Deska: A Tool for Central Administration and Monitoring of a Grid Site

Jan Kundrát Martina Krejčová Tomáš Hubík Lukáš Kerpl

January 24, 2011

1 Background

One of the activities taking place at the Institute of Physics of the AS CR (FZU) [1] is running a local Tier-2 computing center connected to an international WLCG computing grid [2], supporting user programs from various scientific communities from the whole world including the D-0 experiment in Fermilab [3] and the famous LHC accelerator from CERN [4].

2 Motivation

In December 2010, the total installed computing capacity available to users was about 3000 CPU cores with more than 500 TB of disk storage, which translates to several hundreds of physical machines, with more hardware to arrive in early 2011. All the resources were managed by just a few system administrators (slightly more than three full time equivalents at the time of writing). Due to the impressive machines-per-administrator ratio, the staffers were exploring ways about how to make the system administrators' life easier and less error prone.

Over the years, certain management and monitoring systems have been deployed at the Institute, including Cfengine [5], Nagios [6], Ganglia [7], Munin [8] and various in-house monitoring appliances. While these tools certainly aim to make the sysadmin's life easier, introduction of each new tool also adds one more place for the administrator to go to when she wants to do any maintenance work. In practice, information about each particular machine or service is usually scattered among at least a dozen of places, some of which are notoriously prone to omission from the update process, leading to eventual inconsistencies. Therefore, it has been decided that the Institute shall fund a development of a tool which will *automate the configuration process* and focus on bringing the control back to one central place.

3 System Requirements

The goal of the project is to develop a database storing generic description of all the resources that the Grid site is using; such a database should be capable of describing hardware infrastructure of a data center[1], coupling of operating system instances (aka "hosts") to physical machines as well as logical relations between various services and their dependencies. This database shall then act as the single authoritative source of information for generating configuration of all components in the system.

3.1 Generic Database

History has proven that it is extremely hard to give accurate predictions about future developments and trends in computing. Many contemporary inventory management tools are severely limited by

[1] That is all physical machines, their hardware models, part numbers, rack locations, network interconnects, history of hardware failures etc.

constraints from the past. For example, even though the whole concept of virtualization predates the moment when personal computers became commodity, it is still common to stumble upon tools which cannot keep track of virtual machines or do not support the "blade" form factor of physical servers. Therefore, the Deska DB shall not impose similar restrictions to its data model, and the core DB code should only assume that it is working with *collections of objects* and some *relations between them* – actually defining a usable *scheme* of the database is a separate problem from the general database design.

As has been demonstrated numerous times during the Institute's course of operation, keeping track of all modifications is crucial for finding out what caused a particular outage. Therefore, the database shall support *versioning* of the records contained therein. We do not expect a need to support non-linear history (especially branching and merging), though.

User authentication and authorization shall be handled by already existing operating system facilities (PAM, SSH) and provisions for future ACL-checking hooks shall be considered. The whole system shall be able to scale up to thousands of managed objects on a commodity x86_64 server hardware.

3.2 Database API

The Database shall provide an API for access to the data contained therein. The first, read-only part of the API shall be complete enough to offer all functionality required by various modules and utilities described below. The second part of the API shall be used by the management interface for performing actual modifications to the data. It is up for the implementors to decide on the technical means of the API's "on-the-wire" protocol.

Bindings to scripting languages like Python shall be introduced, as the staffers prefer writing their own configuration backends in a language which allows faster prototyping than C++. Therefore, the API shall offer a complete bindings for Python, along with well documented sample modules.

3.3 Management Interface

Due to the Institute's administrators' strong Unix background and on their explicit request, it is expected that the interface to the database shall be provided via a CLI application instead of using a traditional web-based approach. The main reason for this design choice is their familiarity with text-based configuration tools and bad experience with using web applications in general. The application interface shall be similar to a CLI-oriented interface used on enterprise-grade Ethernet switches, notably to the Cisco IOS shell. The format shall enable storing of plaintext dumps of the database in a version control system for auditing purposes.

The CLI interface shall work on the same level of abstraction as the generic database, that is, it shall not contain any specific knowledge of the database scheme being used. All information required for the operation of the CLI shall be retrieved via the Deska's database API, and no changes to the CLI source code shall be required when the database scheme changes.

In order to streamline the day-to-day operation of the Institute, the CLI application shall include "wizards" for accelerating common tasks like deploying a new machine. A strong emphasis shall be given to usability. The CLI shall support a non-interactive mode of operation suitable for scripting and batched operations.

3.4 Database Scheme

In addition to the general-purpose database described above, the team shall design a scheme suitable for describing the whole IT infrastructure of the WLCG part of the Institute. This scheme should be extensible to allow installing future pieces of equipment without large-scale code changes, and shall try to anticipate future trends in hardware development, as well as account for more exotic hardware which could be found at Institute's partners' sites.

Nonetheless, the design of the general database shall be influenced by the real-world DB scheme, and shall offer reasonable assistance for administrators willing to extend the structure. The goal here is not to achieve purity at any cost, but implement a solution which is easy to deploy, use and maintain. An example of such assistance could be a script for converting the user-supplied database scheme into one directly usable in the Deska database.

3.5 Add-on Modules

Additional components for generating configuration on basis of the actual system state, as retrieved through the read-only database API, shall be developed for all systems which are currently deployed at FZU's site, namely for Nagios, BIND name service demon, DHCP server and Ganglia and Munin master servers. The design of the API should reflect generic needs of any data center to allow eventual deployment of new components, should the need arise. All modules should use only the public and documented API for retrieving all of the data they need.

In addition, tools shall be written to perform consistency checks between the state recorded in the database and the actual system shape where the automatic configuration generation is not feasible for safety reasons[2].

3.6 Visualization of the Datacenter Layout

The database API shall be used to employ the contained data for providing a graphical visualization of all computer rooms and all other equipment. These views shall be integrated with the pre-existing monitoring tools to provide more intuitive overview of possible issues encountered during operation. It has not been decided yet whether these tools are going to be web-based or implemented as a desktop application. Re-use of existing tools and frameworks is expected.

3.7 Management Utilities

The information contained in the database could be used for variety of other tasks besides generating configuration files. One practical use case is creating a unified interface for console access to the physical machines. This interface shall build on the existing tools for handling the actual access (likely a SSH tunnel, a standard KVM access or perhaps a Java-based browser method), but encapsulate all the details like physical location from the user.

A similar application is controlling the host power state; again, the goal here is not to reimplement the OpenIPMI implementation, but hide various differences in the protocol spoken by various machines, automatically using correct credentials for access etc.

These management utilities are likely going to be very site-specific, but effort shall be taken to attempt to properly separate local assumptions about concrete access method from the rest of the utility.

4 Problems to Solve

While the specification of an "inventory management tool" could be perceived as an easy task; that is, however, not the case. The biggest complication of the whole design is finding a right balance between usability and completeness. The Institute has had a web-based inventory management application capable of tracking various pieces of data for years, but the staffers were hesitant to use it and, more importantly, to *depend* on it for their day-to-day operation. As per an explicit request given by the staffers, this project shall develop a console-based tool with strong focus on scripting, ease of function and the general feel of *helping* them with getting things done instead of building artificial obstructions to their work for the sake of "evidence".

[2] A classic example of such a system are network switches – system administrators are rather leery when it comes to automated configuration as a single error can cause large network segments to separate.

Given the always-changing nature of the data center operations, it is unlikely that a static database structure would accommodate future developments. Therefore, the structure of the database shall not hard-code any current assumptions about how a datacenter looks like, but shall rather provide a way to describe generic *collections of objects and their hierarchies*. Needless to say, such an approach puts even bigger pressure on proper design of the system (and the database structure implemented at the Institute), as finding a balance between being "generic enough" and "usable enough" is extremely complicated. This necessary complexity, which is required in the DB level, shall not be obvious to the users when working with the UI – indeed, the interface should be intuitive enough so that the administrator will actually notice the improvements and start to like the system in favor of manual hacks.

The requirement of a dynamic database structure also makes the issue of versioning a rather complex problem, as it is now required to implement history management of almost arbitrary data.

The database shall offer demonstrable strength when faced with malicious input. It shall employ industry-standard means of defying security threats, and be friendly to firewall. The Institute is going to depend on the availability of the Database for its core operation, so it means that it should be more reliable than a typical gLite middleware service. The database should also scale to tens of thousands managed objects on contemporary server hardware.

5 Future Improvements

The system shall be extensible enough to be used outside of the Institute's computing infrastructure. A nice feature would be also a "one-click" tool for fully automated deployment of new machines, handling everything from setting up PXE environment for installation to actual system configuration.

6 Target Platform

The Institute is a heavy user of a custom Linux distribution based on Red Hat Enterprise Linux 5, which is going to be the platform of choice for running the server. The clients shall be reasonably easy to install on various Linux desktop distributions. All parts of the system should be written in a portable manner. Required maintenance effort shall be considered when choosing components and third-party libraries in order to deliver a sustainable solution.

References

[1] Institute of Physics of the AS CR home page, `http://www.fzu.cz/`

[2] The WLCG LHC Computing Grid, `http://lcg.web.cern.ch/LCG/`

[3] The DZero Experiment, `http://www-d0.fnal.gov/`

[4] The Large Hadron Collider, `http://lhc.web.cern.ch/lhc/`

[5] Cfengine, `http://www.cfengine.org/`

[6] Nagios, `http://www.nagios.org/`

[7] Ganglia, `http://ganglia.sourceforge.net/`

[8] Munin, `http://munin.projects.linpro.no/`