USING QUATTOR TO DEPLOY THE GLITE PROTOTYPE TESTBED

Marian Zurek, German Cancio-Melia, Alberto Di Meglio, Guillermo Diez-Andino Sancho, Joachim Flammer, Robert Harakaly, Lanxin Ma (CERN, Geneva, Switzerland)
Charles Loomis (LAL-Orsay)

Abstract

The gLite Middleware is composed of more than 25 different services, implemented in different languages, using different technologies and all coming with individual configuration needs. In addition gLite can be run in multiple operational scenarios and supports presently hundreds of configuration options. Past experience has shown that configuration and management are one of the biggest challenges of such a system. As part of the investigations of ways of configuring, deploying and managing gLite, the quattor system was tested as a candidate for large-scale installations. This paper will discuss how the functionality provided by quattor has been experimentally applied in the context of the internal gLite testbed management. Different aspects are described ranging from the quattor server installation itself to the population of the Software Repository and the definition of the Configuration Database. It is also shown how the required information can be generated automatically from the gLite build system using predefined configuration files and dependencies lists in order to allow seamless integration of gLite within the quattor system. The most challenging part, the service lifecycle management, has also been addressed. A quattor NCM component has been developed to transform the quattor data structures into gLite configuration files and to act on the gLite configuration scripts to reconfigure the service as information changes. Future steps and possible areas of improvements are described.

THE GLITE MIDDLEWARE

gLite [1] is the next generation middleware for grid computing. Born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as part of the EGEE project, gLite provides a leading-edge, best-of-breed framework for building grid applications tapping into the power of distributed computing and storage resources across the internet.

Currently, the gLite middleware (figure 1) consists of more than 25 different services, implemented in different languages, using different technologies and coming with individual configuration needs.

THE GLITE BUILD SYSTEM

The gLite build system [2] has been designed to satisfy a complex set of requirements: platform and language independence, flexibility and extensibility. This includes in particular the possibility of adding plug-ins to generate special configuration and dependencies files for a variety of tools and platforms.

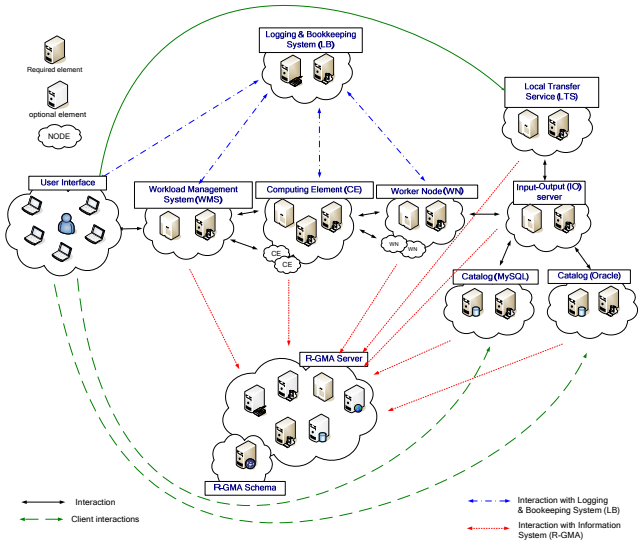

Figure 1: gLite middleware

In order to satisfy the language independence requirement and still be able to run a build of all the different components using the common configurations, the concept of *target wrappers* has been introduced. Each module in the project can be developed in any one of the supported languages using the appropriate build tool (Ant for Java, make and Autotools for C and C++, Perl makefile.PL, etc.). The target wrapper files exports a set of standard common targets, but have a different implementation of such targets depending on the underlying module language and build tool. For example, for Java the implementation is a native sequence of Ant tasks to perform initialization, quality assurance checks, compilation, unit testing, etc. For C++ on the other hand, the implementation may consist in a set of Ant commands that delegates the equivalent operation to autoconf, automake and make.

Configuration scripts

The gLite configuration scripts are a set of platform independent scripts, both service specific and common (e.g. for the MySQL configuration), implemented in Python. Python has been chosen because it combines the advantages of a scripting language with an object oriented approach. The script offers the possibility to configure, start and stop a service, to query the configuration information, to show the status of a service as well as some service specific options (e.g. removing part of a service). They are packaged in dedicated configuration RPMS within the gLite build system and offer a common interface to gLite service management [3].

Configuration files

A central part of the gLite configuration model is based on a common set of configuration files – implemented using XML – that contain all the necessary configuration information of the different services. While still being human readable, XML allows a proper arrangement as well as automated treatment of the configuration information using standard technologies like XSLT transformations.

The common configuration files and scripts protect the user from the underlying complexity and diversity of the individual services. Rather, the configuration system presents the user a homogeneous set of configuration files and file formats, as well as a command line interface with a unique look-and-feel [3].

quattor RPM templates

A quattor template plugin has been developed for the gLite build system to produce the required RPMS node templates using configuration information already present in the system to manage all service dependencies. The templates, one per high-level node such as the Workload Management System (WMS), Computing Element (CE) or the information system (R-GMA) Server and so on, can be directly committed to the quattor Configuration DataBase (CDB).

THE QUATTOR TOOLKIT

quattor is a system administration toolkit providing a powerful, portable and modular toolsuite for the automated installation, configuration and management of clusters and farms running UNIX derivates like Linux and Solaris. Within the quattor system, a system administrator describes the configuration of a set of machines, and the services on them, using a High Level Description (HLD) language called *pan*. This task-specific language permits the definition of the configuration schema, the configuration values, and complex validation functions. To fully configure a service, the list of packages and schema for the parameters are required and stored in the quattor configuration database (CDB). Figure 3 shows an overview of the quattor architecture. For more information refer to [4].

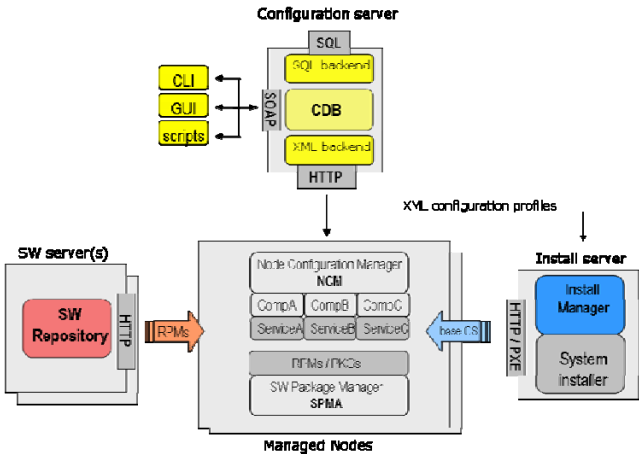

Figure 2: quattor architecture overview

quattor NCM components

The Node Configuration Management (NCM) subsystem of quattor is responsible for the effective deployment of the configuration information provided by the Central Configuration Database (CDB) and accessed via the Node View API.

The NCM provides a framework for adapting the actual configuration of a node to its desired configuration, as it is described in the node's profile inside the CDB. Plug-in software modules called 'components' are responsible for the configuration of local services, analogously to LCFG 'objects' or SUE 'features'. Components are used to enforce configuration policies. For this, they can read CDB configuration information via the NVA-API, and create/update/delete local service configuration files in order to match the CDB configuration description. Each component contains the knowledge for translating the CDB configuration into each local service's specific config file syntax. A component may also require notifying a service about a configuration change (e.g. by running a 'restart' or 'reload' method in a SysV init script). Components can be invoked by multiple means:

- by a dedicated daemon (cdispd), whenever a piece of configuration information they are interested in has changed in the CDB,
- on demand (e.g. by hand or via a remote execution mechanism),
- on regular intervals (eg. via cron).

A component does not change the local configuration if it matches the CDB description. Regular execution tasks, like purging/tmp, are to be implemented otherwise (e.g. using tmpwatch).

Examples of components include: a component for managing the sendmail configuration, a component for managing the network (TCP/IP, DNS) settings, a component for managing the configuration of the package management agent.

The NCM subsystem is deployed on every node to be managed centrally. These nodes can be of any type: farm batch (worker) nodes, NFS or web servers, or even desktop machines.

MANAGING THE GLITE CONFIGURATION USING THE NCM COMPONENT

As described in the previous section, a quattor system administrator describes the configuration of a set of machines, and the services on them, using the pan configuration language. This task-specific language permits the definition of the configuration schema, the configuration values, and complex validation functions with human-friendly syntax.

To fully configure a service, both the list of packages and a schema for the configuration parameters are required in pan format. The build system provides directly the package lists, including external dependencies, in pan format (see figure 3).

Complete configuration templates in well-defined XML format are provided as part of each gLite release. An XSLT transformation translates the provided configura-

tion files from the gLite-specific schema into the pan syntax. The transformation retains all of the default parameter values together with the schema for the configuration information and is used to populate the quattor CDB. Consequently, the system administrator, through pan, has complete control of the service configuration and only needs to define values for the user-defined parameters in the gLite schema of each high-level service.

The quattor system compiles and validates the configuration before sending it to the machines hosting the gLite services. At the moment all of the configuration parameters appear as strings in the pan configuration, taking little advantage of the validation features of the pan language. However, improvements are being added to include detailed type information in the gLite configuration. This will further reduce the likelihood of an incorrect configuration being deployed to running services.

On the quattor client the gLite NCM component reads the configuration from CBD, generates the configuration files in the XML-format required by gLite and performs any required configuration and management operation for the services using the common interface provided by the gLite configuration scripts. Usually a separate component is required for each service to be configured by quattor. However only a single component is necessary for gLite because of the consistent configuration and management provided by the gLite scripts, reducing the effort needed to manage the gLite service lifecycle.

Whenever a parameter value is changed in the CDB, an automatic notification can be sent to the gLite NCM component, which regenerates the XML configuration files and restart the services as necessary.

Overall, the consistent gLite configuration infrastructure permits a high level of integration with quattor. The results are less error-prone configuration, faster installation and more reliable operation of the grid services.

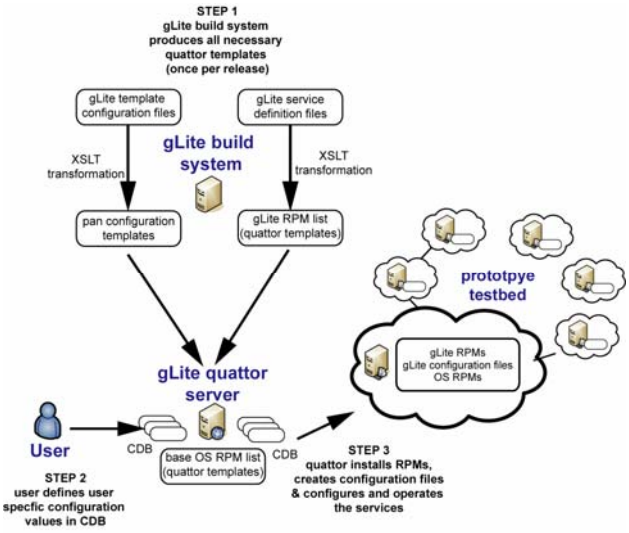

Figure 3: gLite build system and prototype testbed

THE GLITE 'PROTOTYPE' TESTBED AND DEPLOYMENT EXPERIENCES

The gLite/quattor integration has been tested with gLite 1.4 and gLite 1.5 releases on the so-called 'Prototype' testbed. The Prototype testbed is a dedicated integration testbed that advanced users and application developers can use to have early previews of the gLite technologies and test integration between the gLite middleware and grid applications. Although the testbed is small compared to a production grid environment, it offers all the diversity and complexity of a real-world scenario.

A dedicated quattor server has been installed to satisfy the gLite releases deployment. CERN Scientific Linux version 3 has been chosen as base operating system. A set of quattor templates has been created for a basic OS installation, using the package repository already present at CERN. This has allowed simplifying the servers' installations and avoiding duplication of efforts. The updates and default definitions are periodically synchronised to satisfy the security and site policy updates.

For each gLite service a top-level template has been defined to fully define the node operational package set. Those top-level templates are generated by the gLite build system and included in the profiles specific for each service.

New templates are generated at every new release or when packages quick fixes (patches) are released. This allows a very fast deployment cycle and strict control of the node configuration.

In total 12 machines are operational within the Prototype testbed. Among them one may identify: information system (R-GMA & BDII), Virtual Organization Membership Service (VOMS), Workload Management System (WMS), Logging and Bookkeeping (LB), Computing Element (CE) – 2 nodes, Worker Nodes (WN) – 4 nodes, File Transfer Service/File Placement service (FTS/FPS) and Load Sharing Facility (LSF).

CONCLUSIONS

The quattor toolkit enables the system administrator to fully automate the system installation, configuration and management of gLite services. It guarantees full reproducibility of a given configuration with the precision of the package release. The hierarchical mechanism allows the reuse of the generic templates, which can significantly reduce the total number of templates created for a given service.

However a number of areas have been identified where improvements could be made.

The pan language syntax is not the most intuitive and the compilation process not always produces clear messages in occasion of primary commit/compilation errors. Recent quattor development addresses this issue by means of Eclipse-compatible templates, which should improve the templates/component definition process.

One of the most desired missing features is the capability of quattor to handle multiple releases of the same component. This could greatly improve the management and deployment of multiple releases at the same time for testing purposes or even in real-life

applications when the presence of multiple releases of the same service on different nodes is required.

An additional related and welcome feature would be the possibility of tagging the templates directly in quattor. In other words this would enable the administrator to "get-back-to-the-last-known-good-configuration" in case of errors or simply in order to test different releases.

Finally a deeper integration with the very popular APT package management system could improve the system functionality. Although quattor can certainly replace APT for standard controlled deployments, it would be for example very useful to be able to 'synchronize' an existing APT-based installation with the quattor CDB for a node. This feature could be used for example to let APT "teach" the quattor CDB about an updated RPMS set (package versions and releases) and generate automatically updated templates for upload.

ACKNOWLEDGEMENTS

This work has been funded by the European Communities under contract IST-2003-508833.

REFERENCES

[1] http://www.glite.org

[2] A. Di Meglio, Joachim Flammer, Robert Harakaly, Elisabetta Ronchieri, Marian Zurek, *A Pattern-Based Continuous Integration Framework for Distributed EGEE Grid Middleware Development*, Proceedings of CHEP 2004, Interlaken, Switzerland, 2004

[3] J. Flammer, A. Di Meglio, G. Diez-Andino Sancho, Robert Harakaly, Lanxin Ma, Marian Zurek, *An XML-Based Configuration and Management System for the gLite Middleware*, Proceedings of CHEP 2006, Mumbai, India, 2006

[4] http://www.quattor.org