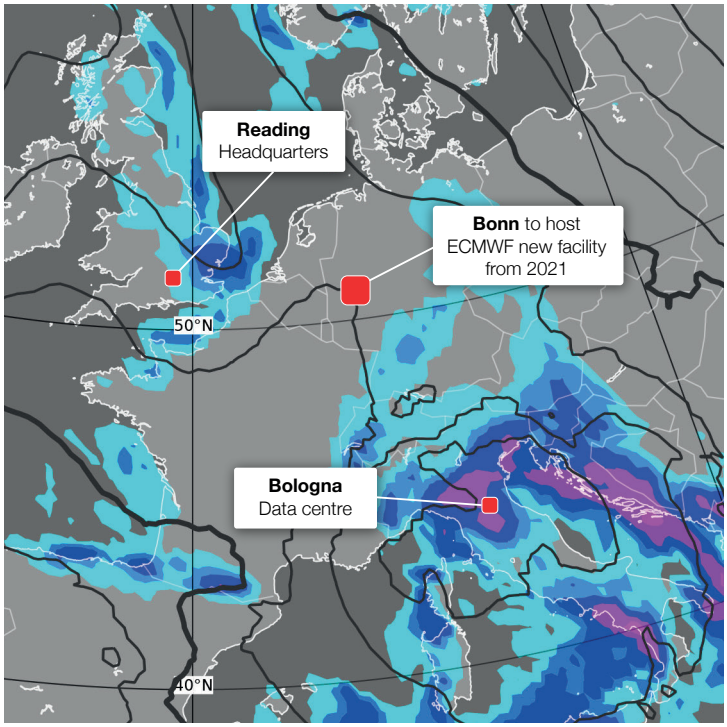
ECMWF Feature article

from Newsletter Number 166 – Winter 2020/21

COMPUTING

ecFlow 5 brings benefits to Member States



www.ecmwf.int/en/about/media-centre/media-resources

doi: 10.21957/4tba2hz3f7

This article appeared in the Computing section of ECMWF Newsletter No. 166 – Winter 2020/21, pp. 33–35.

ecFlow 5 brings benefits to Member States

Avi Bahra, Iain Russell, Sándor Kertész

Managing workflows for large-scale data-intensive computational processes is an ever-growing challenge. These workflows must be repeatable, highly available, monitorable and accurate, while still allowing the flexibility to support changes. At ECMWF this challenge has been met with ecFlow, a workflow package developed in-house to meet the ever-changing requirements of the Centre and its Member and Co-operating States.

ecFlow was designed for general use but has been sculpted by the operational and research needs of weather and climate science. For example, at ECMWF it is used for many purposes including research experiment runs, operational model runs, data post-processing and archiving, and software builds.

ecFlow enables users to run a large number of programs, with dependencies on each other and on time, in a controlled environment. It provides good tolerance for hardware and software failures and allows for controlled restarts. The server, client and graphical user interface (GUI) are highly scalable and can handle workflows with hundreds of thousands of tasks. ecFlow is open source and is written in C++ for optimum performance. It runs on UNIX platforms, with many years of experience on Linux and more recent usage on macOS.

Version 5 of ecFlow brings many modernisations and improvements in terms of features, performance, security and maintainability.

ecFlow's architecture

ecFlow has a client/server architecture (Figure 1). An ecFlow server is responsible for several suites, each a hierarchical collection of tasks. Complex suites can be defined using a Python API that guarantees their syntactic correctness (Figure 2). Simpler suites can be defined through plain text files. The server submits tasks to the machines where they will run, receiving updates as they proceed. Tasks can be defined in any scripting language, for example shell or Python. These scripts can be parametrized, meaning that the same script can be used for many different tasks, with different settings. For example, a script variable 'FORECAST_STEP' could be set to 6 when run in one task and 12 in another. The scripts may also have embedded ecFlow commands that communicate their status back to the server, e.g. to show progress or to trigger another task to start. Sophisticated use of these embedded commands allows tasks to dynamically modify the server's suites, facilitating an adaptive workflow without requiring manual loading of revised suites into the server. ecFlow is not tied to any particular queueing system that may sit in front of the worker machines, but its tasks can be submitted to any such queueing system through the use of a general submission script. ecFlow client applications include a graphical user interface, ecFlowUI, and a command-line program, ecflow_client, both of which can be used to query and modify the server.

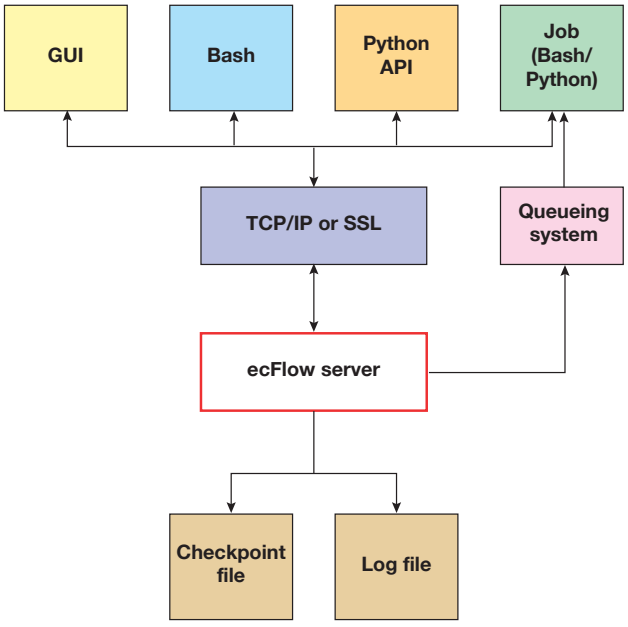


Figure 1 Various clients (GUI, Bash, Python API) can communicate bi-directionally with an ecFlow server using standard Transmission Control Protocols/Internet Protocol (TCP/IP) or Secure Sockets Layer (SSL) protocols. The server can run tasks directly or submit them to a queueing system; either way, they can still communicate back to the server. The server keeps track of events in a log file, providing the basis for statistical analyses of past events, such as the average duration of a given task. A checkpoint file is written to disk at regular intervals, providing a backup of the server's internal state at that moment; this mechanism can also be used to provide continuity when upgrading a server to a newer version of ecFlow.

doi: 10.21957/4tba2hz3f7

```
import os
from ecflow import Defs,Suite,Family,Task,Edit,Trigger,Client

home = os.path.join(os.getenv("HOME"), "example")
defs = Defs(
        Suite("test",
            Edit(ECF_HOME=home),
            Family("f1",
                Edit(SLEEP=20),
                Task("t1"),
                Task("t2",Trigger("t1 == complete")))))

ci = Client()
ci.load(defs)
```

Figure 2 A simple example of a Python script that creates a new suite consisting of a family of two tasks, the second of which will be run as soon as the first has completed. The suite is then loaded onto a server using default settings.

Graphical user interface

ecFlowUI is the graphical user interface to ecFlow (Figure 3). It is written with the C++ Qt library. ecFlowUI supports real-time monitoring of the workflow, allowing jobs to be started, suspended and terminated. Many aspects can be edited on the fly, including the job scripts themselves and their associated variables. Live and historical job output can be viewed with an efficient built-in viewer that can handle output files of arbitrary size. Dependencies between nodes can be visualised in graphical form, and a built-in log analyser can aid in fine-tuning the workflow.

ecFlowUI can monitor several ecFlow servers at once, with facilities to display only those suites or tasks of interest. It can also be used to move a set of tasks from one server to another.

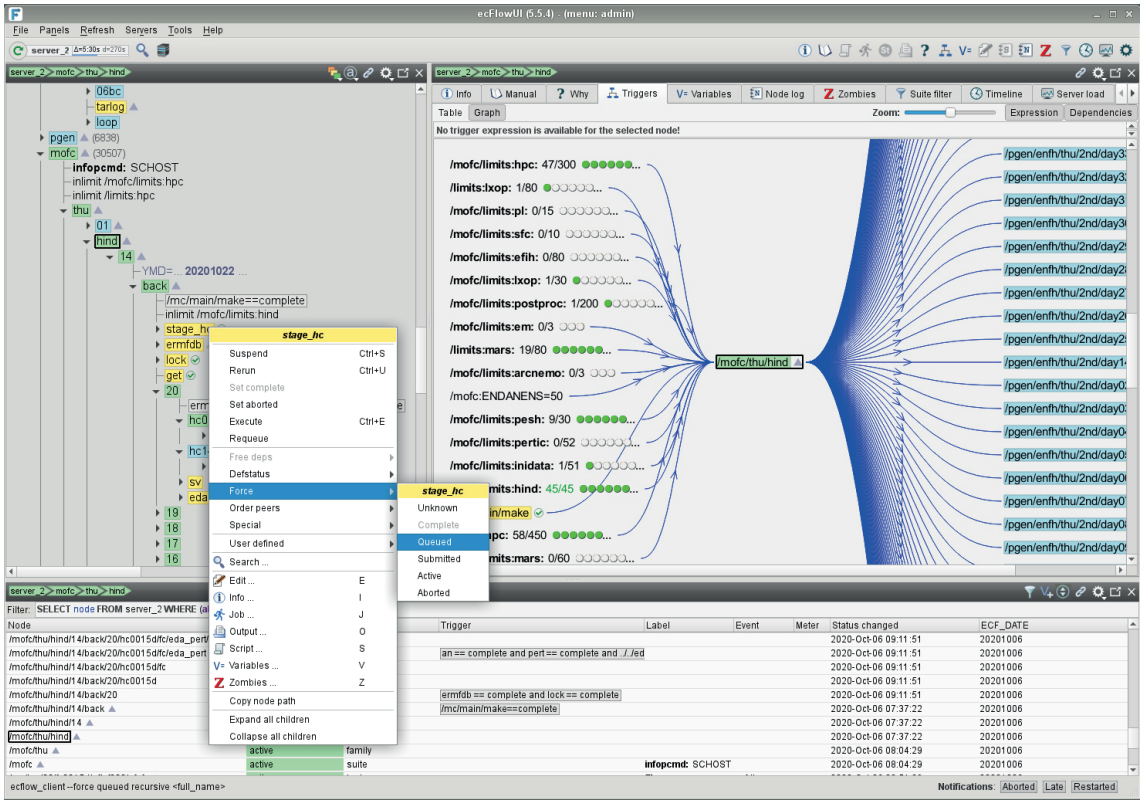


Figure 3 ecFlowUI provides a rich environment for viewing and interacting with suites, including a new Trigger Graph view showing dependencies between items in the suite.

ecFlow version 5

One limiting factor of ecFlow 4 was that its client/server communication was sensitive to changes in the version of the boost library that it links with. This meant that a single client could not necessarily communicate with all the running servers if they had been built with different versions of boost. The technology also limited the ability to make even small changes in communication protocol, which is sometimes necessary in order to allow new features. ecFlow 5 now uses the JSON format for communication, and clients and servers are free to use different versions of boost. This change also allows for new features to be added without breaking compatibility with older servers or clients. With further improvements to the communication, ecFlowUI can now communicate with servers using fewer network requests, meaning less network traffic. An internal improvement is that ecFlow 5 uses features from the C++14 standard, simplifying some code and providing performance benefits.

ecFlow 5 has a number of additional new features requested both by ECMWF users and by Member and Co-operating States. These include:

- Improved security features, such as integrated SSL and password-based access; ecFlowUI can now view both SSL and non-SSL based servers in the same session.
- ecFlowUI now has an interactive trigger graph view to show the interdependency of nodes and attributes.
- Servers now support auto-archive and auto-restore, allowing parts of a suite to be dynamically written to disk when complete and restored later on. This aids the handling of extremely large suites.
- Improved features to help users diagnose problems, for example when a worker machine goes down or a running job becomes detached from the server's records.
- Additional controls to limit the number of submitted or active tasks.
- Various smaller features to help refine the suite definitions.

ecFlow's stability has been validated by daily operational use at ECMWF. In addition, a slew of tests are run every night to ensure that no regressions creep into its releases. With its maturity and proven fitness-for-purpose, future work on ecFlow will emphasise the continuation of this maintenance and stability rather than large new developments.

Migrating to ecFlow 5

Many operational servers at ECMWF have already been migrated to ecFlow version 5. Once ECMWF's computing centre has moved to Bologna, only version 5 will be available. Fortunately, migration from ecFlow 4 to 5 is straightforward and mostly involves stopping the currently running server and then starting it up again using ecFlow 5. The migration page provides more details (*https://confluence.ecmwf.int/display/ECFLOW/Migration+to+ecflow+5*). It is important to note that only an ecFlowUI from version 5 can be used with a version 5 server due to the change in communication protocol. Also noteworthy is that although current versions of ecFlow are built with Python 2 and 3 support, once operational in Bologna only Python 3 will be supported. It is therefore advisable to ensure that any suites are migrated as soon as possible in order to avoid any last-minute problems.

Availability

ecFlow is installed on all of ECMWF's computing platforms, including the Member and Co-operating State server ecgate. If you plan to run an operational ecFlow server at ECMWF, please contact User Services, who will be glad to guide you on the best way to set it up. There are currently a default and a new version of ecFlow 5. To use either one of these, use the commands:

```
module load ecflow/5
```

```
module load ecflow/5new
```

For use external to ECMWF's computing platforms, ecFlow is also available as a binary installation on the conda platform, available through the conda-forge channel with this command:

```
conda install ecflow -c conda-forge
```

The source is also available on github (*https://github.com/ecmwf/ecflow*) or as a tarball from the ecFlow Confluence pages (*https://confluence.ecmwf.int/display/ECFLOW/*).

doi: 10.21957/4tba2hz3f7

doi: 10.21957/4tba2hz3f7