High level design of Colibri's configuration database schema

By Amit Jambure <Amit_Jambure@xyratex.com>
Date: 2011-08-25
Revision: 1.0

[Text in square brackets and with a light-green background is a commentary explaining the structure of a design document. The rest is a design document.]
This document presents a high level design (HLD) of Colibri's configuration database schema The main purposes of this document are: (i) to be inspected by C2 architects and peer designers to ascertain that high level design is aligned with C2 architecture and other designs, and contains no defects, (ii) to be a source of material for Active Reviews of Intermediate Design (ARID) and detailed level design (DLD) of the same component, (iii) to serve as a design reference document.

The intended audience of this document consists of C2 customers, architects, designers and developers.

0. Introduction

[This section succinctly introduces the subject matter of the design. 1--2 paragraphs.]
Colibri maintains all the configuration information in some central repository from where it is served to all other nodes interested in the information. This HLD is about database schema used to maintain all the configuration.

1. Definitions

[Definitions of terms and concepts used by the design go here. The definitions must be as precise as possible. References to the C2 Glossary are permitted and encouraged. Agreed upon terminology should be incorporated in the glossary.]

- Colibri *configuration* is part of meta-data that is updated by management components, as opposed to meta-data updated as part of executing file system operations.
- *Configuration database* is a central repository of cluster configuration.
- *Configuration cache* is configuration data being stored in node's memory.
- *Configuration client (confc)* is a software module that manages node's configuration cache.
- *Configuration server (confd)* is a software module that mediates access to the configuration database. Also, the server node on which this module runs.
- *Trinity* is a set of utilities and GUI used by system administrators to monitor and influence operation of a cluster.

2. Requirements

[This section enumerates requirements collected and reviewed at the Requirements Analysis (RA) and Requirements Inspection (RI) phases of development. References to the appropriate RA and RI documents should go here. In addition this section lists architecture level requirements for the component from the Summary requirements table and appropriate architecture documentation.]

- [R.C2.CONFIGURATION.SCHEMA.CENTRALISED]
  Must store global configuration in a centralized location.

3. Design highlights

[This section briefly summarises key design decisions that are important for understanding of the functional and logical specifications and enumerates topics that a reader is advised to pay special attention to.]

4. Functional specification

[This section defines a functional structure of the designed component: the decomposition showing *what* the component does to address the requirements.]

Configuration database is created and maintained by confd.

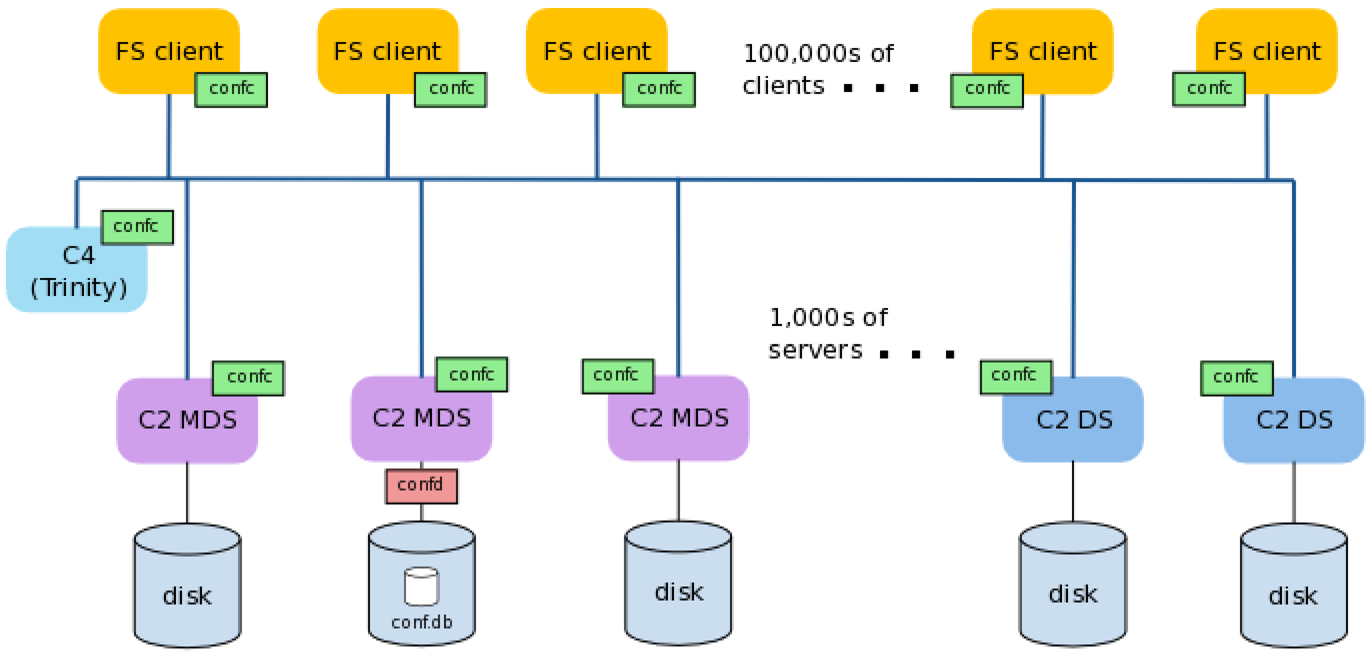
Configuration database contains information about following entities:

- Node
- storage devices
- disk-partitions
- nics
- file-systems
- services
- profiles
- storage pools

Other possible objects whose information can be kept in configuration database are:

- Colibri tuning parameters
- Routers
- Containers
- snapshots
- software versions
- enterprise user data-base
- security, keys

Position of configuration database in configuration subsystem (Diagram picked from HLD of configuration [3]):



5. Logical specification

[This section defines a logical structure of the designed component: the decomposition showing *how* the functional specification is met. Subcomponents and diagrams of their interrelations should go in this section.]

Configuration database is kept at some well-known location in persistent store on confd node. Configuration database is presented as set of tables where each table is a set of <key, value> pairs. Configuration database is implemented using c2_db_* interfaces. confd accesses configuration database using c2_db APIs.

When confd service starts, it opens the configuration database or creates it if it does not already exists.

Configuration database maintains separate table for each of following entities:

- nodes
- storage devices
- storage-device-partitions
- nics
- file-systems

- services
- profiles
- storage pools

Relationships between these entities in configuration database:

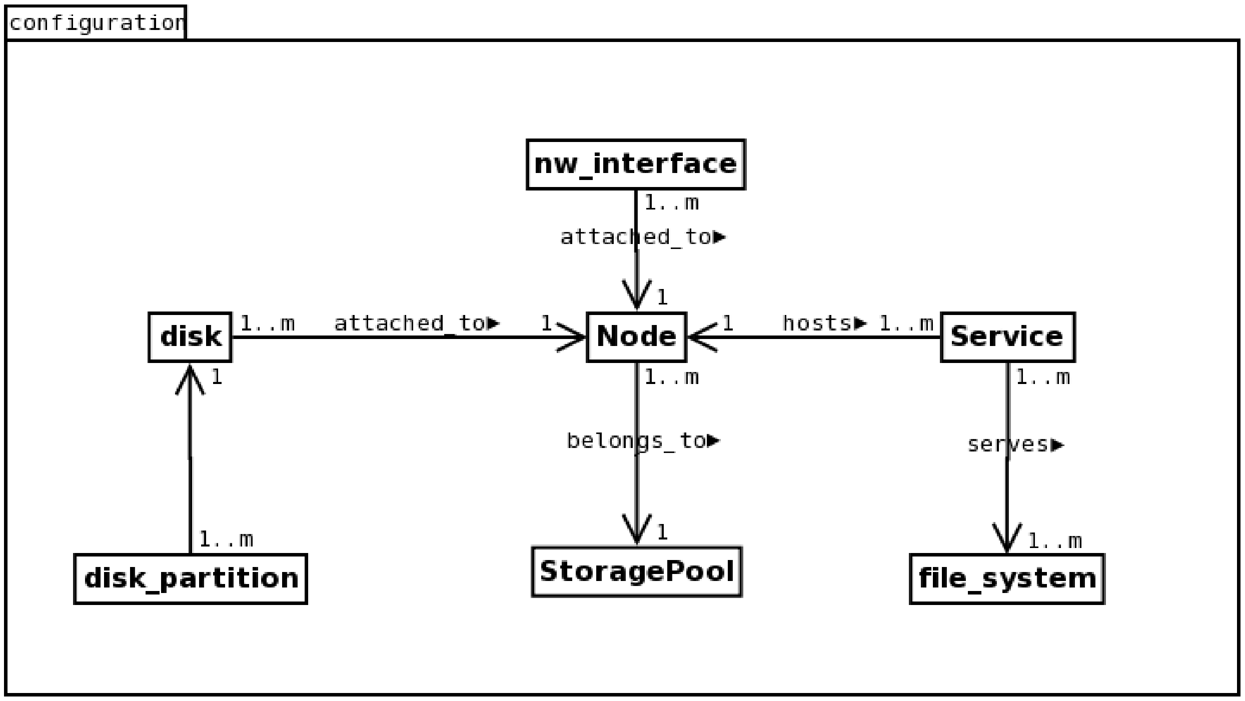


Table Name	nodes
Key	node_name
Record	- node-uuid/serial_num - memory_size - nr_processors - last-known-state (ONLINE/OFFLINE/FAILED/...) - is_virtual - native_endianness - pool_id
Comments	*node_name* is a human readable name independent of network. node_name of a node is unique within the cluster. *is_virtual* is true for virtual nodes and false for physical nodes. Trinity can provide unique name for each node. On linux box, *uuid* of a node can be taken from output of `dmidecode -s system-

	uuid\`. *serial number* of a node can be reteived from \`dmidecode -s baseboard-serial-number\`. nodes table maintains information about only server nodes. *pool_id* is a reference in *storage_pools* table, identifying record of a storage pool to which this node belongs.

Table Name	storage_devices
Key	dev_uuid
Record	- dev_interface_type (SCSI, ATA, ...) - dev_type (disk, flash or tape) - size - is_removable - is_read_only - status (OK, unresponsive, degraded) - node_name - file_name - format (ext3/ext4/...) - partition_table_type (raw (no-partition), DOS partition)
Comments	*dev_uuid* can be taken as serial-num from \`hdparm -I <dev_file>\` *node_name* is foreign key referring record, in *nodes* table, of the node to which the storage device belongs. *file_name* is name of device file that represents this storage device

Table Name	storage_device_partitions
Key	partition_uuid
Record	- start - size - dev_uuid - partition index - file_name

	- format (ext3/ext4/.../?)
Comments	*file_name* is name of block device file that represents this storage device partition. *start* is offset of partition from start of storage *size* is size of partition in number of sectors *dev_uuid* is foreign key referring record, in *storage_devices* table, of storage device to which this partition belongs. One possible way to define *partition_uuid* is <*dev_uuid, partition_index*>

Table Name	nics
Key	hw_addr
Value	- type (eth/ib/...) - node_name - mtu - state - speed
Comments	*node_name* is foreign key referring to record of the node to which this nic is connected, in nodes table.

Table Name	file_systems
Key	file_system_name
Value	root_fid
Comments	Other attributes of file-system e.g. disk-partitions that contain file-system data/metadata, list of services can be inferred from other tables.

Table Name	services
Key	service_uuid
Record	- type (md, io, lock, mgs...) - end_points[] - node_name

	- file_system_name
Comments	*node_name* is name of node on which the service is deployed *file_system_name* is name of file-system served by this service.

Table Name	storage_pools
Key	pool_id
Value	
Comments	At this point, a pool is a list of nodes which can be derived by scanning nodes table with matching *pool_id* attribute.

Table Name	profiles
Key	profile_name
Value	file_system_name
Comments	A profile is a topmost configuration object that is used to group all other configuration data. Profile is used to export multiple "personalities" from the same hardware. For example, profile would include a file system and user data base, so that by selecting a profile, users will see different sets of files and directories and have different identities.

5.1. Conformance

[For every requirement in the Requirements section, this sub-section explicitly describes how the requirement is discharged by the design. This section is part of a requirements tracking mechanism, so it should be formatted in some way suitable for (semi-)automatic processing.]

- [R.C2.CONFIGURATION.SCHEMA.CENTRALISED]
  Configuration database is kept on colibri confd.

5.2. Dependencies

[This sub-section enumerates other system and external components the component depends on. For every dependency a type of the dependency (uses, generalizes, *etc.*) must be specified together with the particular properties (requirements, invariants) the design depends upon. This section is part of a requirements tracking mechanism.]

- [R.C2.CONFIGURATION.NODE.NAME]
  Each node must have a human readable name that is unique within the names of all nodes in the cluster.
- [R.C2.CONFIGURATION.FS.NAME]
  Each file-system should have a human readable name that is unique within names of all file-systems in the cluster.
- [R.C2.CONFIGURATION.SERVICE.ENDPOINT]
  Value provided to end-point attribute of record in *Services* table must have enough information for clients to be able to connect to the service.

5.3. Security model

[The security model, if any, is described here.]

5.4. Refinement

[This sub-section enumerates design level requirements introduced by the design. These requirements are used as input requirements for the detailed level design of the component. This sub-section is part of a requirements tracking mechanism.]

6. State

[This section describes the additions or modifications to the system state (persistent, volatile) introduced by the component. As much of component behavior from the logical specification should be described as state machines as possible. The following sub-sections are repeated for every state machine.]

6.1. States, events, transitions

[This sub-section enumerates state machine states, input and output events and state transitions incurred by the events with a table or diagram of possible state transitions. UML state diagrams can be used here.]

6.2. State invariants

[This sub-section describes relations between parts of the state invariant through the state modifications.]

6.3. Concurrency control

[This sub-section describes what forms of concurrent access are possible and what forms on concurrency control (locking, queuing, *etc.*) are used to maintain consistency.]

7. Use cases

[This section describes how the component interacts with rest of the system and with the outside world.]

7.1. Scenarios

[This sub-section enumerates important use cases (to be later used as seed scenarios for ARID) and describes them in terms of logical specification.]

Scenario	[usecase.component.name]
Relevant quality attributes	[*e.g.*, fault tolerance, scalability, usability, re-usability]
Stimulus	[an incoming event that triggers the use case]
Stimulus source	[system or external world entity that caused the stimulus]
Environment	[part of the system involved in the scenario]
Artifact	[change to the system produced by the stimulus]
Response	[how the component responds to the system change]
Response measure	[qualitative and (preferably) quantitative measures of response that must be maintained]
Questions and issues	

[UML use case diagram can be used to describe a use case.]

7.2. Failures

[This sub-section defines relevant failures and reaction to them. Invariants maintained across the failures must be clearly stated. Reaction to Byzantine failures (*i.e.*, failures where a compromised component acts to invalidate system integrity) is described here.]

8. Analysis

8.1. Scalability

[This sub-section describes how the component reacts to the variation in input and configuration parameters: number of nodes, threads, requests, locks, utilization of resources (processor cycles, network and storage bandwidth, caches), *etc*. Configuration and work-load parameters affecting component behavior must be specified here.]

8.2. Other

[As applicable, this sub-section analyses other aspects of the design, *e.g.*, recoverability of a distributed state consistency, concurrency control issues.]

8.2. Rationale

[This sub-section describes why particular design was selected; what alternatives (alternative designs and variations of the design) were considered and rejected.]

9. Deployment

9.1. Compatibility

[Backward and forward compatibility issues are discussed here. Changes in system invariants (event ordering, failure modes, *etc.*)]

9.1.1. Network

9.1.2. Persistent storage

9.1.3. Core

[Interface changes. Changes to shared in-core data structures.]

9.2. Installation

[How the component is delivered and installed.]

10. References

[1] Configuration one-pager
[2] configuration_schema_notes
[3] HLD of configuration
[4] Colibri architecture meeting notes, Fremont 2010.12.20

11. Inspection process data

[The tables below are filled in by design inspectors and reviewers. Measurements and defects are transferred to the appropriate project data-bases as necessary.]

11.1. Logt

	Task	Phase	Part	Date	Planned time (min.)	Actual time (min.)	Comments

11.2. Logd

No.	Task	Summary	Reported by	Date reported	Comments
1	configuration				
2	configuration				
3	configuration				
4	configuration				
5	configuration				
6	configuration				
7	configuration				
8	configuration				
9	configuration				