

**EGI-Engage**

Final release of the accounting and operational tools

D3.17

|  |  |
| --- | --- |
| **Date** | 21 August 2017 |
| **Activity** | WP3 |
| **Lead Partner** | EGI Foundation |
| **Document Status** | FINAL |
| **Document Link** | <https://documents.egi.eu/document/3037>  |

Abstract

This deliverable describes the final release of the EGI Accounting and Operational Tools during EGI-Engage project, including the developments made during the third year of the project for the Operations Portal, ARGO, Messaging, GOCDB, Security Monitoring, Accounting Repository and Portal. The evolution of these tools has been driven by the need to support new technologies (e.g. cloud) and to satisfy new requirements emerging from service providers and user communities, in particular from the Research Infrastructures contributing to EGI-Engage via the EGI Competence Centres (CCs) and the Resource Providers (RPs) who contribute infrastructure services to the federation. The development roadmap has been reviewed and updated according to a requirement gathering process, which has been accomplished in collaboration with the other EGI Engage WPs in charge of the communication with users and key stakeholders.

**COPYRIGHT NOTICE**



This work by Parties of the EGI-Engage Consortium is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/). The EGI-Engage project is co-funded by the European Union Horizon 2020 programme under grant number 654142.

**DELIVERY SLIP**

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Name*** | ***Partner/Activity*** | ***Date*** |
| **From:** | Cyril Lorphelin Themis Zamani George RyallDaniel KourilAdrian CoveneyIvan Diaz AlvarezDiego Scardaci | CNRS/WP3GRNET/WP3STFC/WP3CESNET/WP3STFC/WP3CSIC/WP3EGI F.-INFN/WP3 | 29/07/2017 |
| **Moderated by:** | Malgorzata Krakowian  | EGI Foundation |  |
| **Reviewed by** | Alessandro Paolini | EGI Foundation | 10/08/2017 |
| **Approved by:** | AMB and PMB |  |  |

**DOCUMENT LOG**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Issue*** | ***Date*** | ***Comment*** | ***Author/Partner*** |
| **v.1** | 29/07/2017 | Full draft ready for internal review | Cyril Lorphelin/CNRS Themis Zamani /GRNET George Ryall /STFCDaniel Kouril/CESNETAdrian Coveney/STFCIvan Diaz Alvarez/CSICDiego Scardaci/EGI F. - INFN |
| **v. 2** | 01/08/2017 | Full draft ready for external review | Diego Scardaci/EGI F. - INFN |
| **FINAL** | 22/08/2017 | Final version | Diego Scardaci/EGI F. - INFN |

**TERMINOLOGY**

A complete project glossary and acronyms are provided at the following pages:

* <https://wiki.egi.eu/wiki/Glossary>
* <https://wiki.egi.eu/wiki/Acronyms>

**Contents**

[1 Operations Portal 8](#_Toc491164297)

[1.1 Introduction 8](#_Toc491164298)

[1.2 Service architecture 9](#_Toc491164299)

[1.2.1 High-Level Service architecture 9](#_Toc491164300)

[1.2.2 Integration and dependencies 11](#_Toc491164301)

[1.3 Release notes 11](#_Toc491164302)

[1.3.1 Operations Portal 4.0 11](#_Toc491164303)

[1.3.2 Operations Portal 4.1 12](#_Toc491164304)

[1.3.3 Operations Portal 4.2 12](#_Toc491164305)

[1.3.4 VAPOR 2.0 13](#_Toc491164306)

[1.3.5 VAPOR 2.1 13](#_Toc491164307)

[1.3.6 VAPOR 2.2 13](#_Toc491164308)

[1.3.7 VAPOR 2.3 13](#_Toc491164309)

[1.4 Feedback on satisfaction 14](#_Toc491164310)

[1.5 Plan for Exploitation and Dissemination 15](#_Toc491164311)

[1.6 Future plans 16](#_Toc491164312)

[2 ARGO 18](#_Toc491164313)

[2.1 Introduction 18](#_Toc491164314)

[2.2 Service architecture 18](#_Toc491164315)

[2.2.1 High-Level Service architecture 18](#_Toc491164316)

[2.2.2 Integration and dependencies 20](#_Toc491164317)

[2.3 Release notes 21](#_Toc491164318)

[2.3.1 Requirements covered in the release 21](#_Toc491164319)

[2.4 Feedback on satisfaction 24](#_Toc491164320)

[2.5 Plan for Exploitation and Dissemination 24](#_Toc491164321)

[Future plans 26](#_Toc491164322)

[3 Messaging Service 28](#_Toc491164323)

[3.1 Introduction 28](#_Toc491164324)

[3.2 Service architecture 28](#_Toc491164325)

[3.2.1 High-Level Service architecture 28](#_Toc491164326)

[3.2.2 Integration and dependencies 32](#_Toc491164327)

[3.3 Release notes 32](#_Toc491164328)

[3.3.1 Requirements covered in the release 32](#_Toc491164329)

[3.3.2 Changelog 33](#_Toc491164330)

[3.4 Feedback on satisfaction 33](#_Toc491164331)

[3.5 Plan for Exploitation and Dissemination 33](#_Toc491164332)

[3.6 Future plans 35](#_Toc491164333)

[4 GOCDB 36](#_Toc491164334)

[4.1 Introduction 36](#_Toc491164335)

[4.2 Service architecture 36](#_Toc491164336)

[4.2.1 High-Level Service architecture 36](#_Toc491164337)

[4.2.2 Integration and dependencies 37](#_Toc491164338)

[4.3 Release notes 37](#_Toc491164339)

[4.3.1 Requirements covered in the release 37](#_Toc491164340)

[4.4 Feedback on satisfaction 38](#_Toc491164341)

[4.5 Plan for Exploitation and Dissemination 38](#_Toc491164342)

[4.6 Future plans 39](#_Toc491164343)

[5 Security Monitoring 40](#_Toc491164344)

[5.1 Introduction 40](#_Toc491164345)

[5.2 Service architecture 40](#_Toc491164346)

[5.2.1 High-Level Service architecture 40](#_Toc491164347)

[5.2.2 Integration and dependencies 41](#_Toc491164348)

[5.3 Release notes 41](#_Toc491164349)

[5.3.1 Requirements covered in the release 41](#_Toc491164350)

[5.4 Feedback on satisfaction 41](#_Toc491164351)

[5.5 Plan for Exploitation and Dissemination 41](#_Toc491164352)

[5.6 Future plans 42](#_Toc491164353)

[6 Accounting Repository 43](#_Toc491164354)

[6.1 Introduction 43](#_Toc491164355)

[6.2 Service architecture 44](#_Toc491164356)

[6.2.1 High-Level Service architecture 44](#_Toc491164357)

[6.2.2 Integration and dependencies 45](#_Toc491164358)

[6.3 Release notes 45](#_Toc491164359)

[6.3.1 Requirements covered in the release 45](#_Toc491164360)

[6.4 Feedback on satisfaction 45](#_Toc491164361)

[6.5 Plan for Exploitation and Dissemination 46](#_Toc491164362)

[6.6 Future plans 47](#_Toc491164363)

[7 Accounting Portal 48](#_Toc491164364)

[7.1 Introduction 48](#_Toc491164365)

[7.2 Service architecture 48](#_Toc491164366)

[7.2.1 High-Level Service architecture 48](#_Toc491164367)

[7.2.2 Integration and dependencies 52](#_Toc491164368)

[7.3 Release notes 53](#_Toc491164369)

[7.3.1 Requirements covered in the release 53](#_Toc491164370)

[7.4 Feedback on satisfaction 53](#_Toc491164371)

[7.5 Plan for Exploitation and Dissemination 54](#_Toc491164372)

[7.6 Future plans 55](#_Toc491164373)

[Appendix I. ARGO development process 56](#_Toc491164374)

[Appendix II. GOCDB development process 64](#_Toc491164375)

[Appendix III. Accounting Repository dev process 65](#_Toc491164376)

**Executive summary**

This deliverable describes the final release of the EGI Accounting and Operational Tools during EGI-Engage project, including the developments made during the third year of the project for the Operations Portal, ARGO, Messaging, GOCDB, Security Monitoring, Accounting Repository and Portal. The evolution of these tools has been driven by the need to support new technologies (e.g. cloud) and to satisfy new requirements emerging from service providers and user communities, in particular from the Research Infrastructures contributing to EGI-Engage via the EGI Competence Centres (CCs) and the Resource Providers (RPs) who contribute infrastructure services to the federation. The development roadmap has been reviewed and updated according to a requirement gathering process, which has been accomplished in collaboration with the other EGI Engage WPs in charge of the communication with users and key stakeholders.

The Operations Portal team implemented a new metric view for the VOs merging data collected from both the Accounting system and the AppDB. The tool now supports the new EGI AAI based on the CheckIn service[[1]](#footnote-1) and improvements were applied to the VO ID Card. The VAPOR module provides now a summary of the CPU and storage capacities by countries, resource or operations centres, and geographical maps with a global view of all the resource providers with a VO filter.

Several new features are now available in the ARGO Monitoring system. The Compute Engine was enhanced to support stream processing in real time. The introduction of this new feature enables the development of new functionalities that go beyond the infrastructure monitoring, as for example an alerting system. The support for probe management in the POEM component greatly simplifies the addition of new probes in the system. In addition, ARGO now only uses GOCDB as a source of topology information, new probes were developed and the UI was enhanced with new reports and updates on existing views.

Final tests to move the new Messaging Service into production are running. It provides an HTTP API that enables users/systems to implement a service-oriented messaging system using the Publish/Subscribe Model over plain HTTP. Work to migrate the ARGO monitoring system, the Operations Portal and the accounting system to the new Messaging Service is progressing well and will be completed shortly.

To meet requirements of communities, including WLCG, the write API of the GOCDB was extended to allow the programmatic creation, update, and deletion of service endpoints and updates to the details of services. This update allows changes to key entities within GOCDB, programmatically, and represents a significant evolution in the way in which GOCDB works, allowing for much greater automated interaction with the information managed by GOCDB.

Work on Security Monitoring is progressing and Secant, the framework to detect security vulnerabilities in images of virtual machines, will be integrated with AppDB in the coming months to support the assessment of the virtual appliances during the endorsement process.

Finally, the accounting system has added storage systems as a source of accounting data and support for long running virtual machines was included in cloud accounting. The Accounting Portal has been enhanced with the introduction of new views and metrics.

# Operations Portal

## Introduction

|  |  |
| --- | --- |
| **Tool name** | Operations Portal |
| **Tool url** | <http://operations-portal.egi.eu> |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/Operations_Portal> |
| **Description** | The Operations Portal provides VO management functions and other capabilities, which support the EGI daily operations. It is a central portal for the operations community that offers a bundle of different capabilities, such as the broadcast tool, VO management facilities, a security dashboard and an operations dashboard that is used to display information about failing monitoring probes and to open tickets to the affected Resource Centres. The dashboard also supports the central grid oversight activities. It is fully interfaced with the EGI Helpdesk and the monitoring system through messaging. It is a critical component as it is used by all EGI Operations Centres to provide support to the respective Resource Centres. The Operations Portal provides tools supporting the daily running of operations of the entire infrastructure: grid oversight, security operations, VO management, broadcast, VO metrics.VAPOR: the Vo Administration and Operations Portal, is a generic tool to assist community managers and support teams in performing their daily activities. The application provides resources status indicators, statistical reports, data management tools. It gathers resource information from the BDII and displays them in an ordered way, replacing the features previously offered by GSTAT. The amount of resources and the resources themselves are shown in different views that group information per Operations Centres, Countries and VOs. |
| **Value proposition** | New features offered by the Operations Portals allow its customers to better monitor and browse the infrastructure and, then, adapting their workflows according to the exact status of the computing and storage resources (e.g. moving some computation from one provider to another since the latter is working better). |
| **Customer of the tool** | EGI; NGI; RI; Resource Provider; Research Communities |
| **User of the service** | Site admins; Operations Managers; VO Manager; VO users;  |
| **User Documentation**  | <https://forge.in2p3.fr/projects/opsportaluser/wiki/Main_Features_of_the_dashboard><http://operations-portal.egi.eu/vapor/globalHelp> |
| **Technical Documentation**  | <https://forge.in2p3.fr/projects/opsportaluser/wiki/Main_Features_of_the_dashboard>  |
| **Product team** | IN2P3/CNRS |
| **License** | Apache 2.0 |
| **Source code** | <https://gitlab.in2p3.fr/groups/opsportal>  |

## Service architecture

### High-Level Service architecture

The Operations Portal has been designed as an integration platform, allowing for strong interaction among existing tools with similar scope but also filling up gaps wherever functionality has been lacking. The displayed information is retrieved from several distributed static and dynamic sources – databases, Grid Information System, Web Services, etc. – and gathered within the portal.

The architecture of the portal is composed of three modules:

* A database – to store information related to the users or the VO;
* A web module – graphical user interface – which is currently integrated into the Symfony framework;
* A Data Aggregation and Unification Service named Lavoisier.

Lavoisier is the component used to store, consolidate and “feed” data into the web application.

The global information from the primary and heterogeneous data sources (e.g. BDII, GOCDB, NAGIOS, GGUS, ARGO, etc.) is retrieved by means of the use of the different plug-ins. The collected information is structured and organized within configuration files in Lavoisier and, finally, made available to the web application without the need for any further computations. This modular architecture is conceived to add easily new data source in this model and use the cached information if a primary source is unavailable. The data sources are refreshed only as needed and only when an action has been triggered. In addition, it is very easy to add a new data source in this model, as depicted in Fig. 1 and Fig. 2. Nevertheless, two critical dependencies are remaining: GGUS[[2]](#footnote-2) and RTIR[[3]](#footnote-3) (red arrows on the left on next figure).

These dependencies are due to the communication via web services between the Operations Portal and GGUS/RTIR for the creation or the update of tickets.

In case of disruptions of the GGUS or RT services, a part of the features of the Operations Portal will be affected: the creation and the update of tickets into the dashboards. For the rest of data sources, the cache mechanism of Lavoisier permits us to ensure the integrity of the application in case of failures of third parties providers.



Figure 1. Operations Portal architecture

For the VAPOR application, we use the same architecture with a dedicated instance of Lavoisier. Information is aggregated from several top BDII objects and from a monitoring tool based on Jsaga (JobMonitor) and local scripts in Python and shell developed specifically to ease the VO support.

VAPOR is fully integrated in the Operations Portal and is presented to the users as an additional feature available.



Figure 2. VAPOR architecture

### Integration and dependencies

Operations Portal dependencies have been already described in the previous section. They are not changed in this release.

## Release notes

### Operations Portal 4.0

This version is a major evolution of the background technologies of the portal.

The aim was to upgrade the different technologies used around the portal and ensure a better maintainability and an improvement of the performances. Here are the main changes for this version:

1. Frameworks & JS Libraries
* Migration to Symfony 3;
* Upgrade of bootstrap library;
* Adoption of the Datatables Js libraries to optimize the presentation of the tables (VO Management, Metrics);
* Use of Google Chart (VO Management, Metrics).
1. Ergonomics
* Addition of links to ARGO and VAPOR applications;
* Changes into global menu presentation (and optimization depending on screen size).
1. Module and project modifications
* Reorganisation of the project infrastructure;
* Removal of obsolete files and features;
* Merge of the VO Management Tool and VO ID cards (all-in-one page);
* Removal of Availabilities/reliabilities module (replaced by ARGO).
1. Downtime Module (new module)

The historical downtime subscription system has been removed and replaced within a dedicated module offering the following features:

* A subscription page (emails, RSS, ical);
* Timelines charts and tables;
* Search tool;
* Data exportable in different formats (CSV, JSON).
1. Continuous Integration
* A procedure about good practices for the development procedure is in place: <https://forge.in2p3.fr/projects/opsportaluser/wiki/Development_Procedure>
* An integration platform has been set-up with PHPUnit , GitlabCI , Docker and SonarQBE: <https://forge.in2p3.fr/projects/opsportaluser/wiki/Continuous_Integration>

### Operations Portal 4.1

This version was focused on:

* Several improvements on the VO ID cards;
* Improvement of the documentation of the main features;
* The fixes of different bugs due to the important changes of the previous version.

### Operations Portal 4.2

This version is foreseen for August and is focused on:

* Integration of complementary metrics for the VO: accounting data and AppDB changes;
* Improvements on the VO ID Card;
* The support of the new EGI AAI based on the CheckIn service (IdP/SP Proxy).
* A backend for the monitoring
	+ Exploration of logs (Apache , symfony, access)
	+ Status of the Lavoisier servers and views
	+ Status of some tables of the DB
	+ The use of ARGO messaging system to collect Nagios notifications

### VAPOR 2.0

The initial prototype (described in D3.4[[4]](#footnote-4)) has been put in production after a test phase of one month.

### VAPOR 2.1

The main features of this release were:

* Integration of GSTAT features;
	+ a map of the resources:

<http://operations-portal.egi.eu/vapor/resources/GL2Map>

* + a table of the resources:

<http://operations-portal.egi.eu/vapor/resources/GL2ResSummary>

* + a Top BDII browser:

<http://operations-portal.egi.eu/vapor/resources/GL2ResBdiiBrowser>

* New menu;
* Bug fixing;
* Integration of feedback given by users;
* Ergonomics improvements.

### VAPOR 2.2

This release has been delivered in February 2017.

For this release, the Operations Portal team has worked closely with the EGI Operations to consolidate the different queries to the Top BDII and the different extracted figures. The results are the following:

* A summary of the CPU and storage capacities by countries, sites or Operations Centres;
* A geographical map with the distribution of sites with a VO filter;
* Some additions in the faulty publications: bad HEPSPEC, mismatches between the different benchmarks, negative values for jobs.

This release has been also focused on the documentation of the different features and the access to the API.

### VAPOR 2.3

This release is currently in the test phase and will be delivered in August 2017.

Once again this release is the results of multiple exchanges with EGI Operations team to enhance the current features. We have worked on different improvements:

* Upgrade of the different JavaScript libraries to improve the performances.
* Identify the duplicated values published by the sites.
* A map has been added with a global view of all the sites.
* A summary of the figures is now available for each site.
* The global storage capacity computation has been improved.
* One new metric has been added in agreement with EGI Operations team : “the Computation power”

## Feedback on satisfaction

Prioritization and testing has been done by dedicated Operations Portal Advisory and Testing Board (OPAnTG)[[5]](#footnote-5) coordinated by EGI Operations team. Furthermore, the Operations Portal team has worked on the automation of tests. Unit and acceptance tests are now done through Docker piloted by GitLab Continuous Integration server.

If tests are failing, new features are not propagated to the test infrastructure. This allows performing a first bug filter before manually tests are executed. Complementary to these tests, the team also adopted a SonarQBE instance to inspect the quality of code.

The architecture of the Operations Portal automatic test suite is described below.

As a result, a minor number of bugs have been identified by the testing team in the most recent releases.



Figure 3. Operations Portal - Automatic test suite.

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| *Name of the result* | Operation Portal |
| *DEFINITION*  |
| *Category of result* | Software & service innovation |
| *Description of the result* | Software enhancement: integrate the VO Administration and operations PORtal (VAPOR) into the Operations Portal and enhance the monitor infrastructure resources including the most relevant features currently offered by GSTAT.  |
| *EXPLOITATION* |
| *Target group(s)* | Users, NGIs, Resource centres, RIs |
| *Needs* | Monitor / browse / Evaluate the resources for VO, sites, Operations Centres |
| *How the target groups will use the result?* | * Exploit the new features in the daily operations of the EGI infrastructure
* Exploit the advanced metrics to better promote the EGI infrastructure
 |
| *Benefits* | * Ease the daily administration of the resources
* Have an overview of the resources and their status
* Be more efficient in the daily job submission
 |
| *How will you protect the results?* | Apache 2 License |
| *Actions for exploitation* | The result is accessible through the web site and the code is hosted on a GitLab. |
| *URL to project result* | <http://operations-portal/vapor><https://gitlab.in2p3.fr/opsportal/>  |
| *Success criteria* | The deployment in production and the use by end users. |
| *DISSEMINATION* |
| *Key messages* | Browse and evaluate your resources |
| *Channels* | EGI Broadcast tool, EGI Meetings |
| *Actions for dissemination* | EGI conferences, publications, participation to workshops organised by potential users. |
| *Cost* |  |
| *Evaluation* | The number of requests and the feedback given by users |

## Future plans

Future plans cover following aspects:

* VAPOR
	+ Enhance the historical scripts, especially the ‘JobMonitor’ Tool;
	+ Consolidation / coherency of the data:
		- Data issued from site publications with incoherencies:
		- Detect and propose corrections:
	+ Extend the current features with user feedback.
* Operations Portal
	+ integration of complementary metrics for the VO;
	+ Add more genericity in the VO Id cards;
	+ Extend the current features with user feedback;
	+ Adapt the current tools to the new communities;
	+ Define a new module for the SLA/OLA management including:
		- workflows to automatic service activations;
		- on-demand generation of reports on resource usage.

# ARGO

## Introduction

|  |  |
| --- | --- |
| **Tool name** | ARGO |
| **Tool url** | <http://argo.egi.eu>  |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/ARGO>  |
| **Description** | ARGO is a flexible and scalable framework for monitoring status, availability and reliability |
| **Value proposition** | Improved portal design that allows new and easier way to access and visualise data for the final users. Third parties can now gather monitoring data from the system through a complete API. A central deployment of the ARGO monitoring engine can serve a large infrastructure reducing the maintenance costs. |
| **Customer of the tool** | EGI; NGI; RI; Resource Provider; Research Communities |
| **User of the service** | Site admins; Operations Managers; large research group |
| **User Documentation**  | <http://argoeu.github.io>;<http://argo.egi.eu> |
| **Technical Documentation**  | <http://argoeu.github.io> |
| **Product team** | GRNET, SRCE, CNRS |
| **License** | Apache License Version 2.0 |
| **Source code** | <https://github.com/ARGOeu/>  |

## Service architecture

### High-Level Service architecture

ARGO is a flexible and scalable framework for monitoring status, availability and reliability of services provided by infrastructures with medium to high complexity. It can generate multiple reports using customer defined profiles (e.g. for SLA management, operations, etc.) and has built-in multi-tenant support in the core framework.

ARGO supports flexible deployment models and its modular design enables ARGO to be integrated with external systems (such as CMDBs, Service Catalogues, etc.). During the report generation, ARGO can take into account custom factors such as the importance of a specific service endpoint, scheduled or unscheduled downtimes, etc.



Figure 4. ARGO architecture.

For the Availability & Reliability monitoring, ARGO relies on a modular architecture comprised of the components described in the following paragraphs.

#### The ARGO Monitoring Engine

For status monitoring, ARGO relies on Nagios. All probes developed for ARGO follow the Nagios conventions and can run on any stock Nagios box. ARGO provides an optional set of add-ons for the stock Nagios that provide features such as auto-configuration from external information sources, publishing results to external Message Brokers, etc.

In order to use the new messaging service, the monitoring engine also supports the new AMS Publisher. The AMS publisher is a new component acting as bridge from Nagios to ARGO Messaging system. It is integral part of software stack running on ARGO monitoring instance and is responsible for forming and dispatching messages that are results of Nagios tests. It is ready and running on the development infrastructure. It is running as a Unix daemon and it consists of two subsystems:

* Queueing mechanism;
* Publishing/dispatching part.

Messages are cached in a local queue with the help of OCSP Nagios calls and each queue is being monitored by the daemon. After configurable amount of accumulated messages, the publisher that is associated to queue sends them to ARGO Messaging system and drains the queue. argo-nagios-ams-publisher is written in multiprocessing manner enabling the support for multiple queue/publish pairs where, for each of them, new worker process will be spawned.

#### The ARGO Connectors

Through the use of custom connectors, ARGO can connect to multiple external Configuration Management Databases and Service Catalogues. Connectors for the EGI and EUDAT e-Infrastructures are already available.

#### The ARGO Consumer

The ARGO Consumer is ingesting monitoring results in real-time from external Message Brokers. The consumer is responsible for the initial pre-filtering of the monitoring results and encodes them using AVRO serialization format[[6]](#footnote-6) before passing to the Compute Engine.

#### The ARGO Compute Engine

A powerful and scalable analytics engine built on top of Hadoop and HDFS[[7]](#footnote-7). The Compute Engine is responsible for the aggregation of the status results and the computation of availability and reliability of composite services using customer defined algorithms. The reorganization of the Compute Engine to support stream processing in real time is one of the key new factors. A new streaming layer has been introduced. Monitoring results flow through the AMS, to the streaming layer (in parallel to the HDFS). The streaming layer is used in order to push raw metric results to the metric result store and to compute status results and push them to the status store in real-time.

#### The ARGO Web API

The ARGO Web API provides the serving layer of ARGO. It is comprised of a high performance and scalable data store and a multi-tenant REST HTTP API, which is used for retrieving the Status, Availability and Reliability reports and the actual raw metric results.

#### The ARGO Web UI

The default web UI is based on the Lavoisier Data Aggregation Framework[[8]](#footnote-8).

### Integration and dependencies

ARGO can utilize external configuration sources through connectors in order to allow the automatic configuration of various ARGO components. The current version of ARGO includes connectors for the following sources:

* GOCDB: It is used as the source of EGI infrastructure topology information and information about declared downtimes.
* VAPOR: It is used as the source for custom factor values, which in the case of EGI it is the HEPSPEC[[9]](#footnote-9) values of the sites.

The dependency to these external tools is optional. ARGO can be used without having any of these connectors enabled, if there is at least a static configuration for the topology of the monitored infrastructure.

Finally, ARGO relies on the Message Broker network as the transport layer for publishing monitoring results from the Nagios Monitoring Engines to the ARGO Compute Engine.

## Release notes

### Requirements covered in the release

As already mentioned ARGO is not just single software, but a suite of software components, each one managed independently. During the third year of the project, there have been a number of releases of the ARGO components that covered the following requirements:

**ARGO Compute Engine & Web API**

* Streaming processing;
* Alerting mechanism;
* Separation of A/R and Metric stores:
* APIv2;
* Stability and performance improvements.

**ARGO Monitoring Engine**

* Migration of ops probes from opsmon.egi.eu to the central monitoring instances argo-mon/2.egi.eu and decommissioning of opsmon.egi.eu;
* Deployment of three new ARGO Monitoring Services:
	+ Testing instance (argo-mon-test) used for testing new ARGO Monitoring Service releases and deployment of new probes and updates of existing probes; instance is constantly monitoring subset of EGI infrastructure and list of sites and service endpoints is extended on demand;
	+ Uncertified instance (argo-mon-uncert) used for monitoring uncertified sites which fully relies on information provided by sites in GOCDB;
	+ Internal instance used for monitoring all internal ARGO components by using ARGO probes and NRPE;
* New probes and updates of existing probes:
	+ New probe for decommissioning of dCache 2.10 and dCache 2.13;
	+ New probes for OneData services;
	+ New probes for AAI CheckIn service;
	+ New probe for NGI Argus service;
	+ New probe for WebDAV service;
	+ Improved probes for FTS3, gsisshd and VOMS services;
	+ Improved probes for CREAM-CE;
	+ Analysis and deployment of new ARC-CE probes;
	+ Scripts provided for handling UNICORE probes configuration;
* Prototype version of ARGO Monitoring Service for biomed VO;
* AMS Publisher: is a new component acting as bridge from Nagios to ARGO Messaging system. It is integral part of software stack running on ARGO monitoring instance and is responsible for forming and dispatching messages that are results of Nagios tests. Successfully running on the development infrastructure for more than a month;
* Support for GOCDB as a single source of topology:
	+ Step 1: Randomly check service endpoint;
* Stability and performance improvements.

**ARGO EGI Consumer and Connectors**

* Use of ARGO nagios AMS-publisher;
	+ Ready on development infrastructure;
* Use of the messaging API for Connectors component;
	+ Ready on development infrastructure;
* Stability and performance improvements.

**ARGO EGI Web UI**

* New Uncertified report;
* New FedCloud Report;
* UI Enhancements;
	+ New pdf report;
	+ Updates to ELIXIR report;
	+ Updates to admin list;
	+ Updates to links to reports.

**ARGO POEM**

* Finalize support for probe management;
* Initial steps for the connection to the EGI IdP/SP Proxy;
* Stability and performance improvements.

#### Changelog

* **25/06/2017**
	+ ARGO Monitoring Plugin - AMS publisher [Version 0.2.0-1] <https://github.com/ARGOeu/argo-nagios-ams-publisher/releases/tag/v0.2.0-1>
	+ ARGO-Monitoring Engine [Version 0.4.0-1] <https://github.com/ARGOeu/argo-ncg/releases/tag/v0.4.0-1>
* **20/06/2017**
	+ ARGO-Poem [Version 1.0.5-1] <https://github.com/ARGOeu/poem/releases/tag/v1.0.5-1>
* **24/05/2017**
	+ ARGO-Connectros [Version 1.5.9-1] <https://github.com/ARGOeu/argo-egi-connectors/releases/tag/v1.5.9>
	+ ARGO-Monitoring Engine [Version 0.3.4-1] <https://github.com/ARGOeu/argo-ncg/releases/tag/0.3.4-1>
* **06/05/2017**
	+ ARGO-Poem [Version 1.0.4-1] <https://github.com/ARGOeu/poem/releases/tag/v1.0.4-1>
	+ ARGO Web UI [Version 1.3.6-2] <https://github.com/ARGOeu/argo-egi-web/releases/tag/V1.3.6-2>
* **04/05/2017**
	+ ARGO-Monitoring Engine [Version 0.3.3-1] <https://github.com/ARGOeu/argo-ncg/releases/tag/0.3.3-1>
* **03/04/2017**
	+ ARGO-Monitoring Engine [Version 0.3.2-1] <https://github.com/ARGOeu/argo-ncg/releases/tag/0.3.2-1>
* **03/04/2017**
	+ ARGO-Connectros [Version 1.5.8-1] <https://github.com/ARGOeu/argo-egi-connectors/releases/tag/v1.5.8>
* **17/03/2017**
	+ ARGO-Connectros [Version 1.5.6-1] <https://github.com/ARGOeu/argo-egi-connectors/releases/tag/v1.5.6>
* **06/03/2017**
	+ ARGO-Connectros [Version 1.5.4-1] <https://github.com/ARGOeu/argo-egi-connectors/releases/tag/V1.5.4-1>
* **03/03/2017**
	+ ARGO-Monitoring Engine [Version 0.3.1-1] <https://github.com/ARGOeu/argo-ncg/releases/tag/0.3.1-1>
* **16/02/2017**
	+ ARGO-Monitoring Engine [Version 0.3.0-1] <https://github.com/ARGOeu/argo-ncg/releases/tag/0.3.0-1>
* **09/02/2017**
	+ ARGO Web UI [Version 1.3.6-1] <https://github.com/ARGOeu/argo-egi-web/releases/tag/v1.3.6-1>
* **30/01/2017**
	+ ARGO Web UI [Version 1.3.5-1] <https://github.com/ARGOeu/argo-egi-web/releases/tag/v1.3.5-1>
* **17/01/2017**
	+ ARGO Compute Engine [Version 1.6.9-1] <https://github.com/ARGOeu/argo-compute-engine/releases/tag/v1.6.9-1>
* **10/01/2017**
	+ ARGO-Poem [Version 1.0.3-1] <https://github.com/ARGOeu/poem/releases/tag/v1.0.3-1>

## Feedback on satisfaction

The ARGO product team uses a development process based around GitHub, which includes procedures that guarantee a high quality of software releases. For details of the ARGO development process, see Appendix I.

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| *Name of the result* | ARGO |
| *DEFINITION*  |
| *Category of result* | Software & service innovation |
| *Description of the result* | Software enhancement: improve the portal designing new and easier way to access and visualise data for the final users and exposing a complete API allowing third parties to gather accounting data from the system.Stability and performance improvements of the central ARGO Monitoring Service. NGI instances were decommissioned or kept for NGI’s internal purposes. In addition, specific monitoring instances like opsmon.egi.eu were decommissioned and all probes were integrated into central ARGO Monitoring Service. A/R calculations are performed solely by using results from the central ARGO Monitoring Service. Uncertified instances are also monitored via the centralized ARGO Monitoring Service. Two additional centralized ARGO Monitoring Services were deployed for testing and verification of new probes and for monitoring internal ARGO components.Centralized ARGO Monitoring Service poses a risk if only one instance is deployed. In case of failure of that instance, the whole infrastructure will not be monitored. Therefore, a high availability setup is used.The reorganization of the Compute Engine to support stream processing in real time is one of the key new factors. A new streaming layer has been introduced. Monitoring results flow through the AMS to the streaming layer (in parallel to the HDFS). The streaming layer is used in order to push raw metric results to the metric result store and to compute status results and push them to the status store in real-time. The streaming and batch job for the status results is running in the development infrastructure producing the same results as the production infrastructure. At the same time, the new AMS publisher has been introduced. It is a new component acting as bridge from Nagios to the new ARGO Messaging system. It is successfully running on the development infrastructure for a while producing the same results as on production. It is integral part of the software stack running on ARGO monitoring instance and is responsible for forming and dispatching messages that are results of Nagios tests. Thanks to the new real-time Streaming processing layer, we are now able to introduce new functionality to the ARGO Monitoring Service that goes beyond infrastructure monitoring, as, for example, the alerting. We are working on a new component on top of the streaming engine. This component will analyse the monitoring results and send notification based on a set of rules. The minimum set of rules support should mimic the Nagios behaviour. |
| *EXPLOITATION* |
| *Target group(s)* | RIs, Service providers, Users, NGIs, Resource centres |
| *Needs* | * Used for the Availability and Reliability monitoring
* Provide complete API allowing third parties to gather data from the system.
* Used as a source of alerts for resource centres administrators through the Operations Portal Dashboard
* Used for middleware versions monitoring and upgrade campaigns
 |
| *How the target groups will use the result?* | The ARGO Availability and Reliability Monitoring Framework is used by the ARGO Monitoring Service that is operated by EGI for the monitoring of the availability and reliability of the EGI infrastructure. The ARGO Monitoring Service can be provided also to research communities and other infrastructures as a service in order to monitor the status, availability and reliability of their services. |
| *Benefits* | The developments during this period, allowed EGI to replace the older implementation of the SAM Nagios Monitoring Engine, which required one monitoring engine per NGI, with a new implementation using the ARGO Monitoring Engine, which provided a monitoring engine that could deliver monitoring probe scheduling and execution as a service for all the NGI and communities. Central ARGO requires less maintenance effort and enables faster and streamlined deployment of new tests or update of existing tests. This leads to improvements in the performance, robustness and reliability of the ARGO Monitoring Service.Furthermore, real-time computations give the ability to take immediate action for urgent issues. The goal is to obtain the insight required to act prudently at the right time - which increasingly means immediately. |
| *How will you protect the results?* | The ARGO Monitoring Framework is released under the Apache 2.0 license. |
| *Actions for exploitation* | The new version of the ARGO Monitoring Framework has already been adopted by the production ARGO Monitoring Service. In order to further exploit the results, we should promote the service also to research communities and other infrastructures that can benefit of its features. |
| *URL to project result* | <http://argo.egi.eu/> <https://github.com/ARGOeu/>  |
| *Success criteria* | The deployment of the results to the production EGI infrastructure. The usage of the service to monitor third party services. |
| *DISSEMINATION* |
| *Key messages* | Offer a guaranteed quality of services. |
| *Channels* | EGI Broadcast tool, EGI Meetings. |
| *Actions for dissemination* | EGI conferences, publications, participation to workshops organised by potential users |
| *Cost* |  |
| *Evaluation* | The number of requests for information is the main way to evaluate the impact of the dissemination actions. |

## Future plans

Future plans cover following aspects:

**ARGO Compute Engine**

* Streaming processing;
* Alerting mechanism;
* Separation of A/R and Metric stores;
* Stability and performance improvements.

**ARGO Monitoring Engine**

* Finalize support for GOCDB as a single support of topology;
* Integration with probe management feature in POEM;
* Use of the messaging API on production;
* Fedcloud probes updates;
* Stability and performance improvements.

**ARGO Web UI**

* UI Enhancements.

**ARGO EGI Consumers and Connectors**

* Decommission of Consumer and use ARGO nagios AMS-publisher instead;
* Finalize the use of the messaging API for Connectors component on production;
* Stability and performance improvements.

**ARGO POEM**

* Finalize the probe management feature;
* Connect to the EGI IdP/SP Proxy;
* Stability and performance improvements.

# Messaging Service

## Introduction

|  |  |
| --- | --- |
| **Tool name** | ARGO Messaging Service |
| **Tool url** | <http://argoeu.github.io> |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/Message_brokers>  |
| **Description**  | The Messaging service enables reliable asynchronous messaging for the EGI infrastructure.  |
| **Value proposition** | e-Infrastructures and research communities are building distributed services and workflows in order to satisfy their operational and research requirements. Synchronization between services, gathering of telemetry, monitoring and accounting data, and any secure messages exchange are core requirements in any type of distributed services. The Messaging Service provides an easy to use and reliable transport layer for the secure exchange of messages between services such as accounting data, monitoring data, event notifications, etc. |
| **Customer of the tool** | EGI; NGI; RI; Resource Provider; Research Communities |
| **User of the service** | Site admins; Operations Managers; Large research group |
| **User Documentation**  | <http://argoeu.github.io>; |
| **Technical Documentation**  | <http://argoeu.github.io> |
| **Product team** | GRNET, SRCE |
| **License** | Apache License Version 2.0 |
| **Source code** | <https://github.com/ARGOeu/>  |

## Service architecture

### High-Level Service architecture

The Messaging service enables reliable asynchronous messaging for the EGI infrastructure. The current implementation of the Messaging service relies on a Message Broker Network of ActiveMQ services and uses the STOMP protocol for the publication and consumption of messages.



Figure 5. Messaging service architecture.

During the project, we have developed a new version of the Messaging service that is going to replace the STOMP interface with an HTTP one, which will make the implementation of new clients easier and more robust. The new ARGO Messaging Service is a real-time messaging service that allows you to send and receive messages between independent applications.

The ARGO Messaging Service is a Publish/Subscribe Service, which implements the Google PubSub protocol. It provides an HTTP API that enables users/systems to implement a message-oriented service using the Publish/Subscribe Model over plain HTTP. Publishers are users/systems that can send messages to named-channels called Topics. Subscribers are users/systems that create Subscriptions to specific topics and receive messages.

It supports both push and pull message delivery. In push delivery, the Messaging Service initiates requests to your subscriber application to deliver messages. In pull delivery, your subscription application initiates requests to the server to retrieve messages.



Figure 6. The new ARGO messaging service.

#### AMS Metrics

The AMS Pub/Sub API exports usage metrics that can be monitored programmatically. The list of available metrics is the following:

* Memory usage per AMS instance: percentage value that displays the Memory usage of AMS service in the specific node;
* CPU usage per AMS instance: percentage value that displays the CPU usage of AMS service in the specific node;
* Messages published per topic/project/user:
	+ Counter that displays the number of messages published to the specific topic (per project and per user);
* Messages delivered per topic/subscription/project/user:
	+ Counter that displays the number of messages delivered to the specific subscription (per project, per user and per topic);
* Bytes in/out per topic/subscription/project/user:
	+ Counter that displays the total size of data (in bytes) published to the specific topic;
	+ Counter that displays the total size of data (in bytes) consumed from the specific subscription;
* Topics per project/user:
	+ Counter that displays the number of topics belonging to the specific project;
	+ Counter that displays the number of topics belonging to the specific user;
* Subscriptions per project/topic/user:
	+ Counter that displays the number of subscriptions belonging to the specific project:
	+ Counter that displays the number of subscriptions that a user has access to the specific project:
	+ Counter that displays the number of subscriptions belonging to the specific topic;
	+ Counter that displays the number of subscriptions belonging to the specific user.

#### Operational Metrics[[10]](#footnote-10)

The Operational Metrics mainly include metrics related to the CPU or memory usage of the AMS nodes. The list of operational metrics is the following:

* Memory usage per AMS instance;
* CPU usage per AMS instance;
* Messages published per topic/project/user;
* Messages delivered per topic/subscription/project/user;
* Bytes in/out per topic/subscription/project/user;
* Topics per project/user;
* Subscriptions per project/topic/user.

#### Accounting

The list of accounting metrics is the following:

* Number of messages published per topic/project/user;
* Number of messages delivered per topic/subscription/project/user;
* Number of topics per project/user;
* Number of subscriptions per project/topic/user.

#### AMS - Library[[11]](#footnote-11)

It is a simple Python library for interacting with the ARGO Messaging Service.

The Messaging Service is implemented as a Publish/Subscribe service. Instead of focusing on a single Messaging API specification for handling the logic of publishing/subscribing to the broker network, the API focuses on creating nodes of Publishers and Subscribers as a Service.

In the Publish/Subscribe paradigm, Publishers are users/systems that can send messages to named-channels called Topics. Subscribers are users/systems that create Subscriptions to specific topics and receive messages.

You may find more information in the ARGO Messaging Service documentation[[12]](#footnote-12).

### Integration and dependencies

The following EGI Core Services rely on the EGI Messaging Service:

* ARGO Availability and Reliability Monitoring Service
* Accounting system
* Operations Portal

All these services are using the EGI Message Broker network today. The ARGO Monitoring Service is already implementing a connector for the new Messaging Service. Accounting and Operations portal are expected to also complete the implementation of their own interfaces to the new Messaging Service, within the timeframe of the EGI-Engage project.

The Messaging Service does not have any dependencies to other services now.

## Release notes

### Requirements covered in the release

* APIv1 test implementation;
* APIv1 final implementation;
* APIv1 final specification;
* Support APEL to use Messaging Service;
* Support AppDB to use Messaging Service;
* Support Operational Portal to use Messaging Service;
* Message Service Accounting: Metrics for Messaging Service;
* Operational statistics;
* Usage Statistics;
* Stability and performance improvements.

###  Changelog

* **28/06/2017**
* **AMS Library [Version 0.3.0-1]** <https://github.com/ARGOeu/argo-ams-library/releases/tag/v0.3.0-1>
* **08/06/2017**
* **AMS Library [Version 0.2.0-1]** <https://github.com/ARGOeu/argo-ams-library/releases/tag/v0.2.0-1>
* **25/10/2016**
* **ARGO - Messaging Service [v1.0.0-1]** [**https://github.com/ARGOeu/argo-messaging/releases/tag/v1.0.0-1**](https://github.com/ARGOeu/argo-messaging/releases/tag/v1.0.0-1)

## Feedback on satisfaction

The ARGO product team uses a development process based around GitHub, which includes procedures that guarantee a high quality of software releases. For details of the ARGO development process, see Appendix I.

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| *Name of the result* | ARGO Messaging Service |
| DEFINITION  |
| *Category of result* | Software & service innovation |
| *Description of the result* | In the new version of the Messaging Service, the STOMP interface has been replaced with an HTTP interface, which makes the implementation of new clients easier and the implementation more robust. This new ARGO Messaging Service is a real-time messaging service that allows services to asynchronously send and receive messages using the Publish/Subscribe model. |
| EXPLOITATION |
| *Target group(s)* | RIs, Service providers, Users, NGIs, Resource centres, EGI Accounting service and the Operations Portal |
| *Needs* | e-Infrastructures and research communities are building distributed services and workflows in order to satisfy their operational and research requirements. Synchronization between services, gathering of telemetry, monitoring and accounting data, and any secure messages exchange are core requirements in any type of distributed services. The Messaging Service provides an easy to use and reliable transport layer for the secure exchange of messages between services such as accounting data, monitoring data, event notifications, etc.  |
| *How the target groups will use the result?* | Infrastructure architects that need to design distributed architectures that require a robust and easy to use messaging backbone, which can scale to billions of messages. |
| *Benefits* | The ARGO Messaging service offers the following features: * Simple HTTP API for client access;
* An easy to use python library;
* Operations & usage metrics ;
* Transparent scalability & high availability;
* Access controls implemented at the API layer;
* Multi-tenant support;
* Performance robustness.
 |
| *How will you protect the results?* | The ARGO Messaging service is released under the Apache 2.0 license. |
| *Actions for exploitation* | * Promote the service to other research communities and infrastructures that can benefit of its features.
* Provide the necessary documentation (all, for a publisher, or for a subscriber)
* Create test accounts per target group to publish messages to topics, or to consume messages as subscribers from a topic.
 |
| *URL to project result* | <http://argo.egi.eu/> <https://github.com/ARGOeu/>  |
| *Success criteria* | * The ARGO Messaging Service should be operated as a production EGI service.
* All the EGI tools services should have migrated from the old Messaging Broker service to the new ARGO Messaging service.
 |
| DISSEMINATION |
| *Key messages* | Interconnect your distributed services in a ease and efficient manner. |
| *Channels* | * Dissemination through the EGI conferences
* Article featured in the EGI newsletter
 |
| *Actions for dissemination* | EGI conferences, publications, participation to workshops organised by potential users |
| *Cost* |  |
| *Evaluation* | The number of requests for information, and/or accounts (either test or production) is the main way to evaluate the impact of the dissemination actions. |

## Future plans

Future plans cover following aspects:

* Move to production
* Stability and performance improvements

# GOCDB

## Introduction

|  |  |
| --- | --- |
| **Tool name** | GOCDB |
| **Tool url** | <https://goc.egi.eu>  |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/GOCDB> |
| **Description** | GOCDB is a central registry to record information about the topology of an e-Infrastructure. This includes entities such as resource centers (sites), services, service-endpoints and their downtimes, contact information and roles of users responsible for operations at different levels. The service enforces a number of business rules and defines different grouping mechanisms including object-tagging for the purposes of fine-grained resource filtering. |
| **Value proposition** | The Extensions to the write API will greatly increase the ability for external tools to interact in a programmatic way with GOCDB. This will make GOCDB more viable for the future and reduce the need for other information systems.  |
| **Customer of the tool** | EGI Operations and WLCG  |
| **User of the service** | Site/service admins, NGI managers and Security teams. |
| **User Documentation**  | <https://wiki.egi.eu/wiki/GOCDB> |
| **Technical Documentation**  | <https://wiki.egi.eu/wiki/GOCDB> |
| **Product team** | STFC |
| **License** | Apache 2  |
| **Source code** | <https://github.com/GOCDB/gocdb> |

## Service architecture

### High-Level Service architecture

GOCDB is a central information repository providing a web portal interface for CRUD operations, and a REST API for data queries.

It is a definitive information source where data is directly populated and managed in the system. Because GOCDB is a primary data-input source, the portal applies a range of business rules and data-validations to control input. It applies a comprehensive Role-based authorization model that enables different actions over different target resources. The Role model allows communities to manage their own resources where users with existing roles can approve or reject new role-requests.

It is intentionally designed to have no dependencies on other operational tools (other than the EGI CheckIn service described below). For example, it does not query other systems to populate its core data model. The underling Oracle DB is hosted by the STFC DB Services Team with nightly tape backups. An additional failover instance is hosted at a second STFC site (Daresbury Labs). The failover instance is synchronized hourly against the production data.

The previous release, introduced a new dependency on the EGI CheckIn service in order to provide federated access to GOCDB for users without client certificates. However, for users with certificates there continues to be no dependencies on other operational tools. Other than the extensions to the capability to the write API, this release brings no major alterations to the architecture.

### Integration and dependencies

GOCDB newly depends on the EGI CheckIn service to provide federated authentication and access without client certificates. When accessed using a client certificate, GOCDB continues to depend on no other tool.

## Release notes

### Requirements covered in the release

By August the Write API will have been extended to meet requirements of WLCG[[13]](#footnote-13). This will allow programmatic:

* Creation, update, and deletion of service endpoints
* Update of details of services

There will be no changes to the way authentication and authorisation for the write API since the previous release. Building upon the previous release, these updates allow changes to key entities within GOCDB programmatically. This represents a significant change in the way in which GOCDB works, allowing for much greater automated interaction with the information managed by GOCDB. This will help secure GOCDBs future in an evolving information space.

 A number of smaller bugs[[14]](#footnote-14) will also have been addressed.

## Feedback on satisfaction

Before every production release, GOCDB development is frozen and a period of testing is announced that lasts for approximately two weeks to one month using the GOCDB test instance[[15]](#footnote-15). This testing phase is widely disseminated using the relevant mail lists, and all operational tools and users are invited to perform tests against this instance. Recent GOCDB releases successfully passed this stage.

The GOCDB development process is described in Appendix II.

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| *Name of the result* | GOCDB |
| *DEFINITION*  |
| *Category of result* | Software & service innovation |
| *Description of the result* | * Extension of the write API.
 |
| *EXPLOITATION* |
| *Target group(s)* | WLCG tool developers, ARGO service, Resource/service provider admins and NGI managers  |
| *Needs* | The extension to the Write API will enable communities (e.g. WLCG) to further automate their interactions with the GOCDDB and move away from other information sources.  |
| *How the target groups will use the result?* | The results are integrated into the production instance of GOCDB, on which much of the target group’s infrastructure relies.  |
| *Benefits* | The result will improve the efficiency of target group’s use of the GOCDB service, as well as ensure its continuing fitness to serve them. |
| *How will you protect the results?* | Apache 2 licence |
| *Actions for exploitation* | The code needs to be integrated into the production instance of the GOCDB in order to provide the described functionality. The full source code will be available for use (under the Apache 2 licence) at <https://github.com/GOCDB/gocdb> |
| *URL to project result* | [https://github.com/GOCDB/gocdb/releases/tag/5.8](https://github.com/GOCDB/gocdb/releases/tag/5.7)[[16]](#footnote-16) <https://goc.egi.eu/>  |
| *Success criteria* | Regular use of the write API extension by at least one tool. |
| *DISSEMINATION* |
| *Key messages* | The Write API has now been extended to have greater functionality. |
| *Channels* | WP3 meetings, EGI OMB meetings, WLCG Information Systems Evolution Task Force |
| *Actions for dissemination* | Announcement emails to multiple EGI mailing lists and WLCG information system evolution mailing list.Description of new features to EGI Conference (May 2017: <https://indico.egi.eu/indico/event/3249/session/32/contribution/31>. |
| *Cost* |  |
| *Evaluation* | Uptake of use of new features. |

## Future plans

Future plans cover following aspects:

* Data freshness check[[17]](#footnote-17);
* Replacement of the GOCDB UI with a modern Web framework;
* Extending GOCDB in the info-service space supporting dynamic attributes;
* Improve change logging.

# Security Monitoring

## Introduction

|  |  |
| --- | --- |
| **Tool name** | Secant |
| **Tool url** | <https://github.com/CESNET/secant>  |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/Tools>  |
| **Description** | Secant is a framework to detect security vulnerabilities in images of virtual machines. It tries to detect the most common security issues that often lead to incidents and prevent them from appearing in the context of EGI cloud facilities. |
| **Value proposition** | Security incidents may cause significant problems for users, service providers and infrastructure operators. Secant was designed to detect common weakness in virtual appliances so that these can be fixed before they threaten a production infrastructure. |
| **Customer of the tool** | Cloud providers, VA owners, EGI operations, the EGI CSIRT |
| **User of the service** | Administrators, operators, security staff |
| **User Documentation**  | <https://github.com/CESNET/secant>  |
| **Technical Documentation**  | <https://github.com/CESNET/secant>  |
| **Product team** | CESNET |
| **License** | Apache License Version 2.0 |
| **Source code** | <https://github.com/CESNET/secant>  |

## Service architecture

### High-Level Service architecture

Secant runs as a service that periodically checks for new images available in a repository and performs their security assessment. When a new image becomes available in the system, it is taken by Secant and checked for security vulnerabilities. In order to perform the security checks, Secant instantiates a virtual machine from the appliance that is being verified and performs two phases of security checks. During the first phase, Secant launches a series of external scans that tries to detect vulnerabilities exposed by the machine to the Internet. Following these tests, and if the machine supports that, Secant runs a series of internal probes on the virtual machine, which checks security properties of the installed software. Both internal and external probes are modular and new tests can be easily added when needed. After the probes are executed, Secant processes the results and generated the assessment.

### Integration and dependencies

Secant needs to integrate support of a cloud management framework, which enables to both manage virtual machines and maintain the list of images to assess. The current implementation supports OpenNebula for the management of virtual machine and uses the EGI CloudKeeper[[18]](#footnote-18) to maintain the list of images and templates in the cloud repository.

In order to facilitate integration with existing infrastructure services, support for a messaging has been introduced recently. Secant uses the ARGO messaging to consume information about available images and to deliver assessment reports once assessment has been finished.

## Release notes

### Requirements covered in the release

Following the principles of continuous delivery, Secant does not have fixed releases. The outcomes of recent development are always available from the pilot installation deployed at CESNET. The features introduced recently involve integration of the EGI Messaging, support of CloudKeeper, and integration work aiming at utilization of Secant for the Application Database and EGI endorsement of virtual appliances.

## Feedback on satisfaction

Secant runs in a piloting environment established at CESNET and its MetaCloud site. The development follows expectations of the EGI CSIRT team and the service was presented to the team several times. Since the integration works are on-going, assessment tests can only be executed manually. So far, several dozens of virtual appliances underwent testing by Secant and findings were incorporated by the developers.

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| *Name of the result* | Secant |
| *DEFINITION*  |
| *Category of result* | Software & service innovation |
| *Description of the result* | Secant is a framework to detect security vulnerabilities in images of virtual machines. It tries to detect the most common security issues that often lead to incidents and prevent them from appearing in the context of EGI cloud facilities. |
| *EXPLOITATION* |
| *Target group(s)* | Users, RIs, Resource centres, NGIs, security teams, VA endorsers. |
| *Needs* | Prevent from security incidents that misuse common vulnerabilities exposed by servers connected to the Internet. |
| *How the target groups will use the result?* | The tools will facilitate the endorsement process and will help the endorsers detect common weaknesses. The tools will also be available to users preparing their images or installations on the top of running virtual machines. |
| *Benefits* | Achieving a common security bottom line of virtual machines in clouds, based on shared knowledge and tooling. |
| *How will you protect the results?* | The tool is released under a standard open-source license. |
| *Actions for exploitation* | Secant will be freely available and its utilization documented. |
| *URL to project result* | <https://github.com/CESNET/secant>  |
| *Success criteria* | Availability of the tool for performing assessments. |
| *DISSEMINATION* |
| *Key messages* | Secant help identify common security vulnerabilities in virtual appliances. |
| *Channels* | EGI Conferences, meetings with cloud experts. |
| *Actions for dissemination* | Integration with AppDB will facilitate the introduction of the assessment in the endorsement process. |
| *Cost* |  |
| *Evaluation* | Utilization of Secant in endorsement process. |

## Future plans

After Secant has been fully integrated with AppDB, it will be necessary to overview the endorsement process to support the assessment. We will need to take into account emerging technologies (like containers) to examine their impact on the assessment process.

# Accounting Repository

## Introduction

The EGI Accounting Repository is an accounting tool that collects resource usage data from sites participating in the EGI and WLCG infrastructures as well as from sites belonging to other e-infrastructures and organisations that are collaborating with EGI, including OSG and NorduGrid. The Repository uses software from the APEL project run by the STFC.

The accounting information is gathered from different sensors into a central Accounting Repository where it is processed to generate statistical summaries that are available through the EGI Accounting Portal. Statistics are available for view at different levels of detail by users, VO managers, resource provider administrators, and anonymous users according to well-defined access rights. Table 1 provides a summary of the tool covered in this release.

Table 1. APEL tool summary.

|  |  |
| --- | --- |
| **Tool name** | APEL |
| **Tool URL** | <http://apel.github.io/> |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/Accounting_Repository> |
| **Description** | EGI Core Service – The Accounting Repository collects and stores user accounting records from various services offered by EGI. |
| **Value proposition** | Improved information about the usage of the cloud resources within the EGI infrastructure. Added storage systems as a source of accounting data |
| **Customer of the tool** | e-Infrastructures, research infrastructures and, in general, distributed infrastructures. |
| **User of the service** | Resource providers, NGI admins, EGI operations, end users. |
| **User Documentation**  | <https://wiki.egi.eu/wiki/APEL> |
| **Technical Documentation**  | <https://wiki.egi.eu/wiki/APEL> |
| **Product team** | STFC |
| **License** | Apache License, Version 2.0 |
| **Source code** | <https://github.com/apel/apel><https://github.com/apel/ssm> |

This section provides a short introduction to the components provided by the APEL project as part of the EGI Accounting Repository. Then, the high-level architecture of the tool and its components are described, along with the integrations and dependencies it has. Release notes and the results of testing for this release are then provided. Finally, plans for exploitation, dissemination, and future work are shown.

## Service architecture

### High-Level Service architecture

Figure 7 shows how the APEL client, central APEL server and the EGI Accounting Portal interact.



Figure 7. APEL components and their interactions. Components in red are provided by the APEL project.

1. Users can run an APEL parser to extract data from a batch system and place it into their APEL client database, or they can use third-party tools to extract batch, cloud or storage data. This data is then unloaded into a message format suitable for transmission.
2. Users run Secure Stomp Messenger[[19]](#footnote-19) (SSM) to send these messages containing records via the EGI Message Brokers to the central APEL server. The messages can contain either individual records or aggregated collections known as summary records. This is configurable in the APEL client.
3. The central APEL server runs an instance of SSM, which receives these messages, and a loader process reads the records in the messages and loads them into a MySQL database.
4. A summariser process runs to create summaries of any individual records received and loads them into a “super summaries” table along with any received summary records. This summariser runs as a cron job approximately once a day.
5. A database unloader process unloads the summary records into a message to be sent on by a sending SSM, via the EGI Message Brokers, to the EGI Accounting Portal.

This release has added storage systems as a source of accounting data.

### Integration and dependencies

All communication between clients and servers is via the EGI Message Broker network using the APEL SSM package. The SSM software can be configured to send or receive messages. Where the messages are destined for is controlled by the queue, which is set in the SSM configuration.

The central APEL server uses the EGI service registry (GOCDB) to get a list of APEL endpoints so that only data from endpoints correctly defined in GOCDB are processed.

The SSM can be configured to get a list of message brokers from the EGI information system (querying a BDII) or it can be pointed directly at a message broker.

There are no changes to the dependencies in this release.

## Release notes

### Requirements covered in the release

The following list shows the changes that will be included in Accounting Repository by the end of EGI-Engage:

* Added support for cloud usage record v0.4 format.
* Added support for long running virtual machines in cloud accounting.
* Advanced storage accounting to production level.
* Added support for version 10 of the LSF batch system.
* Documentation of a method to extract APEL format records from non-APEL SQL databases.
* Developed a draft GPGPU usage schema for cloud accounting.
* Added support for more operating systems.
* Various minor fixes.

## Feedback on satisfaction

The APEL project uses a development process based around GitHub, which includes a semi-automatic testing procedure used to assess the quality of software releases.

For details of the testing procedure used, see Appendix III. This process is also detailed in the APEL Development Process document[[20]](#footnote-20). Table 2 summarises the results of testing this release.

Table 2. APEL software testing results.

|  |  |  |
| --- | --- | --- |
|  | ***Result*** | ***Link*** |
| **Unit tests** | Build passed | <https://travis-ci.org/apel/apel/builds/242356858> |
| **Coverage** | No reduction in coverage | <https://coveralls.io/builds/12374751> |

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| Name of the result | Accounting Repository |
| *DEFINITION*  |
| Category of result | Software and service innovation |
| Description of the result | Update to the software that provides the EGI Accounting Repository including a number of small fixes and improvements as well as support for a new cloud accounting usage record schema and storage accounting. |
| *EXPLOITATION* |
| Target group(s) | RIs, international research collaborations, service providers, Funding agencies and decision/policy makers |
| Needs | Usage accounting data that can aid in ensuring resources are used as expected. |
| How the target groups will use the result? | Service providers update client installations. Extra metrics collected in the repository will be presented in the Portal for various uses. |
| Benefits | Support for different version of batch system and packages now available for EL7 based systems. |
| How will you protect the results? | Open source license (Apache License, Version 2.0) |
| Actions for exploitation | Roll out update to production server infrastructure and package the software for use at the client end. Work with Accounting Portal to update views. |
| URL to project result | <https://github.com/apel/apel/releases/latest> |
| Success criteria | Smooth roll out and any issues resolved quickly |
| *DISSEMINATION* |
| Key messages | New version of the accounting software available that support extra metrics for cloud and storage accounting |
| Channels | EGI OMB, WP3 meetings |
| Actions for dissemination | Announce at an OMB and WP3 meeting |
| Cost | Low |
| Evaluation | Installation of new release and feedback on new features |

## Future plans

Although this is the final release of the EGI Accounting Repository under EGI-Engage, the repository will likely be developed further under follow-on projects. Areas that are a natural extension of work done under EGI-Engage include: applying the research done on big data tools to a new prototype accounting repository, extending support for GPGPU accounting if it becomes available in grid batch systems, and producing a production level dataset accounting service.

# Accounting Portal

## Introduction

|  |  |
| --- | --- |
| **Tool name** | Accounting Portal |
| **Tool url** | <https://accounting.egi.eu>  |
| **Tool wiki page** | <https://wiki.egi.eu/wiki/Accounting_Portal>  |
| **Description** | The Accounting Portal provides data accounting views for users, VO Managers, NGI operations and the general public. |
| **Value proposition** | Improved look & feel. New views that allow to aggregate data in different ways. Improved support for scientific disciplines. |
| **Customer of the tool** | Infrastructure users, VO Managers, Operations Centres, Sites and the general public. |
| **User of the service** | Infrastructure users, VO Managers, Operations Centres, Sites and the general public. |
| **User Documentation**  | <https://documents.egi.eu/public/ShowDocument?docid=2789>  |
| **Technical Documentation**  | <https://documents.egi.eu/public/ShowDocument?docid=2545>  |
| **Product team** | CESGA, CSIC |
| **License** | Apache  |
| **Source code** | <https://github.com/cesga-egi/accounting>  |

## Service architecture

### High-Level Service architecture

The Accounting Portal is a web application based on Apache and MySQL, which has as its primary function to provide users with customized accounting reports, containing tables and graphs, as web pages. It also offers RESTful web services to allow external entities to gather accounting data.

The basic architecture of the Portal consists on:

1. A backend, which aggregates both data and metadata in a MySQL database, using the APEL SSM messaging system[[21]](#footnote-21) to interact with the Accounting Repository and several scripts, which periodically gather the data and metadata described below;
2. A Model represented by the database schemas, both external and internal, which defines database tables for several types of accounting (HTC, Cloud, Storage, user statistics etc.) and metadata (topology, geographical data, Resource Centre status, nodes, VO users and admins, Resource Centre admins etc.), and a series of parametrised queries;
3. A set of views that exposes the data to the user. These views contain a form to set the parameters and metric of the report, a number of tables showing the data parametrised by two selectable dimensions and filtered by several parameters, a line graph showing the table data, and pie charts showing the percentage distribution on each dimension.

Figure 8. Accounting Portal architecture.

A graphical representation of these components is depicted on Figure 8

#### Backend

The Accounting Portal backend is a varied collection of messaging systems and scripts that gather accounting data and metadata from several external sources like GOCDB, the Operations Portal or WLCG REBUS for the portal consumption.

The accounting data are sent by each Resource Centre to the central APEL Accounting Repository, processed and made into summaries using internal processes by APEL. Data is organised to enable the ease implementation of optimised queries. Metadata is a category of data, which complement that raw data and allows the portal to organize, categorize and impart new meaning to it.

#### Model

The model in the portal is designed to interchange data with the Accounting Repository and other operational tools. The queries are parametrized to avoid SQL injections (SQL attack vectors based on malicious code on SQL input parameters).

Since there is a large number of possible queries, and the accounting data have many reads but are only written on updates from the repository, the portal can be very aggressive with database indexes, and there are periodic optimizations of these queries.

The queries have a common structure derived from the views:

* **Metric**: It is the number we want to use for the accounting, it varies from view to view (e.g. Number of VMs on cloud), but we usually have:
	+ Number of jobs: The number of jobs run, regardless the CPU or the time used.
	+ CPU time: The time used by CPU core in hours while executing jobs.
	+ Normalised CPU Time: The time used by CPU core multiplied by a corrective factor depending on a benchmark run on the machines. This benchmark is usually HEPSPEC06.
	+ Elapsed Time: The wall time, or real time spent in executing jobs, this should be greater than the CPU time since it also includes I/O and SO time.
	+ Normalised Elapsed Time: Wall time normalised in the same way that the CPU time.
	+ Efficiency: Wall time divided by CPU time. This indicated the percentage of time used doing calculation instead of doing I/O or servicing other tasks. This is important for pledges and VO admins.
	+ Monetary Cost: An estimation of the equivalent monetary cost of the accounted work, this is only an indicative value.
* **Time period**: All queries are limited to a time period expressed in months, and which can go from January 2004 to the present.
* **Dimensions:** All data shown in the portal is parametrized by two dimensions (the “rows” and “columns” of the tables), these include, but are not limited to:
	+ Date: The month of the accounting data.
	+ Region: The Operation Centre or federation in which it was accounted.
	+ Country: The country that the data was accounted for.
	+ VO: The VO that the jobs were run as.
	+ Resource Centre: The Resource Centre the data was accounted for.
	+ Number of processors: The number of cores used by the job.
* **VO Group**: The VOs that appear in the accounting:
	+ LHC: The VOs directly associated with the Large Hadron Collider in Geneva, comprises “alice”, “atlas”, “cms” and “lhcb”.
	+ TOP10: The top 10 VOs in the selected range in raw CPU consumption.
	+ ALL: All available VOs.
	+ Custom: It shows all VOs available in the range so the user can select which to display.
* **dteam VO**: It excludes the “dteam” and “ops” VOs that are only used for admin and test purposes.
* **Local Jobs**: Some Resource Centres can account jobs that have been processed locally on Resource Centre bypassing the infrastructure middleware. The available options are “Infrastructure Jobs only”, “Infrastructure and local jobs” and “Local Jobs only”.

There are customized reports and views, which use other inputs, but in general those are the usual inputs of the common queries.

#### SSM and Messaging

The Accounting Portal has to refresh its database periodically with data from the Accounting Repository to assure their freshness. The system uses the EGI Messaging System, a queue messaging system based on ActiveMQ, which is also used for the communications between Resource Centres and the Accounting Repository. Since the repository uses it internally for all communications, it is also needed to gather the accounting data from them. The SSM system is composed by:

* A SSM loader for each accounting source (multicore, cloud, storage, etc…). This daemon waits for messages arriving on a queue and authenticates it with a DN and certificate. If the message is deemed valid, it is saved to a spool directory for further processing.
* A DB loader, this daemon monitors the spool directory and if there are messages these are introduced in the DB in order. This introduction at present does not delete the previous data in the tables, it only overwrites it, then manual intervention is needed for stale data.

### Integration and dependencies

There are dependencies on other tools and components that provide metadata used in the portal. This metadata includes:

* **Geographical Metadata**: Resource providers’ country and Operation Centre affiliation. Generally, this follows current borders, but there are important exceptions. This is gathered from GOCDB using its XML-based API.
* **Topological Metadata**: Resource providers are presented in trees, there are Country and Operation Centre trees that correspond to geographical classifications, but there are also trees based on topological classifications like Tier1 and Tier2 Resource Centres, OSG Resource Centres and uncategorised Resource Centres. Inside Tier2 Resource Centres, the federation they belong to is also important and can trigger special code in some cases. Gathered from several sources, including OSG and WLCG databases.
* **Role Metadata**: VO members and managers, and Resource Centre admin records. This metadata controls the access to restricted views. Information is gathered from GOCDB and individual VOMS servers constructing a list of individual VOMSes and querying them with the VOMS API.
* **Country affiliation data**: Each user record contains a user identifier that has his/her user name, institution and sometimes country. Scripts in the backend map each user with the country of the institution which issues their certificate. These data are used in anonymised statistics per country on: how many resources from other countries are used by given country and the distribution of its resources used by other countries.
* **VO Data**: To make possible VO selection in the user interface, the portal stores lists of VOs. They are also used to filter incorrect VO names, provide access to VO managers, and arrange accounting by VO discipline (such as “High Energy Physics”, “Biomedicine”, “Earth Sciences”, etc.). Information is gathered from the Operations Portal using its XML based APIs.
* **Resource Centre status metadata**: Resource Centres must be filtered to exclude those that are not in production (because not certified yet, suspended, closed or being in test mode). There must be also metadata to aggregate the accounting history of Resource Centres whose name has been changed. Information is gathered from GOCDB using its XML tables and internal tables compiled as part of EGI PROC 15[[22]](#footnote-22).
* **Pledge metadata**: The WLCG reports have to contain only those Resource Centres where MoUs or other pledges between VOs and Resource Centres are honoured, so the validity date and pledged hours are needed. Information is gathered from WLCG using the REBUS service.
* ***Other metadata:*** There are also other metadata like local privileges, SpecInt calculations, publication status, VO activities and more. Some of these metadata is calculated internally using other types of metadata and published for other EGI operational tools, like VO activity data and Resource Centre UserDN publishing.

## Release notes

### Requirements covered in the release

* Implementation of the Storage Accounting views
* Added geographical JSON encoding options
* Add day, month, quarter, half-year and year scaling to hour based units
* Processors and initial Flavor variable support on cloud accounting
* Added processors and elap \* processors to cloud views
* Improved Scientific Discipline report
* Solved filtering of valid cloud Discipline classifications
* Metric Unit field changes to cosmetic one option select on non-hourly metrics
* Fixed unit definition matrix on wlcg pages
* REST API implementation (JSON + CSV output)
* Simplifying URLs and separating CSV + JSON links
* REST API documentation on a detailed wiki form
* Mass mailing notification support for VO Managers and Resource Centre Admins
* Bug fixing

## Feedback on satisfaction

Several tests were executed in collaboration with the EGI UCST and Operations Team. User communities were involved in the testing phase and the portal was updated according to the gathered requirements.

Feedback collected on the final release by all the stakeholders involved in the testing phase was very positive.

## Plan for Exploitation and Dissemination

|  |  |
| --- | --- |
| *Name of the result* | Accounting Portal |
| *DEFINITION*  |
| *Category of result* | Software & service innovation |
| *Description of the result* | Completed refactored portal with a modern and more attractive look & feel and several new features such as new home page, a WLCG specific sub-portal, new EGI reports, improved scientific discipline support, reorganized menus, contextualised help inline, improved CSV support, reimplemented VO metrics.  |
| *EXPLOITATION* |
| *Target group(s)* | Infrastructure users, VO Managers, Operations Centres, Resource providers and the general public. |
| *Needs* | Modern look & feel, new ways to access data, new reports. |
| *How the target groups will use the result?* | Reporting activities, problem solving, MoU estimation. |
| *Benefits* | Better reports, better problem solving, better MoU estimation. |
| *How will you protect the results?* | Attribution via open source license  |
| *Actions for exploitation* | The result is published in a public web page, immediately exploitable. |
| *URL to project result* | [http://accounting.egi.eu](http://accounting.egi.eu/) |
| *Success criteria* | Continued use. |
| *DISSEMINATION* |
| *Key messages* | A modern accounting portal with several new features is now available. |
| *Channels* | * Dissemination through the EGI conferences
* Article featured in the EGI newsletter
 |
| *Actions for dissemination* | EGI conferences, publications, participation to workshops organised by potential users |
| *Cost* |  |
| *Evaluation* | Number of accesses. |

## Future plans

Future plans cover following aspects:

* Additional options to aggregate data.
* Reports for spotting increasing/decreasing VO usage.
* Accounting data analytics.
* Dynamic pie charts.
* Change type of graph dynamically
* Support GPGPU Accounting.
* Support Data Accounting.
* Bug fixing.
1. ARGO development process

The following text is a copy of the “ARGO Development Process” document. The latest version of the document can be found here:

[https://docs.google.com/document/d/1W0pT-zcBHG1E\_hfftW67DH01LBZC7zMKLlIgJTlsFh8/edit#](https://docs.google.com/document/d/1W0pT-zcBHG1E_hfftW67DH01LBZC7zMKLlIgJTlsFh8/edit)

**Open development**

We follow an open development process. All the repositories of ARGO are hosted on GitHub under the ARGOeu organization. Each component that can be standalone, is hosted in its own repository in the ARGOeu organization.

Each component should have a CONTRIBUTING guidelines document, describing how contributions can be made. There will be a general CONTRIBUTING guidelines document. Components that are maintained in their own repositories can should link to the general CONTRIBUTING guidelines document or have their own set of guidelines if required.

* <https://github.com/ARGOeu>

**Forked repositories**

Following the spirit of DVCS, each of us forks the repositories from GitHub to her/his own account. We can work on new or on-going features on our own forks and when we feel it is ready or whenever we want feedback from the rest of the team, and then we can open a pull request towards the respective ARGO repository.

Useful information:

* <https://help.github.com/articles/fork-a-repo>
* <https://help.github.com/articles/syncing-a-fork>

**Pull requests & core team**

All of the members of the core team should be able to merge pull requests in the ARGO repositories. The person who opens a pull request never merges it {her,him}self, but asks/expects another core team member to review it and merge it. The idea behind this is that at least two people (the committer and the reviewer) will be involved for each new feature that we develop.

Advices for a committer:

* Do commit early and often
* Do make useful commit messages (they will be used for the release CHANGELOG). Creating insightful and descriptive commit messages is one of the best things you can do for others who use the repository. It lets people quickly understand changes without having to read code. When doing “history archaeology” to answer some question, good commit messages become very important.
* Format of a commit message:
	+ Title: [Jira issue ID] - descriptive title
	+ Description: summary of your job with enough information so that a can understand the context and the intention of the change.



The person who opens a pull request should make sure that {s}he includes enough information so that the reviewer can understand the context and the intention of the changes proposed in the pull request. A member can use the PULL\_REQUEST\_TEMPLATE that is supported by GitHub since earlier this year. <https://github.com/blog/2111-issue-and-pull-request-templates>. It is strongly encouraged that we open pull requests as soon as possible in the developer process in order to trigger prompt feedback.

**1 pull request should refer to 1 feature, task, bug**. Pull requests that are not ready to be merged should be marked as Work-In-Progress (WIP). Having the pull request open, means that each commit is visible to the ARGO CI, which can then build the component, run all the unit tests and attempt to package the component and at the end provide status feedback within the pull request.

Useful information:

* <https://help.github.com/articles/creating-a-pull-request>
* <https://help.github.com/articles/checking-out-pull-requests-locally>
* <https://help.github.com/articles/creating-a-pull-request>
* <https://help.github.com/articles/merging-a-pull-request>
* <https://quickleft.com/blog/pull-request-templates-make-code-review-easier>
* <https://help.github.com/articles/merging-a-pull-request>

**Pull request review process**

When a feature is ready, the developer removes the WIP mark from the pull request. Removing the WIP mark effectively signals the rest of the team that the pull request can be peer reviewed. At least one team member (other than the committer) has to act as the reviewer of the pull request. During the peer review process, the reviewer has to check the feature implemented, the code quality, the unit test coverage as computed, the existence of proper documentation and whether the component can be packaged successfully. If all these checks pass, then the reviewer can accept the pull request in order to be merged in the devel branch.

**Branches and builds**

Each repository should have at least 2 long-term branches:

* the devel branch, which should always be deployable
* the master branch, which should always be releasable

**Pull requests**

Pull requests for new features should be opened initially against the devel branch. For every pull request that is opened, the ARGO CI will execute the following workflow

Checkout pull request

Execute unit tests

Build Component

Build Ephemeral Packages

Report status to Github

Before a pull request can be merged in the devel branch, a member of the development team (other than the original committer) has to review the pull request and check the following according to the “Definition of Done”:

|  |  |  |
| --- | --- | --- |
| **#** | **Check** | **Status** |
| 1 | Quality of Code |  |
| 2 | Passes acceptance criteria automatic Unit tests for non-UI (80% or greater code coverage for business logic tier for new code) |  |
| 3 | CI build job is up-to-date and compiles, tests, and analyses the existing & newly added code |  |
| 4 | DB migration script for DB Schema tasks |  |
| 5 | Sufficient documentation:* APIs + Interfaces (public)
* Manuals (where applicable)
* Changelog / Release Notes
* Inline comments where 'complex' code
 |  |
| 6 | Ability to be properly packaged |  |

**Devel branches**

When new code is merged on the devel branch of each component, the CI system (a) picks it up, (b) builds the codebase, (c) runs again the unit tests, (d) runs the sonarqube code analysis suite and publishes the results on the ARGO sonarqube instance, (e) builds the devel packages and publishes them on the ARGO devel RPM repository, (f) extracts, builds the documentation and publishes it on the devel website and (g) reports the status of the CI on Github. New RPMs published on the devel RPM repository are automatically installed on the devel testbed.

Checkout Devel Branch

Execute unit tests

Build Component

Generate Devel Documentation

Build & Publish Devel Packages

Execute Code Analysis Tool

Report Status on Github

Deployment on Devel Testbed

The devel testbed is using actual production data and is being operationally monitoring by the same monitoring probes that are used to monitor also the production instance. Furthermore at the end of each sprint, the product team performs the sprint review ceremony in which the important features are presented to the ARGO stakeholders and live tested on the devel testbed. After the successful completion of the sprint review, the new code base is merged on each component's master branch.

In case more than one developer is working on the same component or a developer is working in parallel in more than one feature for the same component, the use of feature branches is advised.

The Devel branch is considered to be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release. Some would call this the **“integration branch”.** This is where automatic builds are built from.

When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number.

**Master Branches**

When new code is merged in the master branch of each component, the CI system picks it up and execute the follow workflow: (a) builds the codebase, (b) runs the unit tests again, (c) builds the production packages, (d) publishes them on the ARGO production RPM repository and (e) extracts & builds the documentation and publishes it on the ARGO website.

Checkout Master Branch

Execute unit tests

Build Component

Generate and Publish Prod Documentation

Build & Publish Prod Packages

Each time changes are merged back into master; this ***is a new production release by definition***.

Useful information:

* <http://martinfowler.com/bliki/FeatureBranch.html>

**Releases**

The release follows the process when new code is merged in the master branch of each component. Some prerequisites for a helpful release:

* **Spec files** should follow the correct release number shown in the following table. Spec files (%changelog) should not contain information about features or fixes, but information about changes in the package[[23]](#footnote-23). Do NOT put software's changelog at here. This changelog is for RPM itself. If the package has no changes, the description should say “New RPM package release”.
* **Release**: New release is created in the component repository. (Go to releases → Draft new release) The release contains the release number and detailed information. The information is created via the PR descriptions, so the PRs should have descriptive titles and messages. The release description should have the following sections:

|  |
| --- |
| # New features/Enhancements# Fixes# Documentation updates |

**Release numbers**

|  |  |
| --- | --- |
| v1.0.**[1]** | Patch release. A new minor release typically including just backwards-compatible bug fixes. No new functionality is added. |
| v1.**[1]**.1 | Feature release. MINOR version when you add new functionality in a backwards-compatible manner.  |
| v**[1]**.1.1 | Major release. Significant changes in the functionality. Mandatory if the changes are breaking backward compatibility. |

A todo list of a release is described in [this document](https://docs.google.com/spreadsheets/d/1D1Zbsk3z_LOe-q6E0Kv7b3r46fmNWhew7CSiDtNvnN4/edit#gid=0).

**Releases process**

**Planning**: On every **first meeting** of the month we plan the new features, functionalities (jira tasks) of the components. It is not obligatory to have new features, functionalities, fixes for all components. For the planning process a Jira Sprint will be used, with the selected jira tasks. It will be nice to comment and update the status of each Jira task.

**Testing**: All the new features, functionalities and fixes must be tested for 2 weeks at least in the devel infrastructure. This effectively means that, in the next release, only the features that are ready to be tested in the middle of the month will be included.

**Release**: All tested features, functionalities and fixes will be deployed to the production infrastructure at the beginning of the next month. If a feature, functionality, fix is not properly tested or requires more development it will be added to the next release.

**Process based on proc23**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Responsible** | **Action** | **Notes** |
| **1** | **Service Provider team** | **Once release is ready the team opens a GGUS ticket to Operations with the following information:*** **Name of the tool**
* **Date of release**
* **Release notes**
* **Suggested deployment date**
* **Testing instance url and testing instructions**
* **Names of testers if testing is manual (if not defined Development team may ask to appoint testers)**
 | **This refers only to monitoring boxes and WEB UI** |
| **2** | **Operations Team** | * **Inform the Noc-Managers about the upcoming release, asking if there is anyone else interested in performing the tests**
* **can add further people for performing the tests**
* **The suggested duration of the test phase is two weeks**
* **Update the ticket**
 |  |
| **3a** | **Operations Team / Noc-Managers** | **Update the ticket with the information on the performed tests and their result** |  |
| **3b** | **Service Provider team** | **Update the ticket with information about results of the overall testing phase** |  |
| **4** | **Service Provider team** | **Provide in the ticket the link to updated documentation** |  |
| **5** | **Service Provider team and Operations team** | **Agree on deployment date and update the ticket** |  |
| **6** | **Operations team** | **10 days before the upcoming deployment, inform the Noc-Managers.****Update the ticket** |  |
| **7** | **Service Provider team** | **Schedule a downtime of the service in case it is needed** |  |
| **8** | **Service Provider team** | **Deploy release and update the ticket** |  |
| **9** | **Operations team** | **Close the GGUS ticket after a week of the deployment only if the release was successful** |  |

**Step 1: Example**

[**https://ggus.eu/index.php?mode=ticket\_info&ticket\_id=129318&come\_from=submit**](https://ggus.eu/index.php?mode=ticket_info&ticket_id=129318&come_from=submit)

1. GOCDB development process

**Testing:**

* The GOCDB source code includes DBUnit and Unit tests for selected core packages. For a data-centric product like GOCDB, emphasis is placed on the DBUnit tests, which are essential to assert expected behaviour on the deployed RDBMS.
* The GOCDB test suite prioritizes quality functional testing of the most critical code-paths rather than achieving high blanket coverage of less meaningful tests.
* As of Jan/2016 this includes 67 DBUnit tests with 668 assertions.
* Coverage reporting is included for selected core packages (DAOs – 55%, Doctrine 35%, Gocdb\_Services 17%) and it is acknowledged that a higher coverage should be achieved for these packages.
* Continuous Integration is carried out on all pull requests to GitHub using Travis. This uses the unit tests to check the GOCDB code base against PhP 5.3, 5.4, 5.5, MySQL and SQLite (though not all the tests currently pass for SQLLite).

**Approach to Source Control:**

* The GOCDB project is hosted in GitHub under the GOCDB organization.
* The main GOCDB repository has two main branches ‘master’ and ‘dev’.
* The master branch is always ‘releasable’.
* The dev branch is always ‘deployable’.
* Developers fork the repository into their own personal repository to work on features using Topic branches.
* When ready, a pull request is opened against the ‘dev’ branch in the main repository for review by other team members.
* After review, the pull request is merged into the ‘dev’ branch.
* When ready, the dev branch is merged into master.
* Tags are subsequently created from the master branch to identify specific releases (v5.5. v5.6 etc).
* Throughout this process, the test suite is continuously executed and any failing tests addressed before creating pull requests and/or merging.
* For certain scenarios, we consider it acceptable to push commits directly to the dev branch rather than always enforcing pull requests which may add unnecessary overhead, such as making documentation changes or small rendering updates.
1. Accounting Repository dev process

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

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[[24]](#footnote-24). Code test coverage checking is performed by Coveralls[[25]](#footnote-25) and Python code quality checks. 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 until the reviewer is satisfied with the suitability of the changes.

Both of the main APEL repositories have two branches used to manage the source code: The development branch and the master 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. 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.

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.

1. <https://wiki.egi.eu/wiki/AAI> [↑](#footnote-ref-1)
2. [www.ggus.eu](http://go.egi.eu/eng) [↑](#footnote-ref-2)
3. [https://wiki.egi.eu/wiki/EGI\_CSIRT:Main\_Page](http://go.egi.eu/eng) [↑](#footnote-ref-3)
4. <https://documents.egi.eu/document/2660> [↑](#footnote-ref-4)
5. <https://wiki.egi.eu/wiki/OTAG#Operations_Portal_Advisory_and_Testing_Board> [↑](#footnote-ref-5)
6. <https://avro.apache.org/docs/1.2.0> [↑](#footnote-ref-6)
7. <http://hadoop.apache.org/> [↑](#footnote-ref-7)
8. <http://software.in2p3.fr/lavoisier/> [↑](#footnote-ref-8)
9. <http://w3.hepix.org/benchmarks/doku.php> [↑](#footnote-ref-9)
10. [http://argoeu-devel.github.io/messaging/v1/api\_metrics/](http://argoeu-devel.github.io/messaging/v1/api_metrics/%20)  [↑](#footnote-ref-10)
11. <https://github.com/ARGOeu/argo-ams-library> [↑](#footnote-ref-11)
12. <http://argoeu.github.io/messaging/v1/> [↑](#footnote-ref-12)
13. <https://rt.egi.eu/rt/Ticket/Display.html?id=11020> [↑](#footnote-ref-13)
14. from the GitHub bug list: <https://github.com/GOCDB/gocdb/issues> [↑](#footnote-ref-14)
15. <https://gocdb-test.esc.rl.ac.uk> [↑](#footnote-ref-15)
16. Link will not be live until release in August [↑](#footnote-ref-16)
17. <https://rt.egi.eu/rt/Ticket/Display.html?id=8240> [↑](#footnote-ref-17)
18. <https://appdb.egi.eu/store/software/cloudkeeper> [↑](#footnote-ref-18)
19. <https://github.com/apel/ssm> [↑](#footnote-ref-19)
20. <https://documents.egi.eu/document/2739> [↑](#footnote-ref-20)
21. <https://wiki.egi.eu/wiki/APEL/SSM> [↑](#footnote-ref-21)
22. <https://wiki.egi.eu/wiki/PROC15_Resource_Center_renaming> [↑](#footnote-ref-22)
23. <https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/Packagers_Guide/sect-Packagers_Guide-Creating_a_Basic_Spec_File.html> [↑](#footnote-ref-23)
24. <https://travis-ci.org/> [↑](#footnote-ref-24)
25. <https://coveralls.io/> [↑](#footnote-ref-25)