Quattor Fabric Description

Michel Jouvin
jouvin@lal.in2p3.fr
June 16, 2006
Quattor Tutorial
T2 Workshop, CERN

- Description goals
- What is a machine profile ?
- Templates
- PAN language
- Templates layout
- Templates customization
- OS and MW upgrades
- Documentation and support

- Configuration description is a unique source of information for all Quattor components
- Quattor description is organized per machine
  - Machine profile (XML file)
  - Should allow to describe everything about HW and SW configuration
- Be (as) easy (as possible) to maintain
  - High level description language : PAN
  - Reusable configuration building blocks : templates

- A profile contains all the hardware and software configuration of a machine
- Hardware description
  - Used to size some elements (e.g. swap), select drivers… and validate system configuration (e.g. partition layout)
  - Mainly used at installation time
- System and software configuration
  - System and software RPMs to be deployed
  - System/service configuration for every system/application component
- Configuration information is a hierarchy with 3 standard branches
  - /hardware : /hardware/cpu, /hardware/ram…
  - /software : /software/components, /software/packages…
  - /system : /system/kernel/version

- Building blocks to describe a machine profile
  - Written in PAN language
- A template can include other templates
  - Some low level templates describing very specific part of the system configuration
    - E.g. : network configuration, how to start a service, to update a configuration file…
  - These templates are put together to produce a service description : NIS configuration, Torque configuration…
  - Service templates are put together to produce a machine type description (WN, CE…)
  - One machine type is used in a real machine profile with few customization
- PAN allows to build generic templates customized through variables

Template Example

- A typical WN profile template (they are all identical)

```
# Template name must match template file name
# A machine profile template must have ‘object’ keyword
object template profile_ipnls2005;

include pro_wn;

# Add repositories
include repository_common;
```

- WN machine type template using other templates…

```
template pro_wn;

#
# Include base configuration of a LCG2 node
#
include pro_lcg2_machine_config_base;

#
# LCG-2 WN configuration
#
include pro_lcg2_machine_config_wn;
include pro_software_lcg2_machine_wn_torque;

#
# Virtual organization configuration.
#
include pro_vo_alice_users;
include pro_vo_dteam_users;
```

- High level language for abstract description of machine configurations (HLD)
  - Developed as part of Quattor during EDG WP4
- Allow description of the final machine state
  - Not how to implement it
  - Comparison between desired state and current state is done by *components* (client side) to decide what to do
- Derived from declarative languages
  - Every statement is an assignment (except 'include')
  - Procedural programming (functions) possible on the righ hand side of assignments (DML)
    - No flow control in the template (only in DML)
  - Variables can be redefined (≠ declarative languages)
  - Variables can have a default value (independent of order

- Process a machine profile template to produce a profile Low Level Description (LLD)
  - Compiler available on any platform (including Windows)
  - Compiler output is a XML file (quite large…)
- 3 phases processing
  - Compilation : executes PAN statements to produce configuration information tree in memory
  - Validation : checks type constraints on path elements, including required resources or properties
    - During validation, no modification can be done in information tree
    - Can execute complex function to do validation
  - LLD creation : after successful validation, write profile LL
    - Nothing written in case of error during compilation or validation

PAN Syntax

- Mix of C and Perl…
  - Every statement must end with a ';'
  - Blocks of instruction (DML) are delimited by {};
  - Operators close to C's but work on string too
- Assignment LHS = path or variable
  - Path : a (quoted) string with a filename like syntax
  - Variable : an arbitrary (unquoted) string preceded by 'variable' keyword
- Dynamically and strongly typed language
  - Type of path or variable determined when created and cannot be changed without undefining it
  - Constraints can be set on path or variables
    - Checked during validation phase
- Default value defined with '?=' instead of '='
- Definitive reference is PAN specification
  - http://quattor.web.cern.ch/quattor/documentation.htm

PAN Assignments Exampl

```
'/hardware/memory/size' = 256;
'/hardware/cpus/0/vendor' = 'GenuineIntel';
'/hardware/cpus/0/model' = 'Pentium III (Coppermine)'
'/hardware/cpus/0/speed' = 800;
'/system/filesystems/0/name' = 'root';
'/system/filesystems/0/device' = '/dev/hda1';
'/system/filesystems/0/mountpoint' = '/';
'/system/filesystems/0/type' = 'ext2';
'/system/filesystems/0/options' = 'defaults';
'/system/filesystems/1/name' = 'cd';
'/system/filesystems/1/device' = '/dev/cdrom';
'/system/filesystems/1/mountpoint' = '/mnt/cdrom';
'/system/filesystems/1/type' = 'iso9660';
'/system/filesystems/1/options' = 'noauto,owner,ro';
```

PAN Assignments Example

```
# Assign values to a nlist
variable WN_AREAS = nlist(
   "alice", "/home/alicesgm",
   "atlas", "/home/atlassgm",
);

# Define a default value for the variable
# (exists but undefined). Used if no other definition
# made (before or after)
variable WN_AREAS ?= undef;


# Default value definition
# Actual value depends on another variable
variable WN_AREAS ?= if ( CE_NFS_ENABLED ) {
                           nlist(escape("/home"),CE_HOS
                         } else {
                           return(undef);
```

- Resources : list (array) and nlist (hash)
  - Can contain other elements (any type)
  - Created by list() and nlist() functions
  - Elements can be added by push() and npush()
    - Must be assigned to the list that must be modified
  - length() returns the number of element in the list
  - Accessed as array/hash in functions
  - A maximum number of elements can be defined
- Properties : simple type, assigned a value
  - String, boolean, int, double
  - Literals (constants) for all types, including true/false
- 2 specific literals
  - undef : variable/path is existing but is not defined and has no type (value of any type can be assigned)
  - null : variable/path is existing but will be deleted if not explicitly assigned

- User defined types : type mytype = {};
- Possible to define arbitrary records

```
type structure_ram = {
  # First element is another record
  include structure_annotation
  "size" : long descro "Size of module in MB"
  "data_rate" ? string
};
```

  - Can define any complex type
  - Elements can be optional (?) or mandatory (:)

PAN Variables and Valu

- Variable scoping : inside the block they are defined
  - Convention is to use lowercase for local variable
  - Some standard (global) variables are lowercase (self…)
- Global variables : defined outside any block
  - LHS of assignement prefixed with ‘variable’
  - Naming convention : uppercase
  - Global variables cannot be modified at a lower scope
- Default value : defined with ‘?=“
  - Used only if no other explicit defintion
  - Not sensitive to the order of definition
- Null value
  - Similar to undef, except that if no other explicit definitio the variable/path is deleted rather than staying undef
    - An undef path returns an error during validation

- Real workhorse of PAN…
- Built-in function : executed inside the compiler
  - Type query, length(), list/nlist creation/iteration, pattern matching
  - No string extraction functions
  - No bitwise functions
- Standard functions : defined in a standard template
  - Mainly list/nlist and software packages manipulation
    - push, npush, pkg_add, pkg_repl…
  - Main difference with a built-in function : performance
- User functions
  - Lot of ‘user functions’ defined in standard OS/MW templa
  - Can be defined anywhere with ‘function’ keyword
  - Function definition is an assignment…

PAN Includes

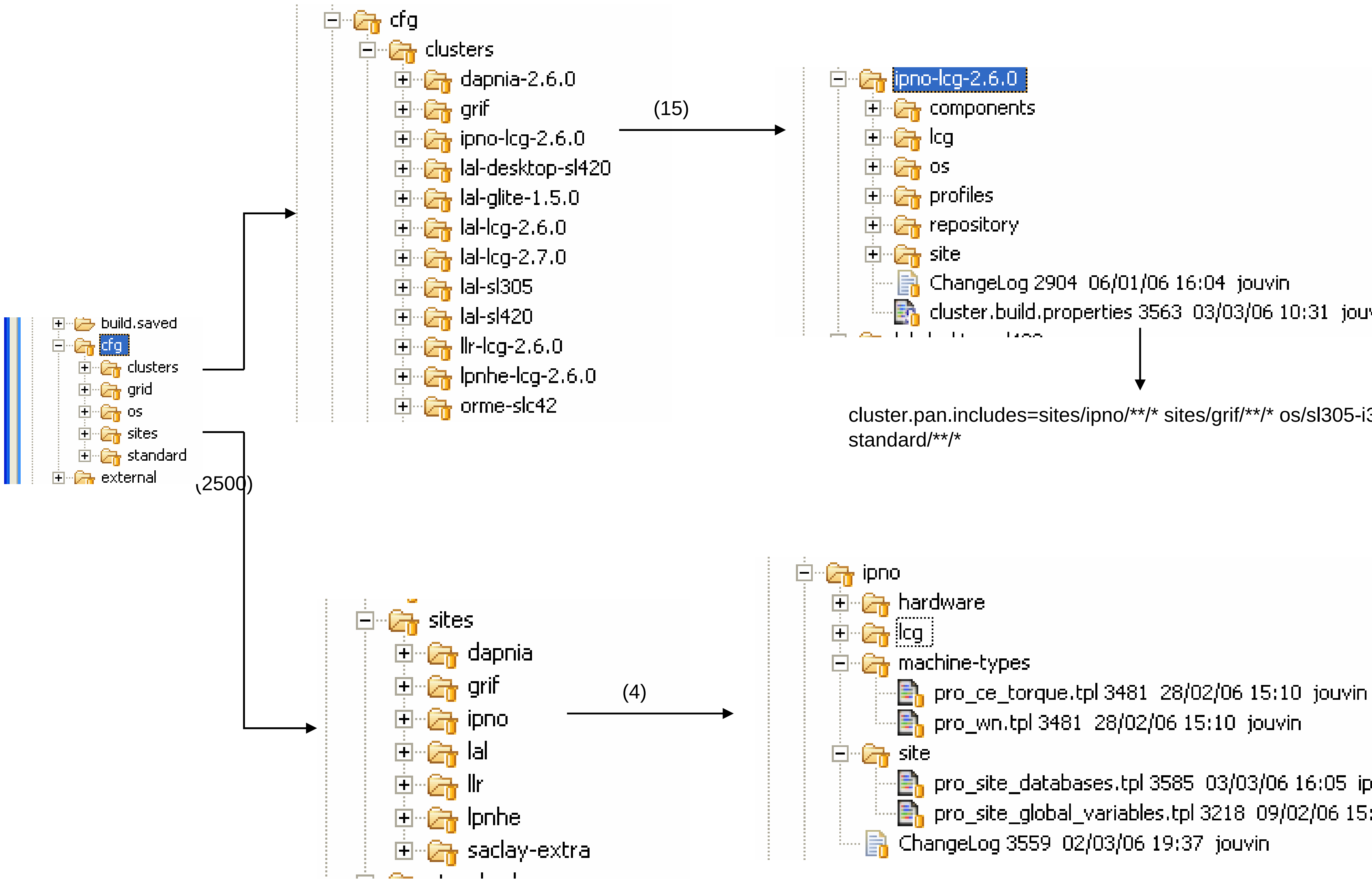
- Ability to include other templates is at the heart of
  - Give the ability to reuse templates as building blocks
- Normal includes : same effect as copying the conte of the included template in the current one
  - include my_other_template;
- Structure templates : resulting information tree is assigned to a path/variable
  - Structure template cannot be used with include
  - LHS paths must be relative (not to start with a /)
  - '/my/path' = create(template, [param_name, param_va
- Conditional includes
  - 'include' statement with a DML as file name (between {}
  - DML can be a variable name or a function
  - If DML returns 'null' value, nothing is included

```
variable LCG2_BASE_CONFIG_SITE ?= null;

# variable indicating if namespaces must be used to access OS templates
variable OS_TEMPLATE_NAMESPACE ?= false;
variable OS_NS_OS = if ( OS_TEMPLATE_NAMESPACE ) {
                             return("os/");
                    } else {
                      return("");
                    };

# Include OS version dependent RPMs
include { OS_NS_OS+"pro_os_lcg_base" };

# Include site configuration for LCG-2 software
include { LCG2_BASE_CONFIG_SITE };
```

- Number of templates can be very large…
  - A machine profile can be made of 300+ templates
  - With several OS/MW versions, CDB can contain 2000+ templates
- Layout goals
  - Avoid transforming template powerfulness into a nightm
  - Minimize the number of site specific templates and keep separate from standard templates
  - Allow several OS/MW version to coexist with minimum (!) template duplication
  - Support multi-site configuration database (repository)
- Layout described here fully supported with SCDB, partially (OS part) with CDB (not tested)
  - Nothing prevent full support by CDB but some works on existing templates required to add support for namespac

Templates Layout

- Machines are organized in “clusters”
  - Group of related machines, nothing to do with any cluste
  - Each cluster is a separate subtree of templates
  - For each cluster, define the OS and MW version used
    - SCDB : done with one cluster specific file : cluter.build.properties
- OS and MW templates : one directory (tree) per ve
  - Convention : os/ tree for OS templates, grid/ tree for MV
  - 1 cluster refers to 1 OS version and 1 MW version
  - OS templates : possible to select OS version per node to avoid creating 1 cluster for every MW/OS mix
  - All theses templates should not be modified
    - Most of them are generated, some are maintained manually by QW
- 1 tree for other standard templates : pan.. (standa
- To share site specific parameters between clusters must create a “site”
  - Just one more template tree the cluster is configured to



```
build.saved
cfg
  clusters
  grid
  os
  sites
  standard
external
```

(2500)

```
cfg
  clusters
    dapnia-2.6.0
    grif
    ipno-lcg-2.6.0
    lal-desktop-sl420
    lal-glite-1.5.0
    lal-lcg-2.6.0
    lal-lcg-2.7.0
    lal-sl305
    lal-sl420
    llr-lcg-2.6.0
    lpnhe-lcg-2.6.0
    orme-slc42
```

(15)

```
ipno-lcg-2.6.0
  components
  lcg
  os
  profiles
  repository
  site
  ChangeLog 2904  06/01/06 16:04  jouvin
  cluster.build.properties 3563  03/03/06 10:31  jou
```

cluster.pan.includes=sites/ipno/**/* sites/grif/**/* os/sl305-i3
standard/**/*

```
sites
  dapnia
  grif
  ipno
  lal
  llr
  lpnhe
  saclay-extra
```

(4)

```
ipno
  hardware
  lcg
  machine-types
    pro_ce_torque.tpl 3481  28/02/06 15:10  jouvin
    pro_wn.tpl 3481  28/02/06 15:10  jouvin
  site
    pro_site_databases.tpl 3585  03/03/06 16:05  ip
    pro_site_global_variables.tpl 3218  09/02/06 15:
  ChangeLog 3559  02/03/06 19:37  jouvin
```

- OS templates : mainly generated templates
  - A few (<10) templates version independent doing the mapping to actual version (pro_os_lcg_base…)
  - Nothing site specific
    - Except repository definition attached to each OS version
- MW templates (QWG) : generated templates (rpm lists) + manually maintained templates (service cor
  - Nothing site specific
    - Except repository definition attached to each MW version
- Other standard templates
  - Pan standard functions, schema…
    - Provided in Quattor core
  - Component related templates
    - Information tree for components, functions provided by component
    - Provided by each component (from Quattor CVS or ?)

Site Customizations

- Site customizations should be done (only) through variables used by standard templates
  - Parameter values, e.g. DNS domain name
  - Conditionals, e.g. shared NFS fs used on WNs
  - Name of site specific templates included by standard templates
    - E.g. file system partitions, site specific configuration for monitoring
- Some standard site specific templates :
  - pro_site_cluster_info .tpl : cluster specific parameters
    - 1 per cluster, all parameters except MW
    - Included at the very beginning of the configuration
  - pro_lcg2_config_site.tpl : all the parameters for the MW
    - Need to include pro_lcg2_config_site_defaults (generally at the en
  - pro_site_system_filesystems.tpl : define disk partitions
    - Variable FILESYSTEM_CONFIG_SITE can specify another template

Node IP and Hardware

- Recommendation is to have one template describin hardware used by a specific node
  - Build from templates describing a net card, a cpu, ram…
- Node IP and hardware are described in "databases" associating one node name with the corresponding IP address and hardware templates
  - 2 nlist variables : key is node fullname
  - Recommended template for these databases is pro_site_databases.tpl
  - Node fullname is retrieved from the profile name
- Side effect : any change in this template will trig a rebuild of all profiles
  - With a very large number of nodes, may consider splitting this template
- Node IP database could be generated from DNS…

- Basically the same procedure
- Install standard templates for new version in the repository
  - Customize repository location (in repository/)
- Create a new cluster, copying the existing one
  - Edit cluster.build.properties to reflect new version
- Move machine profiles from original cluster to new one and deploy
- For OS upgrade, it is also possible to select the OS version in the machine profile without creating a new cluster
  - Or to upgrade the whole cluster setting the default OS version in pro_cluster_config_site.tpl

- PAN language
  - http://quattor.web.cern.ch/quattor/documentation.htm
  - In particular, PAN specification : http://isscvs.cern.ch:8180/cgi-bin/cvsweb.cgi/~checkout~/elfms/quattor/documentation/pan-spec/pdf/pan-spec.pdf?rev=HEAD&content-type=application/pdf&cvsroot=elfms
- Templates layout and customization
  - https://trac.lal.in2p3.fr/LCGQWG
  - If you want to contribute, need an account (request me)
- QWG Templates source : SVN repository
  - https://trac.lal.in2p3.fr/LCGQWG/wiki/Download
- Support :
  - Bugs : Savanah http://quattor.web.cern.ch/quattor/bug_reports.htm
  - Help : mailing list project-quattor@cern.ch