The data operation centre tool. Architecture and population strategies

Stefano Dal Pra\textsuperscript{1}, Alberto Crescente\textsuperscript{2}
\textsuperscript{1} INFN-CNAF, viale Berti-Pichat 6/2, 40127 Bologna, Italy
\textsuperscript{2} INFN-PADOVA, via Marzolo 8, 35100 Padova, Italy
E-mail: stefano.dalpra@cnaf.infn.it

Abstract. Keeping track of the layout of the informatic resources in a big datacenter is a complex task. DOCET is a database–based webtool designed and implemented at INFN. It aims at providing a uniform interface to manage and retrieve needed information about one or more datacenter, such as available hardware, software and their status.

Having a suitable application is however useless until most of the information about the centre are not inserted in the DOCET’s database. Manually inserting all the information from scratch is an unfeasible task.

After describing DOCET’s high level architecture, its main features and current development track, we present and discuss the work done to populate the DOCET database for the INFN-T1 site by retrieving information from a heterogenous variety of authoritative sources, such as DNS, DHCP, Quattor host profiles, etc. We then describe the work being done to integrate DOCET with some common management operation, such as adding a newly installed host to DHCP and DNS, or creating a suitable Quattor profile template for it.

1. Introduction

The Italian National Institute of Nuclear Physics (INFN) is an active sustainer of the World LHC Computing Grid (WLCG) developed by the HEP community. The Tier–1 at INFN–CNAF in Bologna is the largest datacenter of INFN, participating together with Tier–2 and minor sites to the WLCG infrastructure \cite{1}. The computing resources at the INFN–T1 have grown quite rapidly in the past few years, raising from 3000 cores, 2.6PB of disk storage (2009) to 13000 cores, 11PB disk storage (today). Tier1DB, the instrument initially adopted to catalogue these resources, proved some limitation in representing the variety of the hardware in the centre. As a consequence, its contained data became gradually inaccurate or obsolete, as new hardware was taking place and old hardware was being dismissed. In order to overcome these difficulties a new data operation centre tool, DOCET, was developed by INFN.

2. DOCET architecture

2.1. Objectives

DOCET was designed as an evolution of a previous datacenter database, Tier1DB, developed, deployed and used at INFN-T1. As such, it inherits experience from the previous product trying to be more generic and easy to adopt in different centres and to provide ways to solve specific issues arising from custom needs. To do so, a fair amount of design efforts have been spent to obtain:
Clear and terse GUI The web user interface to DOCET must easily provide most of the commonly wanted information and offer easy and quick ways to update or insert new data. Different views are available to explore the contents according to different user’s perspective (tree-view, device views, network views and so on).

Data completeness The data base schema must be able to hold a rich variety of data, so that answering to any question that may arise (i.e. “how many hosts are there with less than 8GB ram?”) is at least potentially possible.

Data access protection Data should be visible or modifiable by specific user groups, depending on membership. The groups are freely defined by the DOCET administrator.

Flexibility The database schema should be able to hold the description for almost any kind of current hardware. Moreover, it should accept new hardware products without requiring schema changes.

Multipurpose DOCET users should get the information they need despite what their particular task is. This means that the data presentation layer must be rich enough to satisfy different kind of users (administratives, technicians, administrators) and meet their typical use cases.

Data self-consistency New data should be consistent with existing ones. Two distinct hardware objects cannot have the same position, two network card shouldn’t have the same physical address and so on. On the other hand, partial knowledge about a new object should be allowed. This is mainly because quite often the complete set of information about an object comes from many subjects, each holding its piece.

2.2. Data model

We briefly describe here which kind of data are to be managed by DOCET and the logical data model that have been defined and implemented on the purpose.

2.2.1. Physical description. Physical objects are described in DOCET by their position and their composition. This is done by subsequently detailing locations and hardware. The information here provided would be enough to answer questions about existing objects: how many hosts of model X do we have? Where they are? And many other.

Locations From a top–down point of view, the object at a higher level is the site, defined by its name and related information. A site then contains one or more rooms, which in turn hosts shelves. The shelf finally contains hardware objects, and can be of many types (desk, rack etc. or user–defined). This is general enough to describe location for farm’s computers as also for office’s ones.

Hardware A hardware can be of many types (pre–defined or user–defined: switch, blade, enclosure . . .) and is described as a collection of components. Each component has its type and size (eg. Hard disk, 250GB) and manufacturer.

A device is a physical object defined as an instance of a specified hardware configuration.

Quite commonly a hardware can be a container for other hardware, such as enclosures and blades, and is of course possible to describe these hierarchies, as also network connections between host’s network card and switch’s ports. The physical representation can be fine–grained up to the description of any single network cable connecting two devices.

At the time of this writing, the enforcing of electrical data description have been evaluated and early extensions to the physical description are to be implemented.

2.2.2. Administrative and operational description. A complete physical representation of the owned hardware and connections between hosts can’t be fully useful until additional information are added.
Administrative information would let us to answer questions like: *Which hosts are older than 3 Years? Which ones have been brought from vendor X? Is host H under assistance?*

Operational information deals with device’s history and would let us to know or quickly summarize things like: *Which devices are broken and need to be repaired? What action have been taken on host H?*

To answer this, DOCET keeps track of vendors, purchase batches and terms of assistance and each device can be related to the purchase batch it belongs to. Moreover actions and failures can be added or updated for any device, as it happens to need it.

2.3. DOCET User interface

The web graphical user interface have been implemented conforming to the data model as in the previous section. The application have been developed as a 3–Tiers Java application: Data–Tier, Logic–Tier and Presentation–Tier. The Data–Tier interfaces to the database with Hibernate–PostgreSQL. The Logic–Tier is written in Java and the Presentation–Tier relies on the Vaadin libraries.

Figure 1. Hardware configurations and Failures views.

The User Interface is MDI (Multiple Document Interface), so that many windows can be opened at the same time inside the user’s web instance. The main window offers an upper–side menu bar and a lower–side status bar. The remaining space is intended to host the management windows. The menu bar offers: locations, components, configurations, administration, devices, network, virtualization, operations and views. Their usage and behaviour reflects the Data Model as described in the previous section. The management windows share a common look&feel, so to offer a uniform behaviour across different management sections.

2.4. The database schema

The DOCET database schema layout is organized into seven main sections:
- administrative (vendors, assistance etc.)
- device (computers, switch; physical objects)
- network (connections with and between devices)
- hardware configurations (built as collections of components)
- virtual machines
- location (physical disposition of devices)

3. Populating an empty docet instance

One of the earlyest DOCET versions were installed for two sites, INFN–PADOVA and the Tier–2 of INFN–LNL. It was populated, maintained and constantly upgraded by the developers, thanks to the feedback from users. Once mature enough the tool was adopted also at the Tier–1 centre of the INFN–CNAF. This was done with the challenging goal of check existing features and operational model against a bigger and more complex infrastructure, improve them to better fit with new and different sites, and possibly extending its capabilities by adding new features.

The first and major problem to face was that of defining a suitable way to gradually populate an empty instance up to a satisfactory level of completeness and correctness. DOCET requires that elements are to be inserted one at a time through the web GUI; eventhought batch insertion instruments are available, they would need to be somewhat more flexible in order to be fully useful in this case.

3.1. What we need

Direct insertion to the DOCET’s database has been considered as the only way to go.
The idea is to grab as much data as possible from any available and reliable electronic source, cross-correlate each piece of collected information and properly insert them in the database tables. It would be possible to obtain and insert a fair amount of correct data, and the amount of the missing ones should be little enough to make then manual insertion feasible.

What we fundamentally need is:

- an interface class to the DOCET’s PostgreSQL database, providing methods to manipulate insert or update values in the database schema and taking care of the too database specific gory details. This class was realized in python
- a suitable set of data sources (see next subsection)
- an intermediate data storage for partially collected data: a PostgreSQL support database t1db have been created. Useful to have partial data stored or updated there before being inserted or updated in DOCET
- a scripting language (python in our case) to do the work: extracting data from sources and inserting into the DOCET’s PostgreSQL database.

### 3.2. Information sources

The following sources of data have been taken into account to collect data from; some of them are considered as authoritative when they contains pieces of information which they are responsible of, and must be reliable.

Should an information from an authoritative source be false, then an immediate problem would happen and soon detected and tracked up to solution. This essentially means that
information retrieved from an authoritative source should be considered more reliable than those taken from other sources. Moreover, each data source is specialized to a specific purpose; thus, some information must be correct, while others may be weak or poorly specified. However, when putting it all together, the coherence constraints are stronger. This means that weak or poorly detailed information are detected just while cross-checking values from different data sources.

We detail in the following subsections the information sources that have been adopted and the kind of data they provide.

3.2.1. DNS. It provides IP→FQDN (Fully Qualified Host Name) mapping. No assumption can be made about physical existence of any hostname, nor it being a virtual or real machine.

Hosts in the INFN–T1 farm have hostnames in the .cr.cnaf.infn.it domain. Other hosts in the centre are in the .cnaf.infn.it domain. Data are retrieved from the local DNS by parsing the output of a dig reverse command: dig cr.cnaf.infn.it @<DNS> AXFR, and storing the results to t1db.

3.2.2. DHCP. Many hosts in the T1 farm have their network parameters configured by the Dynamic Host Configuration Protocol (DHCP). The configuration file, dhcpd.conf, as used at INFN-T1, contains vlan definitions and FQDN→MAC address mapping. At first, no assumption can be made about a FQDN being actually present in the DNS, nor its related MAC address being actually present: some entry may be related to a dismissed host. A few many subtle inconsistencies may be present in the dhcpd.conf file, both internal (a MAC or FQDN present twice in the file) or external (a FQDN not registered in the DNS, or declared inside an improper vlan section, according to its IP address).

Being the FQDN→MAC mapping quite important for the purpose of populating docet, the t1dhcp.py command line interface have been developed to manage, check and sanitize the dhcpd.conf file. It parses the file and import its contents as python’s data structures. Each operation performs consistency checks before committing changes (i.e. dumping back the python data structure as a correct dhcpd.conf) to the file.

3.2.3. ARP. Each host maintains its own ARP table: a cached mapping between IP and MAC of hosts in the same subnet. Usually it only contains a few entries, but we can force the ARP table to grow by issuing a ping to the broadcast address. After that the ARP table would contain entries for each host in the subnet. arp -n provides to us those entries.

3.2.4. SNMP and MIB. Connections between host’s network port and switch ports can be inserted in docet. To do so, an useful information would come by querying a switch to get a MAC, port_number map. This can be obtained by querying the switch using snmpget and specifying the object identifier (OID) of interest.

3.2.5. spreadsheets, textfiles. Usually, when delivering new hardware to the centre, the vendor provides a document with details of each single host; the serial code of the device and the mac addresses of the network ports in the device. If the device is a blade or a twin, the position in the enclosure is also specified.

A variety of textfiles or spreadsheets have been collected, detailing partial information, updated and maintained for particular purposes. Whenever possible, these have been parsed and used to integrate information in docet.
3.2.6. **Quattor**. Configuration of hosts in the farm is fully described by quattor [3] profile templates, expressed in *pan language*. By parsing pan files a bunch of useful information can be extracted for a given hostname:

- **hardware name**. The profile of each host always refers to a “hardware profile”. Hosts of the same type have the same hardware profile. Parsing it, details on hardware components could be extracted and used to create in *docet* the hardware configurations and their components. This has been actually done; however, there is no need for a quattor template to correctly or fully describe hardware components as this information may not be taken into account by the quattor system. Thus the hardware components listed into quattor’s hardware profiles could be not reliable.

- **software configuration**. This indicates the role of the host.

- **kernel version**.

- **rack and location**. The host’s position. Unfortunately this is not a reliable information.

- **virtual or real**. This makes possible to discriminate whether a machine is virtual, and which hypervisor hosts it.

3.2.7. **Tier1DB**. A partially populated and maintained database was available, with a fair subset of reliable information. These were migrated to *docet* too.

3.3. **Putting it all together**

The first attempt to populate *docet* was done by parsing and importing data from quattor profiles. This succeeded but some limits were evident. Most reliable information from quattor are related to installed software and configuration. Hardware description is however poor, and location is not reliable. Moreover, many devices (switches, some hosts and other devices) are simply not managed by quattor.

The physical location of a hardware is indeed the most difficult problem and partial solutions still are the only best thing to do. The position of Worker Nodes at INFN-T1 is self described in their hostname. For example wn-206-07-01-04-a is located in *room 2, rack 206-07, position 01*, and is the fourth blade in an enclosure (04). Further, -a stands for *eth0* and -b for *eth1*. Our best solution is to retrieve the location from the hostname. When this cannot be done, the location from quattor or from some other alternative source (textfiles, spreadsheets) is taken.

3.3.1. **Manual population**. A few things must be inserted manually anyway. These are a few, so this is not a problem:

- The site name and its rooms
- *docet* users.
- Purchase batches.
- Hardware configurations. These were at first automatically retrieved from quattor and then manually updated.

3.3.2. **Semi-automatic population and update**. This was done by combining the data collected from the above mentioned sources: FQDN and IP from DNS, MAC address and subnet details from *dhcpd.conf*. ARP provides MAC→IP mapping and it is used to integrate the data from *dhcpd.conf*.

Network connection from host and switch–port are obtained through snmp queries to the switches after leaning ideas about Network Discovery Techniques (see [4] and others).
Device’s Serial code and purchase batch is collected from available spreadsheets or other sources.

Whenever a mandatory piece of information is missing, DOCET gets fed up with a well known fake value. It may be updated to its correct value when processing other data sources.

The overall population process was done after writing a bunch of scripts, sometimes adapting them to particular cases, and applying them to the collected data using the python interface class to the DOCET’s PostgreSQL database. Once done, a few manual refinements have been enough to finally obtain a pretty complete and coherent population.

A portion of the developed code have been then adapted for update purposes, such as reflecting into DOCET changes made in the dhcpd.conf or on other authoritative sources.

4. Maintaining data up to date

Once DOCET has enough good quality data to be useful, it is important that changes in the farm are promptly reflected and reported to DOCET. This can be done manually through the web interface, but chances are that soon or later some misalignment may happen. Furthermore, there are quite frequently vast changes involving many hosts (Example: dismissal of a batch of hosts) that would require too much time to be executed using the web interface.

To overcome these difficulties and add some more flexibility, a command–line interface to DOCET has been developed.

4.1. DCQ

DCQ (DoCet Query) is the command line interface to docet. It is implemented through a simple python based webservice, who exposes a set of utilities to search or update or modify values in the docet database.

The client script is almost bare minimal. New features are implemented server–side without changing the client code. To make use of a new desired utility, it has to be implemented as a new method in a python class. This, in turn, is registered by the web-service and from then on, made available to the command line client. Clients are authorized by the webservice by their origin IP. Stronger authentication methods are being added. Repetitive tasks and “almost scriptable” ones can be executed this way.

The Nagios monitoring system used at INFN-T1 retrieves via dcq.py part of its configuration data, such as host roles, network connection (switch–port) and other.

When a device failure happen, it may be added in docet using dcq.py. Subsequently, that host will not be shown to nagios when reconfiguring again, until the failure gets closed.

t1dhcp.py is a commandline interface to manipulate dhcpd.conf, preventing inconsistencies (doubly inserted mac addresses, unexistent hostnames etc.). It provides search, update and delete functionalities. After any update changes are reflected to docet.

More functionalities are being added to DCQ. The mid–term goal is to get aid from DOCET to perform some management tasks. We mention a few:

- Better integration with quattor: producing templates for host or hardware profiles by mean of information known to docet for those host/hardware.
- Full integration with DHCP configuration. At present, changes in the dhcpd.conf file are propagated to DOCET. We would like to add to DOCET the ability to directly change DHCP configuration.
- Integration with the certificate management tool Sanctorum (cfr. 5th EGEE User Forum, Uppsala 2010). Sanctorum would ask docet for a list of hosts needing X509 certificates, and go on with its tasks with no further need of human interaction. That way the sanctorum host may completely disallow any kind of remote access to sanctorum. This would provide a really secure method to mostly automatize the management of host certificates.
5. Conclusions

Docet has a fine-grained and flexible level of detail for describing hardware in a big centre such as the T1 at INFN–CNAF and provides a comfortable graphic user interface. Populating docet from scratch can be done up to a good level of detail and manually refined with reasonably small effort. The procedure is however really difficult to be made general because it strongly depends on the nature of the data sources and their level of reliability. On the other hand, the roadmap to follow must be almost identical in all situations, and a certain amount of scripting and parsing capabilities will be enough to do the work. The population work had the positive side effect of enlightening inconsistencies between different data sources (example: FQDN specified in the dhcpd.conf and not registered in the DNS) and correct them.

After the population process was done, we started working to extend Docet’s capabilities. A command line interface has been added, and its capabilities are being improved and enriched. This is especially useful to maintain data up to date in a faster way when operating manually on a uniform list of changes. Moreover some tasks have been automated: adding in Docet a failure entry for broken hosts detected by other components, or closing it when they are detected to be back in production again. This helps in hosts management (opening a call for hardware assistance, keeping track of host downtimes); rewiring in the network connections can be detected and automatically updated in Docet.

Keeping up to date and reliable the information in Docet is easier and faster than before, thanks to a fair integration with the previously described information sources.

A final goal is to have Docet as a referral authoritative data repository for many management tasks. This could happen when the configuration for a given component (i.e. dhcpd.conf) can be safely derived from information managed by Docet.

References

[1] Canaparo, M., Capannini, F., Carota, L., Carpene, M., Cavalli, A., Cecchi, M., Cesini, D., Chierici, A., Ciaschini, V., Cristofori, A., Dal Pra, S., dell’Agnello, L., De Girolamo, D., Donatelli, M., Dongiovanni, D.N., Fattibene, E., Ferrari, T., Ferraro, A., Forti, A., Ghiselli, A., Gregori, D., Guizzanti, G., Italiano, A., Magnoni, L., Martelli, B., Mazzucato, M., Misurelli, G., Onofri, M., Paolini, A., Prosperini, A., Ricci, P.P., Ronchieri, E., Rosso, F., Salamone, D., Sapunenko, V., Venturi, V., Veraldi, R., Veroses, P., Vistoli, C., Vitlaci, D., Zani, S., Zappi, R., INFN-CNAF Activity in the TIER-I and GRID for LHC Experiments, Proceedings of IEEE International on Parallel & Distributed Processing Symposium (IPDPS), Rome, Italy, May 2009.

[2] http://www.vaadin.com/
[3] http://quattor.sourceforge.net/
[4] http://www.netdisco.org/
[5] http://indico.cern.ch/contributionDisplay.py?contribId=115&sessionId=6&confId=69338