APEL Development Process and Software Release Quality Verification Procedure

Author:	Adrian Coveney
Version:	FINAL
Document Link:	https://documents.egi.eu/document/2739

DOCUMENT LOG

Issue	*Date*	*Comment*	*Author*
v0.1	2016-01-21	Document creation	A. Coveney / STFC
v0.2	2016-01-21	Incorporated revisions from S. Pullinger	A. Coveney / STFC
v0.3	2016-02-01	Revised following internal review	A. Coveney / STFC
FINAL	2016-02-23	Revised following external review	A. Coveney / STFC

TERMINOLOGY

The EGI glossary of terms is available at: https://wiki.egi.eu/wiki/Glossary

Contents

1 Introduction

APEL is an accounting tool that collects accounting data from sites participating in the EGI and WLCG infrastructures as well as from sites belonging to other Grid organisations that are collaborating with EGI, including OSG, NorduGrid and INFN.

The accounting information is gathered from different sensors into a central accounting database where it is processed to generate statistical summaries that are available through the EGI/WLCG Accounting Portal.

Statistics are available for view in different detail by Users, VO Managers, Site Administrators and anonymous users according to well-defined access rights.

This document describes the development process used by the APEL project including the semi-automatic testing procedure used to assess the quality of software releases.

2 Software overview

2.1 Components

The APEL project produces its own software, which is written in Python and uses MySQL as the database backend. Source code is hosted on GitHub under the APEL organization[1]. The software comprises the components detailed later in this document. Figure 1 shows these components and how they interact. All of them are part of the same repository on GitHub[2], except for the apel-ssm component, which can be used without the other components so warrants its own repository[3].

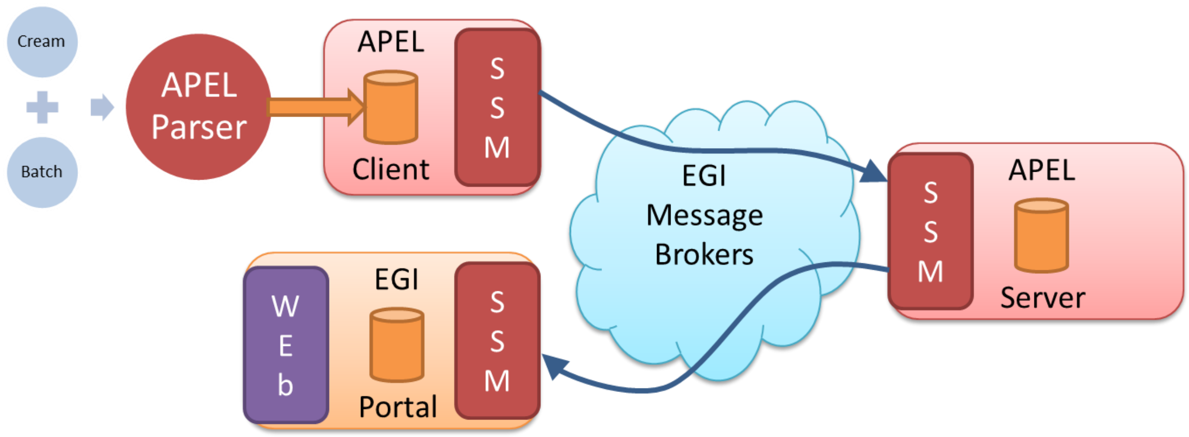

Figure 1 - APEL components and their interaction

2.1.1 apel-parsers

The parsers extract data from these batch systems:

- LSF 5.x to 9.x
- PBS/Torque
- Univa/Sun/Oracle Grid Engine
- SLURM
- HTCondor

and place it in the client database.

[1] https://github.com/apel
[2] https://github.com/apel/apel
[3] https://github.com/apel/ssm

2.1.2 apel-client

The client processes the data and sends it to the central APEL server using SSM.

2.1.3 apel-server

The server processes data sent by the clients and sends it on to the EGI Accounting Portal using SSM.

2.1.4 apel-lib

This contains all the library Python code for apel-parsers, apel-client and apel-server.

2.1.5 apel-ssm

This is a secure implementation of the STOMP messaging protocol. It consists of a library used by the apel-client and command line scripts called by externally developed clients.

3 Software development workflow

3.1 Forks

As Git is a distributed version control system, all the developers who work on the APEL project have their own copy of the repositories, a fork, in their own GitHub accounts. The developers work on local copies of these forks, fixing bugs or creating new features.

3.2 Pull Requests

When the changes a developer has been working on are ready to be merged back into the parent repository a pull request is opened. The developer should include information about the changes, such as their purpose and whether they address an outstanding issue, so that someone else can understand the context of these changes. Where new features are added, they should be covered by a corresponding unit test.

Opening the pull request initiates the execution of a number of checks. The main one is the execution of the test suite using the hosted continuous integration service Travis CI[4]. Code test coverage checking is performed by Coveralls[5] and Python code quality checks by Landscape[6]. These tools report the result of their checks directly in the pull request for the developers to see. The continuous integration test must pass before the changes can be merged back into the parent and it is highly recommended that the other checks also pass.

The changes are reviewed by at least one other member of the APEL team who did not submit the pull request. This is so that at least two people have seen or worked on the changes that are to be added. After this stage, the reviewer can either approve the changes, or suggest improvements.

If approved, then the changes are merged into the parent repository by the team member with the release manager role. If not approved, then the developer can incorporate the suggestions and add more changes to the pull request which leads to the automated checks being made again and then the process can repeat from there until the reviewer is satisfied with the suitability of the changes.

3.3 Branches

Both of the main APEL repositories have two branches used to manage the source code:

[4] https://travis-ci.org/
[5] https://coveralls.io/
[6] https://landscape.io/

3.3.1 Development branch

The development branch (shortened to "dev" in the version control system) is where pull requests are merged to and so contains the latest features as they are completed. Therefore the code in this branch should always be deployable to test systems.

3.3.2 Master branch

The master branch is where the development branch is merged to when preparing the software for a release. Therefore the code in this branch should always be releasable to production systems.

3.4 Further testing

Extra testing can be performed using a test system if it is thought that the changes are not tested comprehensively enough in the unit tests or if there are potential integration issues. The APEL project has a test server where new versions of the software are installed so that external developers can test against them before deploying to production.

4 Service management

4.1 Production deployment

Once a new software release has passed the automated testing and any necessary additional testing, it can be deployed to the production system. This involves stopping any services that are being updated, using RPM to upgrade the packages, and then restarting the services.

If a problem is noticed, it is possible roll back the changes by stopping the services and using RPM again to re-install the preceding package.

If the changes to the software require an update to the database schema, the database should first be backed up to provide an easy way to return to the pre-upgrade state in case of problems.

4.2 Data management

A database backup of the central accounting repository is taken daily which is archived to tape. Along with the database transaction logs, this allows the database to be recovered to any point in time.