ALARO-1 multi-physics under cy40t1, implemented stochastic physics for the surface prognostic variables and new approach for perturbing surface observations was prepared and tested in November last year (see Belluš, 2017: New ALADIN-LAEF phase I, RC LACE Report). That time the substantial increase of necessary computer resources prevented its operational implementation. However, during discussion at the last LSC meeting it was agreed that the cost represented by System Billing Units (SBUs) needed for new ALADIN-LAEF operations will be shared among the LACE members who dispose with some SBUs (i.e. ECMWF full members like HR, SI, AT) and cooperating Turkey (see Figure 1). Everything was successfully managed thanks to the initiative of new LACE PM Martina Tudor. LACE has got also new account for running time-critical applications at ECMWF HPCF (user *zla*). Therefore, the main goal for this 4-weeks stay was nothing less then the preparation of ecFlow suite for new time-critical application of ALADIN-LAEF system at ECMWF HPCF.

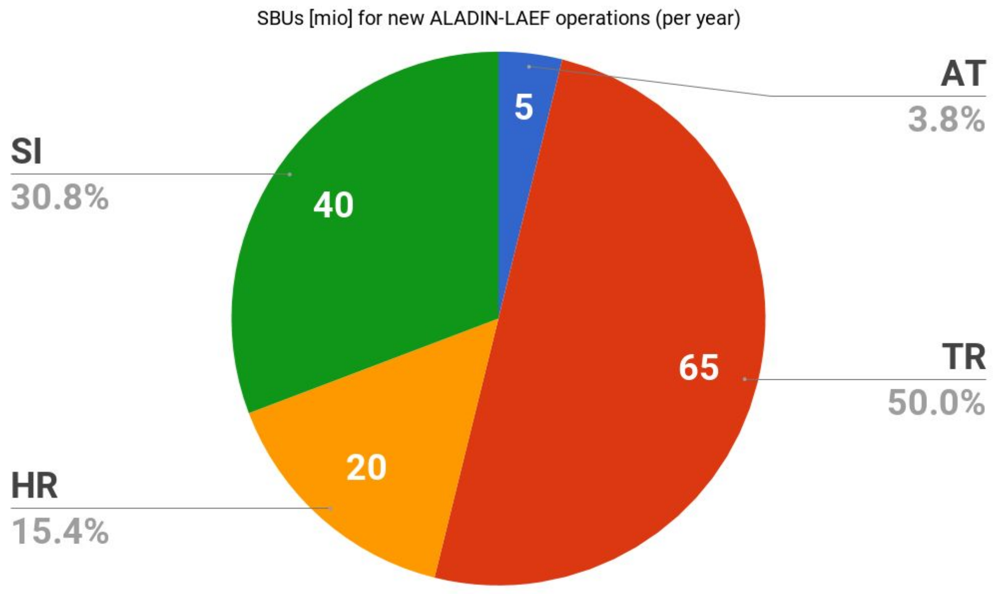


***Fig.1:*** *Expected distribution of SBUs needed for new ALADIN-LAEF operations at ECMWF HPCF among LACE partners and cooperating Turkey.*

## ::I New ALADIN-LAEF system specifications

The first phase (Phase I) of planned operational changes to ALADIN-LAEF system towards the higher resolution has already been started. Except the increased horizontal and vertical resolutions (5 km, 60 levels) this upgrade comprises many other enhancements like change to the linear grid, new version of model (cy40t1), IC uncertainty simulation by ESDA with the internal perturbation of OBS, model uncertainty simulation by combination of SPPT and ALARO-1 multi-physics, and so on. It is the second major upgrade since the beginning of its operations in 2011. The first upgrade happened in 2013, when the horizontal and vertical resolutions were increased from 18 to 11 km and 37 to 45 vertical levels respectively. That time also the LAEF domain has changed and an ensemble of surface data assimilation was implemented (Bellus et al. 2016). The most recent level of upgrade is summarized in Table 1.

***Tab. 1:*** *ALADIN-LAEF system specifications for current and new version (new version is expected to become pre-operational till the end of 2018 and operational in 2019).*

| ALADIN-LAEF | current | new |
|---|---|---|
| Code version | cy36t1 | cy40t1 |
| Horizontal resolution | 10.9 km | 4.8 km |
| Vertical levels | 45 | 60 |
| Number of grid points | 500x600 | 750x1250 |
| Grid | quadratic | linear |
| Time step | 450s | 180s |
| Forecast length | 72 h (00/12 UTC) | 72 h (00/12 UTC) |
| Members | 16+1 | 16+1 |
| IC perturbation | ESDA [surface], breeding-blending [upper-air] | ESDA [surface], blending (Phase I) / ENS BlendVar (Phase II) [upper-air] |
| Model perturbation | ALARO-0 multi-physics | ALARO-1 multi-physics + surface SPPT |
| LBC perturbation | ECMWF ENS | ECMWF ENS |

Current ALADIN-LAEF system has run from the beginning of its operations under SMS (Supervisor Monitoring Scheduler). However, the SMS becomes obsolete and its development has already been stopped. At the present time it is supported only on old and already tested platforms. Therefore, the SMS is being replaced by ecFlow. It uses object oriented methodology and modern standardised components. The proprietary scripting language used by SMS (CDP - Command and Display Program) has been replaced by Python, which is widely used in scientific and numeric computing. Moreover, the current ALADIN-LAEF suite has not been technically updated for several years. Thus we decided to start building our new operational ALADIN-LAEF suite from scratch rather than converting the old one from SMS to ecFlow. Another reason for such a key decision is being in quite big difference between the “current” and currently developed ALADIN-LAEF components. All of them are now written in Perl.

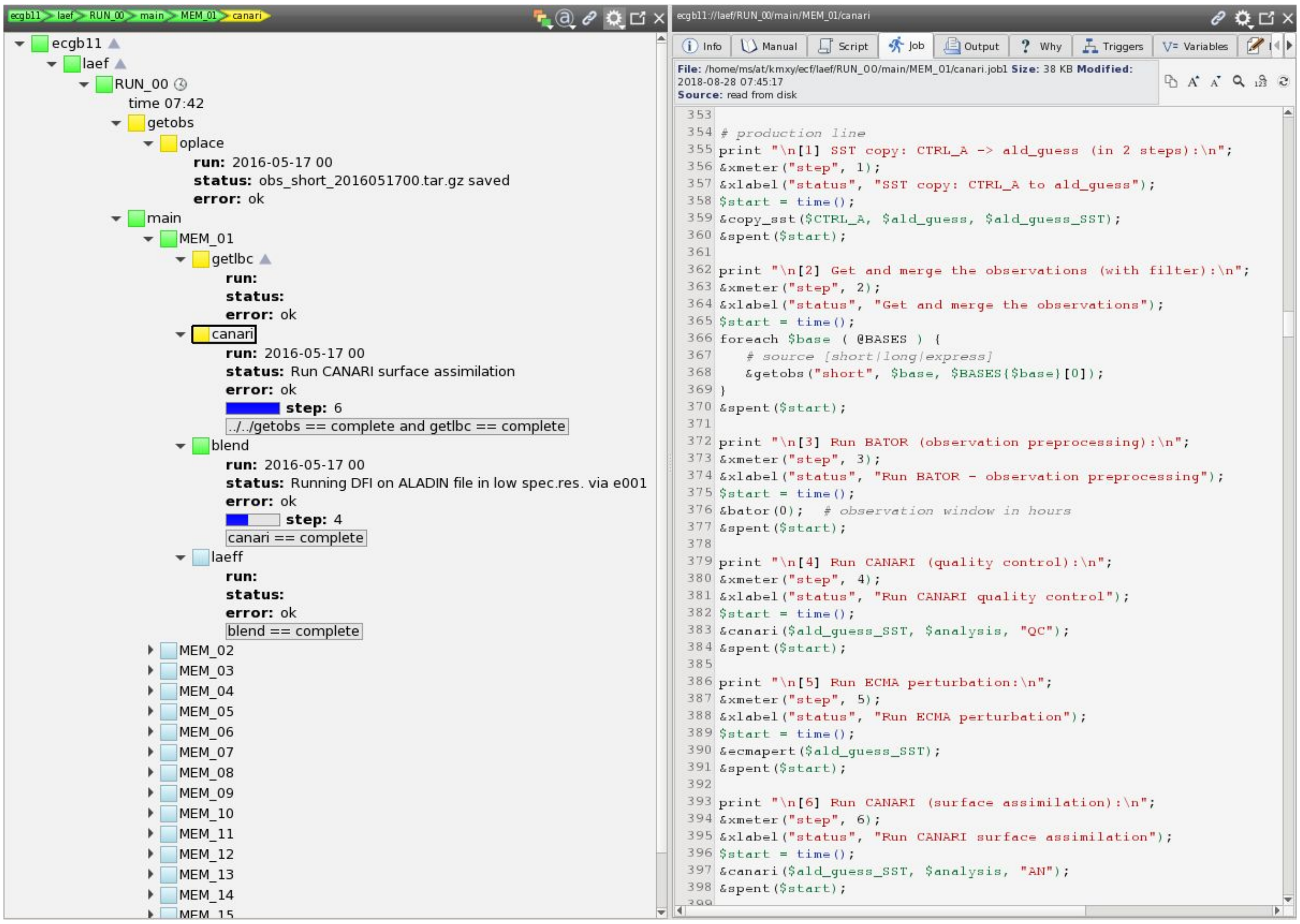
***Tab.2:*** *Technical characteristics of the main tasks running on HPCF (not including preprocessing and postprocessing).*

| Task | nproc | wallc time | output | SBUs | Total SBUs |
|---|---|---|---|---|---|
| | values per member (*per 12 hour integration) | | | | 16+1 mem (*72h, 2x day) |
| **canari** | 288 | 240-300s | 518 MB | 280-370 | ~11050 |
| **blend** | 288 | 480-540s | 518 MB | 660-720 | ~23460 |
| **laeff** | 288 | 900-960s | 7.15 GB | ~1200* | ~244800* |
| Total SBUs consumption per year (an approximation only) | | | | | **~102 mio** |

As one can see in the above table (Table 2), the total consumption for new ALADIN-LAEF operations per year would be approximately one hundred millions of billing units (not including preprocessing and postprocessing). That involves only the big tasks like ensemble of data assimilation, spectral blending and model integration. The additional small tasks to fetch the OBS files, convert LBCs and possibly to do some post processing as well are not included, but they should be relatively cheaper. It can be further estimated, that about 1.34 TB of disk space will be needed to store 1 day of full set of historical files for new ALADIN-LAEF system (72h forecast with hourly frequency for 16+1 members and 00/12 UTC runs). This volume does not include the assimilation and blending outputs. Their total volume per day would be only 34 GB (0.0336 TB). Additionally, there will be needed another 225 GB of available storage to keep all current boundary conditions for given day (2 runs). If we decide to store for safety reasons all of these data for current day, it would be about 1.6 TB in total.

## ::II Creating ecFlow suite at ECMWF HPCF

After the detailed study of ecFlow documentation available at ECMWF website, it was decided to follow the recommendations and proceed with building the suite definition file by Python script (instead of shell or other alternatives). It has many advantages, e.g. Python API allows the entire suite definition structure to be specified, checked and loaded into the ecFlow server (see Figure 2). On-the-fly generated ALADIN-LAEF suite definition file has over 1.3k lines, while the Python script which creates it has only about 250 lines of code. At the same time we have been able to use recently developed LAEF Perl “bricks” as native ecFlow tasks (including Perl modules for system setup and supporting functions), only with minor modifications involving necessary ecFlow client communication (see Figure 2 and 3). That helped a lot, since this counts all together more than 3k lines of reusable code.


ecgb11 > laef > RUN_00 > main > MEM_01 > canari
ecgb11://laef/RUN_00/main/MEM_01/canari
Info
Manual
Script
Job
Output
Why
Triggers
Variables

```
ecgb11
  laef
    RUN_00
      time 07:42
      getobs
        oplace
          run: 2016-05-17 00
          status: obs_short_2016051700.tar.gz saved
          error: ok
      main
        MEM_01
          getlbc
            run:
            status:
            error: ok
          canari
            run: 2016-05-17 00
            status: Run CANARI surface assimilation
            error: ok
            step: 6
            ../../getobs == complete and getlbc == complete
          blend
            run: 2016-05-17 00
            status: Running DFI on ALADIN file in low spec.res. via e001
            error: ok
            step: 4
            canari == complete
          laeff
            run:
            status:
            error: ok
            blend == complete
        MEM_02
        MEM_03
        MEM_04
        MEM_05
        MEM_06
        MEM_07
        MEM_08
        MEM_09
        MEM_10
        MEM_11
        MEM_12
        MEM_13
        MEM_14
        MEM_15
```

**File:** /home/ms/at/kmxy/ecf/laef/RUN_00/main/MEM_01/canari.job1 **Size:** 38 KB **Modified:** 2018-08-28 07:45:17
**Source:** read from disk

```perl

# production line
print "\n[1] SST copy: CTRL_A -> ald_guess (in 2 steps):\n";
&xmeter("step", 1);
&xlabel("status", "SST copy: CTRL_A to ald_guess");
$start = time();
&copy_sst($CTRL_A, $ald_guess, $ald_guess_SST);
&spent($start);

print "\n[2] Get and merge the observations (with filter):\n";
&xmeter("step", 2);
&xlabel("status", "Get and merge the observations");
$start = time();
foreach $base ( @BASES ) {
   # source [short|long|express]
   &getobs("short", $base, $BASES{$base}[0]);
}
&spent($start);

print "\n[3] Run BATOR (observation preprocessing):\n";
&xmeter("step", 3);
&xlabel("status", "Run BATOR - observation preprocessing");
$start = time();
&bator(0);  # observation window in hours
&spent($start);

print "\n[4] Run CANARI (quality control):\n";
&xmeter("step", 4);
&xlabel("status", "Run CANARI quality control");
$start = time();
&canari($ald_guess_SST, $analysis, "QC");
&spent($start);

print "\n[5] Run ECMA perturbation:\n";
&xmeter("step", 5);
&xlabel("status", "Run ECMA perturbation");
$start = time();
&ecmapert($ald_guess_SST);
&spent($start);

print "\n[6] Run CANARI (surface assimilation):\n";
&xmeter("step", 6);
&xlabel("status", "Run CANARI surface assimilation");
$start = time();
&canari($ald_guess_SST, $analysis, "AN");
&spent($start);
```

***Fig.2:** New ALADIN-LAEF suite (Phase I) under the ecFlow environment (ecFlow GUI screenshot). Suite definition file is generated by Python code, while all tasks, include files and configuration modules are written in Perl.*

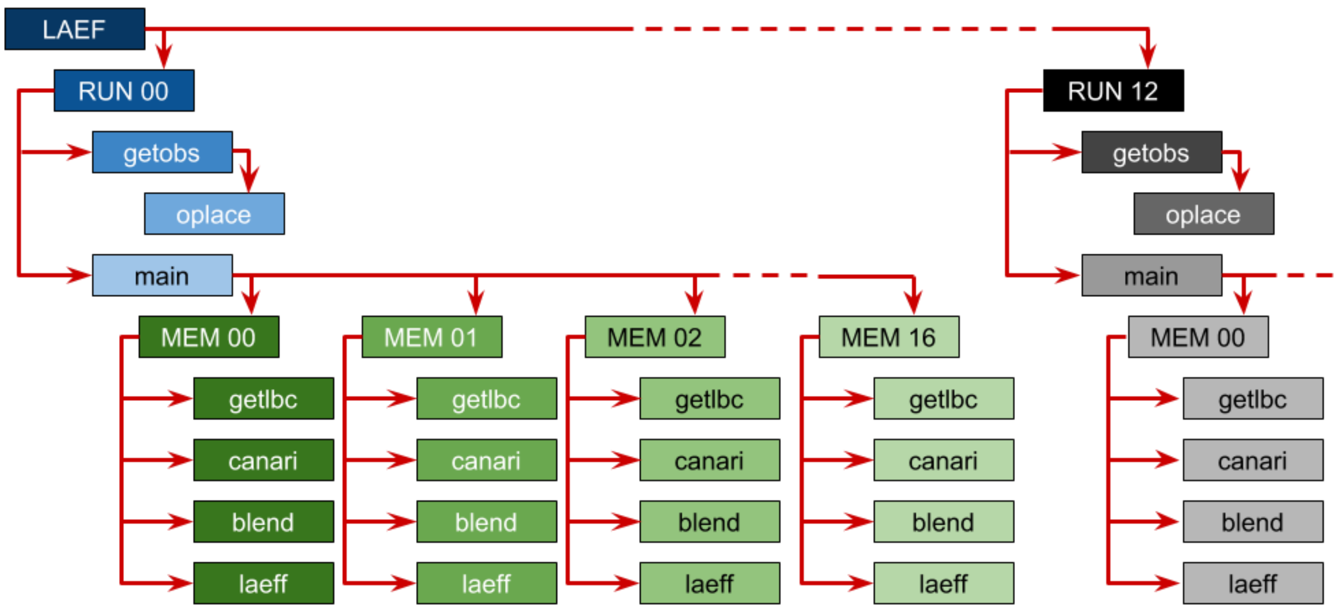


***Fig.3:** New ALADIN-LAEF suite (Phase I) flow chart depicting the particular tasks dependencies.*

Before even starting we have to check whether the ecflow_server is running under our user. We can do so either by looking at the processes (*ps -fu <user>*) or submitting a command *ecflow_client --ping --host=ecgb11 --port=4147*, where ecgb11 is our host and 4147 is user specific ID (both uniquely identify the ecFlow server). In case the ecFlow server is not running, it can be launched by the script *ecflow_start.sh* (localized e.g. in */usr/local/apps/ecflow/4.9.0/bin*).

Since ecFlow environment is not set up by default for ecgate's users, it is also necessary at the beginning of each session to export some ecFlow variables and load appropriate modules. This can be done by sourcing following shell script:

```
export ECF_HOST=ecgb11
export ECF_PORT=4147
export ECF_LOGPORT=37647

echo "ECF_HOST="$ECF_HOST
echo "ECF_PORT="$ECF_PORT
echo "ECF_LOGPORT="$ECF_LOGPORT

module load ecflow
module load schedule
```

Now we can open the ecFlow graphical user interface with the command *ecflow_ui* (an example of such GUI with loaded ALADIN-LAEF suite can be seen in Figure 2).

Suite definition file is dynamically generated by Python script (see also Appendix), which was written during the stay at ZAMG. The ECF_MICRO character used for the code preprocessing was redefined to "^" (caret). The reason is to have the least interaction with the original Perl code. The default setting for ECF_MICRO is "%" (percent), which is used for the hash associative arrays in Perl and thus is widely spread in the code. Even now, all the original caret signs (e.g. in some regular expressions) must have been exchanged by "^^" (caret caret) to avoid their substitution by ecFlow preprocessing.

In order to establish the communication between the ecFlow server and all LAEF jobs, we had to implement the ecflow_client commands into the original Perl scripts (e.g. in canari.pl, blend.pl and laeff.pl). In the following lines you are going to see some examples.

**^include <pbs.h> - ASCII**
Inclusion of the PBS header with the directives for queuing system on HPCF. The PBS header may look like this (note that it can contain some ECF or other variables which are substituted by the preprocessing):

```
#------------------------------------------
#PBS -N ^NAME^
#PBS -A ^ACCOUNT^
```

```
#PBS -q ^CLASS^
#PBS -m a
#PBS -M martin.bellus@gmail.com
#PBS -l EC_total_tasks=^NP^
#PBS -l EC_hyperthreads=1
#PBS -l EC_threads_per_task=1
#PBS -o ^ECF_JOBOUT^
#PBS -j oe
#------------------------------------------
```

**^include <head.pl> - Perl**
Here are defined ecflow_client commands to be used further from within our Perl code:

```
sub xinit() {
  system("ecflow_client --init=$$");
}

sub xabort() {
  system("ecflow_client --abort=$$");
}

sub xcomplete() {
  system("ecflow_client --complete");
}

sub xmeter($$) {
  my $name  = shift;
  my $value = shift;
  system("ecflow_client --meter=$name $value");
}

sub xevent($) {
  my $n = shift;
  system("ecflow_client --event=$n");
}

sub xlabel($$) {
  my $name  = shift;
  my $value = shift;
  system("ecflow_client --label $name $value");
}
```

Some ENV, ECF (ecFlow) and CNF (LAEF configuration variables) are also set and exported from here and finally &xinit() is called to tell the ecFlow we have started.

**^include <tail.pl> - Perl**
This include is located usually at the end of script. It catches the signal and calls the &xabort() in case of issues, otherwise it reports to ecFlow about successful task completion by invoking &xcomplete().

Our ecFlow task scripts are also written in Perl in order to get the most out of the existing new LAEF scripting system. The following modifications have been included into the Perl scripts to fully benefit from the ecFlow functionality.

All "*exit(1)*" instances (i.e. fatal errors) were exchanged by sequence:
```
&xlabel(〝error〞, 〝...appropriate error message…〞);
&xabort();
exit(1);
```

Some useful reporting to ecFlow was implemented via *labels*:
```
&xlabel("run", "^YYYY^-^MM^-^DD^ ^HH^");
```

A processing state of the tasks was implemented using the combination of ecFlow *meters* and *labels* as well, therefore it can be monitored directly from ecflow_ui:
```
&xmeter("step", 1);
&xlabel("status", "Converting ECMWF file to low spec.res. via ee927");
```

All the ecFlow scripts (Perl application tasks, Python script to generate the LAEF suite definition file and the include files) are stored on ecgate server. The LAEF suite is generated there and also loaded into the ecFlow server. On the other hand, the compiled binaries, namelists and input files must be located on HPCF cluster (cca), because this is where the jobs are executed under the queueing system (PBS). For more technical details please refer to the Appendix.

## ::Conclusions

The first functional ALADIN-LAEF ecFlow suite was created and tested under *kmxy* user on ecgate/cca. Now it must be transferred under the dedicated LACE operational user *zla,* while the next steps towards the time critical (TC) application must be done already in cooperation with the user support section at ECMWF. This is planned in the very near future. There are also still several technical details which must be addressed. The whole system must be tested under TC environment together with real data flow. New products must be specified, set-up and their distribution to the LACE partners have to be arranged. The preoperational run in parallel with the old ALADIN-LAEF suite is highly desirable and it would be great if it can happen by the end of this year.

## ::References

- Framework for time-critical applications (ECMWF Technical Notes)
- https://confluence.ecmwf.int/display/ECFLOW/User+Manual (ecFlow User Manual)
- Iain Russell, Sándor Kertész, 2017: ecFlow training course 2017
- Axel Bonet, John Hodkinson, 2018: ecFlow course 2018
- Axel Bonet, 2018: ecflow - Python

## ::Appendix

Scripts on ecgate

**Root:** /home/ms/at/kmxy/ecf/ (<ecf_root> further on)

**Apps:** <ecf_root>/app

```
-rwxr-x---. 1 kmxy at 25839 Aug 22 12:33 blend.pl
-rwxr-x---. 1 kmxy at 40090 Aug 23 10:02 canari.pl
-rwxr-x---. 1 kmxy at  7743 Aug 24 13:43 getlbc.pl
-rwxr-x---. 1 kmxy at 16214 Aug 22 12:34 laeff.pl
-rwxr-x---. 1 kmxy at  1929 Aug 23 19:25 link_apps_to_tasks.pl
-rwxr-x---. 1 kmxy at  4934 Aug 24 12:27 oplace.pl
```

**Includes:** <ecf_root>/include

```
-rw-r-----. 1 kmxy at 1573 Aug 21 16:45 head.pl
-rw-r-----. 1 kmxy at  294 Aug 22 10:57 pbs.h
-rw-r-----. 1 kmxy at  183 Aug 13 10:41 tail.pl
```

**Suite definition:** <ecf_root>/def

```
-rw-r-----. 1 kmxy at  6328 Aug 28 09:26 create_laef.py
-rw-r-----. 1 kmxy at 35283 Aug 28 09:26 laef.def
-rw-r-----. 1 kmxy at   938 Aug  3 14:31 reload_laef.py
```

Suite definition file (laef.def) is being dynamically created by Python script create_laef.py and loaded to the ecFlow server by another Python script reload_laef.py.

**Suite:** <ecf_root>/laef

This LAEF suite file system is dynamically created by Perl script link_apps_to_tasks.pl (<ecf_root>/app) and contains all necessary tasks. These tasks are then directly submitted to the queuing system of HPCF. Each task is only a symbolic link to the appropriate application script (<ecf_root>/app) and therefore the whole <ecf_root>/laef file system can be anytime deleted and subsequently recreated from scratch.

**Logs:** <ecf_root>/log

ECF log files and check points.

Files on cca

**Root:** /home/ms/at/kmxy/app_LAEF5F (<cca_root> further on)

```
drwxr-x--- 4 kmxy at 4096 Apr 26  2017 blend/nam - blending namelists
drwxr-x--- 4 kmxy at 4096 May  5  2017 canari/nam - assim namelists and constant files
drwxr-x--- 3 kmxy at 4096 Aug 23 17:37 getlbc/nam - namelists for LBC conversion
drwxr-x--- 4 kmxy at 4096 Apr 26  2017 laeff - e001 namelists (multi-physics)
drwxr-x--- 2 kmxy at 4096 Sep 12 15:56 setup - Conf_app.pm, Support.pm
```

The path to LAEF setup directory (<cca_root>/setup) must be defined in ecFlow suite on ecgate via <ecf_root>/include/head.pl script:

```
# set path to (app) Perl configuration modules on CCA (!)
use lib qw(
  /home/ms/at/kmxy/app_LAEF5F/setup
);
```

Conf_app.pm (Perl module) is the main suite configuration file where the executable paths are defined together with working directory, INPUT/OUTPUT directories, expiration times, LBC path, etc. Support.pm is a Perl module containing several functions which are repeatedly used within the LAEF applications (e.g. &modif_namel, &shift_date, &gettrunc, etc.).