Multi-platform Automated Software Building and Packaging

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

2012 J. Phys.: Conf. Ser. 396 062001
(http://iopscience.iop.org/1742-6596/396/6/062001)

View the table of contents for this issue, or go to the journal homepage for more

Download details:
IP Address: 137.138.36.206
The article was downloaded on 06/05/2013 at 22:12

Please note that terms and conditions apply.
Abstract. One of the major goals of the EMI (European Middleware Initiative) project is the integration of several components of the pre-existing middleware (ARC, gLite, UNICORE and dCache) into a single consistent set of packages with uniform distributions and repositories. Those individual middleware projects have been developed in the last decade by tens of development teams and before EMI were all built and tested using different tools and dedicated services. The software, millions of lines of code, is written in several programming languages and supports multiple platforms. Therefore a viable solution ought to be able to build and test applications on multiple programming languages using common dependencies on all selected platforms. It should, in addition, package the resulting software in formats compatible with the popular Linux distributions, such as Fedora and Debian, and store them in repositories from which all EMI software can be accessed and installed in a uniform way. Despite this highly heterogeneous initial situation, a single common solution, with the aim of quickly automating the integration of the middleware products, had to be selected and implemented in a few months after the beginning of the EMI project. Because of the previous knowledge and the short time available in which to provide this common solution, the ETICS service, where the gLite middleware was already built for years, was selected. This contribution describes how the team in charge of providing a common EMI build and packaging infrastructure to the whole project has developed a homogeneous solution for releasing and packaging the EMI components from the initial set of tools used by the earlier middleware projects. An important element of the presentation is the developers experience and feedback on converging on ETICS and on the on-going work in order to finally add more widely used and supported build and packaging solutions of the Linux platforms.

1. Context and Initial Situation

The main goal of the Quality Assurance tools activities is the definition, selection, construction and support of a unified and integrated software engineering infrastructure for the software teams in the European Middleware Initiative (EMI) project [1]. Providing a single set of tools becomes a major challenge when the four middleware distributions composing the EMI project (ARC, gLite, dCache and UNICORE) have been using four completely different tool chains for their lifecycles. The differences in requirements, in project size and characteristics and in goals have led to the adoption of different development solutions throughout the years.

To better understand the heterogeneity of the tools used initially by the product teams, a survey was circulated among developers’ teams and the results were analysed. The results showed a high diversity of tools, mainly based on the number and type of programming languages used, platforms supported and packaging formats distributed. It was finally based on complexity of the release process and on the size and distribution of the teams and on the lines of code of the software produced.
2. Solution adopted and improvements needed for the first two years of EMI

2.1. Build System Selection
The tool selected was ETICS [2] [6], the tool already used for several years by the gLite middleware for building and testing their middleware software. Some of the strong points in favour of ETICS, that make it the tool selected, were: it is built in a pristine environment where it can ensure that all the dependencies are explicitly specified; it is able to build multiple languages; the releases and integration tasks can be done using one unique tool and one unique point of access for all the information, it automatically generates repositories with the packages and reports that have been generated and its plugin systems allows the collection of metrics.

2.2. Improvements Needed
Nonetheless, several modifications were needed to fulfill the EMI development and release requirements. Primarily, the installation of external dependencies had to be done not the ETICS repository but from the version available in the EPEL or the Debian repositories. ETICS had to be adapted to install whatever version was available in those official repositories. New platforms were requested: Scientific Linux 5 (SL5) for the first year and Scientific Linux 6 (SL6) and Debian 6 for the second year; and all for 32 and 64 bits architectures.

2.3. Platforms Required
The build system had to produce packages compatible with EPEL and Debian guidelines of the Linux Fedora/EPEL and Debian and use standard tools such as Mock for Fedora/EPEL and PBuilder for Debian. These tools create a clean change root environment where all the dependencies are installed before the package is built.

The build infrastructure had to be updated, adding about thirty virtual build hosts, to support hundreds of daily jobs. Its reliability and maintainability should be also improved. The repository module had to be extended to correctly handle the last versions of RPM packages and to allow the generation of APT repositories.

The ETICS plugins system had to be extended with new static code analysis tools to provide the capacity of collecting the quality metrics defined and statistical data from each build job and from all used bug trackers.

3. How ETICS works

3.1. ETICS Rationale
ETICS organizes the project in modules and each module has configurations that match tags or branches of source code. To build one of those configurations, after selecting the necessary options, one clean worker node of the selected type is instantiated, the ETICS client is installed and the build begins. All the source code is downloaded according to the information provided in the configuration; the dependencies are calculated and installed (or built and installed if necessary) and then the modules are built one by one, following the order of the dependency tree. At the end of the build, the reports and artifacts generated are stored in YUM or APT repositories and all the links to access the access links are sent to the user who submitted the build job. The plugins that extract metrics and statistics, if selected, are executed during the different execution phases to collect information.

3.2. ETICS User interfaces
There are two different user interfaces to interact with the ETICS system: a web portal application and a command line client. Most of the actions, including all the basic ones, are available in both interfaces.
Using the web portal (http://etics.cern.ch/eticsPortal), it is possible, among other tasks, to check the status of our builds, edit our configurations or open the content of the artifacts created in previous builds.

The command line client is an application written in Python that is installed in the worker nodes to perform the job tasks. It is also the tool used by the users to submit jobs and to script and automate the execution of the user jobs.

3.3. ETICS Infrastructure

The ETICS infrastructure is composed of the servers for the web portal application, databases and repositories, and the pool of worker nodes where the build jobs are executed. The pool used for running the EMI jobs is a set of Linux nodes, virtualized using Microsoft HyperV technology. The paradigm used to start the nodes had to be changed from a static to an elastic one. Initially, there were a fixed number of nodes always started per platform, but that was not enough to satisfy the new demand of jobs for the new platforms. With the new paradigm, the worker nodes are started on demand when the jobs arrive. The system is also leaving one extra worker node always free and ready to decrease the starting time as much as possible. At the end of the job the build node is deleted from the pool, a checkpoint is created (in case a user needs to access later) and restored to its initial clean state.

The worker nodes use a set of minimal images, covering all the platforms supported, previously created by the tools team. These images are created by installing the minimal set of dependencies to a basic installation of the Scientific Linux or Debian installation. These images and the instructions about how to create them are available for the users.

3.4. Service Management

The reliability of this infrastructure was set as the crucial objective of the ETICS services. To achieve this objective, a monitoring system was created. This system installs a daemon in each worker node and server. It executes, based on a XML configuration file, different scripts with a defined periodicity.

Figure 1 - ETICS portal detail
The daemon also checks if it is the latest version and if it is not, the latest one is downloaded and installed. The same is happening for the XML configuration file and the scripts. Those scripts control different aspects as the disk and AFS space availability, backup correctness, etc. Some of them solve the most recurrent problems that were appearing in the system and have a well-defined solution (e.g. clean cache areas, or slow restart processes.

4. Generation of compatible EPEL and Debian packages

One of the requirements of EMI is to create a set of packages compatible with the EPEL[4] and Debian[5] guidelines in order to produce packages compatible with those repositories. To achieve this objective, the integration of standard build and packaging tools were implemented within the ETICS system. The tools selected were Mock for generating the RPM packages and PBuilder for generating the DEB packages.

At the end of the build phase, these tools are triggered to create the final packages. The packages created are stored in the YUM or APT repositories created at the end of the build. The final goal is to make these packages eligible for inclusion in the official distributions’ repositories (EPEL and Debian) and the EGI UM repository.

The compliance level of the generated packages with the official guidelines is assessed using RPMlint for RPM packages or Lintian for Deb packages.

5. Metrics and Reports

One important feature of the ETICS system is its extensibility, using plugins to do extra actions during some of its phases. This feature has been used to provide the data for the QA activity. Many plugins have been added to the system to run static code analysis and other tests (SLOCCCount, FindBugs,
Checkstyle, PMD, RPMlint, CPPcheck, PyLint, IPv6, etc…). This information collected is analyzed and used to generate different types of reports for the different software lifecycle activities [7]. One of the major difficulties found here was the amount of different charts and types needed for the different quality control reports. A chart generator framework has been created to simplify and automate as much as possible the generation of charts for the existing and future QA reports. Figure 3 describes how it works:

![Diagram](image)

**Figure 3 - How the chart generator framework works**

The chart framework collects the information from the different bug trackers, the EMI dashboard and ETICS to generate, according to some definitions, the necessary charts for each report. One of the major advantages of this framework is that it can be easily expanded to include new types of charts.

6. Conclusions and Summary

From the heterogeneous initial situation, the unification into a common development service for the whole EMI software was achieved by adopting ETICS as a common tool for build, testing and packaging tasks. Initially ETICS was not covering all the requirements, but the new features introduced in the system fulfill all the current needs. These improvements, such as the integration of standard packaging tools for Fedora and Debian, provide to the EMI developers the common framework to perform all their daily tasks and at the same time provide also a complete and clear view to perform the EMI release manager tasks.

The integration of standard packaging tools, Mock and PBuilder, together with the new plugins to check their compliant level with the distributions packaging guidelines, RPMlint and Lintian, allow EMI developers to generate packages ready to include in the Fedora/EPEL and Debian repositories; one of the main goals of EMI.
Other tools developed, such as the chart generator framework, filled the gaps in the project that were not covered by ETICS. The creation of this reporting framework has reduced the time used to generate the QA reports, allowing the Quality Assurance team to focus in the QA analysis instead of the generation of such reports.

7. Acknowledgements
This work was partially funded by the EMI project under European Commission Grant Agreement INFSO-RI-261611.

References
[1] EMI project: http://www.eu-emi.eu/
[2] ETICS project: http://etics.web.cern.ch/etics/
[3] EMI SA2 tools wiki page: https://twiki.cern.ch/twiki/bin/view/EMI/SA2
[4] EPEL packaging guidelines: http://fedoraproject.org/wiki/EPEL/GuidelinesAndPolicies
[5] Debian packaging guidelines: http://www.debian.org/doc/debian-policy/
[6] Software Build Systems: Principles and Experience (ISBN-10: 0321717287)
[7] Measuring Performance : Using the new metrics to deploy strategy and improve performance (ISBN-10: 0970247117)