Rudder User Documentation

COLLABORATORS

	TITLE : Rudder User Documentation		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jonathan Clarke, Nicolas Charles, and Fabrice Flore-Thebault	october 2011	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2.3.0	October 2011	First release of the Rudder User Documentation.	NC, JC, FFT

Contents

List of Figures

List of Tables

Online version

You can also read the *Rudder* User Documentation on the Web.

Chapter 1

Introduction

This chapter presents the main concepts and the architecture of *Rudder*: what are the server types and their interactions.

Reading this chapter will help you to learn the terms used, and to prepare the deployment of a *Rudder* installation. === Concepts

1.1 Rudder functions

Rudder addresses two main functions:

1. Configuration management;
2. Asset management;

The configuration management function relies on the asset management function. The purpose of the asset management function is to identify *Nodes* and some of their characteristics which can be useful to perform configuration management. The purpose of configuration management is to apply rules on *Nodes*. A rule can include the installation of a tool, the configuration of a service, the execution of a daemon, etc. To apply rules on *Nodes*, *Rudder* uses the informations produced by the asset management function to identify these *Nodes* and evaluate some specific informations about them.

1.2 Asset management concepts

Each *Node* is running a Rudder Agent*,* which is sending regularly an inventory to the Rudder *Server.*

1.2.1 New Nodes

Following the first inventory, *Nodes* are placed in a transit zone. You can then view the detail of their inventory, and accept the final *Node* in the *Rudder* database if desired. You may also reject the *Node*, if it is not a machine you would like to manage with *Rudder*.

1.2.2 Search Nodes

An advanced search engine allows you to identify the required *Nodes* (by name, IP address, OS, versions, etc.)

1.2.3 Groups of Nodes

You will have to create sets of *Nodes*, called groups. These groups are derived from search results, and can either be static or a dynamic :

Static group Group of *Nodes* based on search criteria. The search is performed once and the resulting list of *Nodes* is stored. Once declared, the list of nodes will not change, except manual change.

Dynamic group Group of *Nodes* based on search criteria. The search is replayed every time the group is queried. The list will always contain the nodes that match the criteria, even if the data nodes have changed since the group was created.

1.3 Configuration management concepts

We adopted the following terms to describe the configurations in *Rudder*:

Policy Template This is a configuration skeleton, adapted to a function or a particular service (eg DNS resolver configuration). This skeleton includes the configuration logic for this function or service, and can be set according to a list of variables (in the same example: *IP address*es of DNS servers, the default search box, . . .)

Policy Instance This is an instance of a *Policy Template,* which allows to set values for the parameters of the latter. Each *Policy Instance* can have an unique name. A Policy Instance should be completed with a short and a long description, and a collection of parameters for the variables defined by the *Policy Template.*

Configuration Rule It is the application of a policy to a group of nodes. It is the glue between both Asset Management and Configuration Management parts of the application.

***Applied Polic*y** This is the result of the conversion of a *Policy Instance* into a set of *CFEngine* Promises for a particular *Node*.

As illustrated in this summary diagram, the configuration rules are linking the functions of inventory management and configuration management.

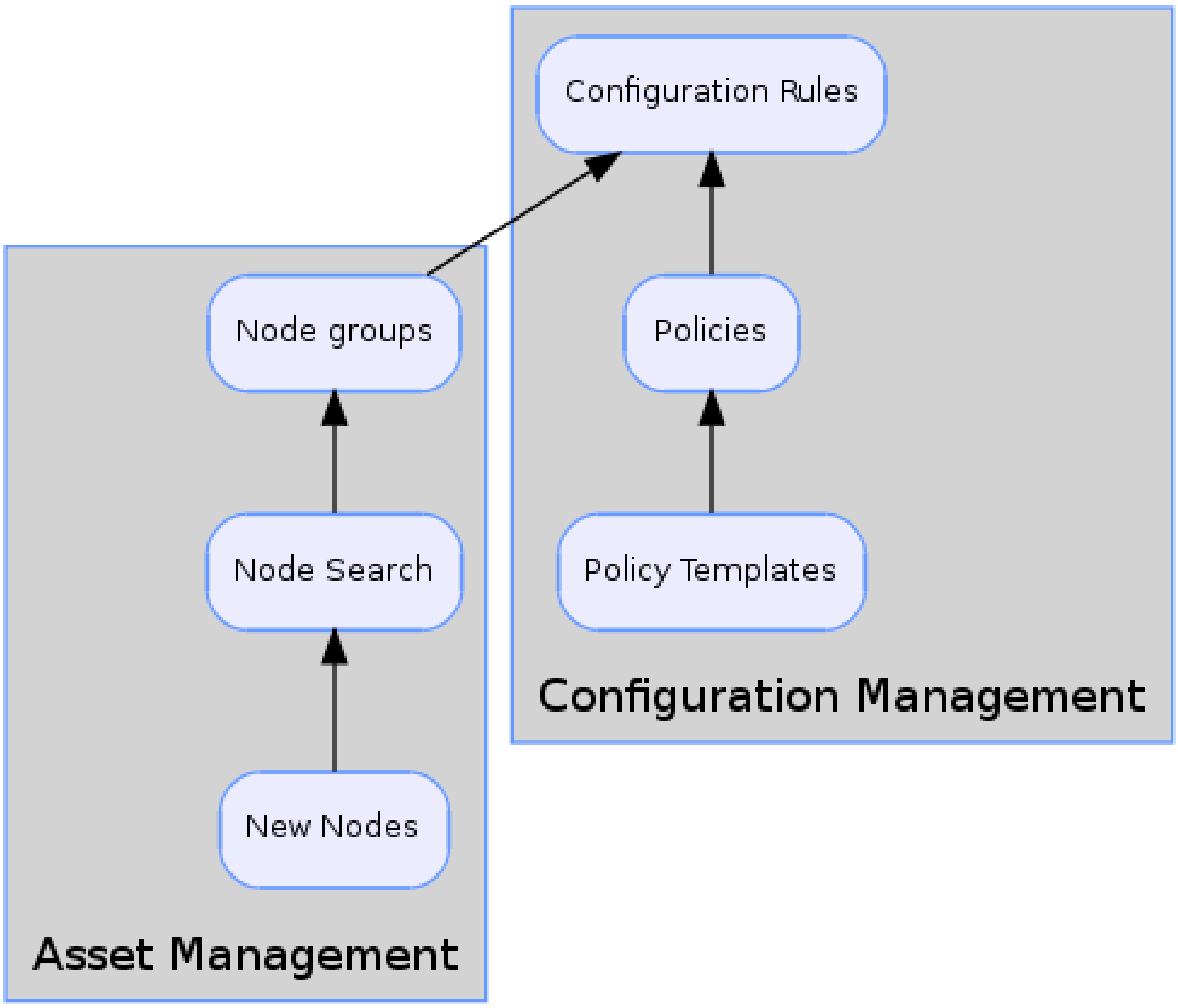


Figure 1.1: Concepts diagram

1.4 Rudder components

The *Rudder* infrastructure uses three types of machines:

Rudder Node A *Node* is client computer managed by *Rudder*. To be managed, a *Node* must first be accepted as an authorized node.

Rudder *Root Server* This is the core of the *Rudder* infrastructure. This server must be a dedicated machine (either virtual of physical), and contains the main application components: the web interface, databases, configuration data, logs…

Rudder *Relay Server* Relay servers are not available in the current version. In a future version, these optional servers will let you adapt your *Rudder* architecture to your existing network topology, by acting as a proxy for flows exchanged between managed nodes and the root server.

1.5 Specifications for Rudder Nodes

The following operating systems are supported for *Rudder Nodes* and packages are available for these platforms:

- *CentOS* GNU/Linux 5
- *Debian* GNU/Linux 5 (*Lenny*)
- *Debian* GNU/Linux 6 (*Squeeze*)
- Microsoft Windows Server 2000
- Microsoft Windows Server 2003
- Microsoft Windows Server 2008
- *RedHat* Enterprise Linux 5
- *SuSE* Linux Enterprise Server (SLES) 10 SP3
- *SuSE* Linux Enterprise Server (SLES) 11 SP1
- *Ubuntu* 10.04 LTS (lucid)
- *Ubuntu* 10.10 (maverick)

Windows Nodes
Installing *Rudder* on *Windows* requires the commercial version of *CFEngine* (named *Nova*). Hence, as a starting point, we suggest that you only use Linux machines. Once you are accustomed to *Rudder*, contact *Normation* to obtain a demo version for *Windows* platforms.

Unsupported Operating Systems
It is possible to use *Rudder* on other platforms than the ones listed here. However, we haven't tested the application on them, and can't currently supply any packages for them. Moreover, the *Policy Templates* are likely to fail. If you wish to try *Rudder* on other systems, please contact us.

1.6 Specifications for Rudder Root Server

1.6.1 Hardware specifications

A dedicated server is strongly recommended.

Your Rudder *Root Server* can be either a physical or a virtual machine.

At least 1024 MB of RAM must be available on the server, depending on the base requirements of your operating system.

Rudder *Server* is running on both 32 and 64 bit versions of every supported Operating System.

1.6.2 Supported Operating Systems

The following operating systems are supported as a Root server:

- *Debian* GNU/Linux 5 (*Lenny*)
- *Debian* GNU/Linux 6 (*Squeeze*)

- *SuSE* Linux Enterprise Server (SLES) 11 SP1

Unsupported Operating Systems
Installing Rudder *Servers* on other operating systems is planned, but has not been tested with the current version.

1.6.3 Packages

Rudder components are distributed as a set of packages.

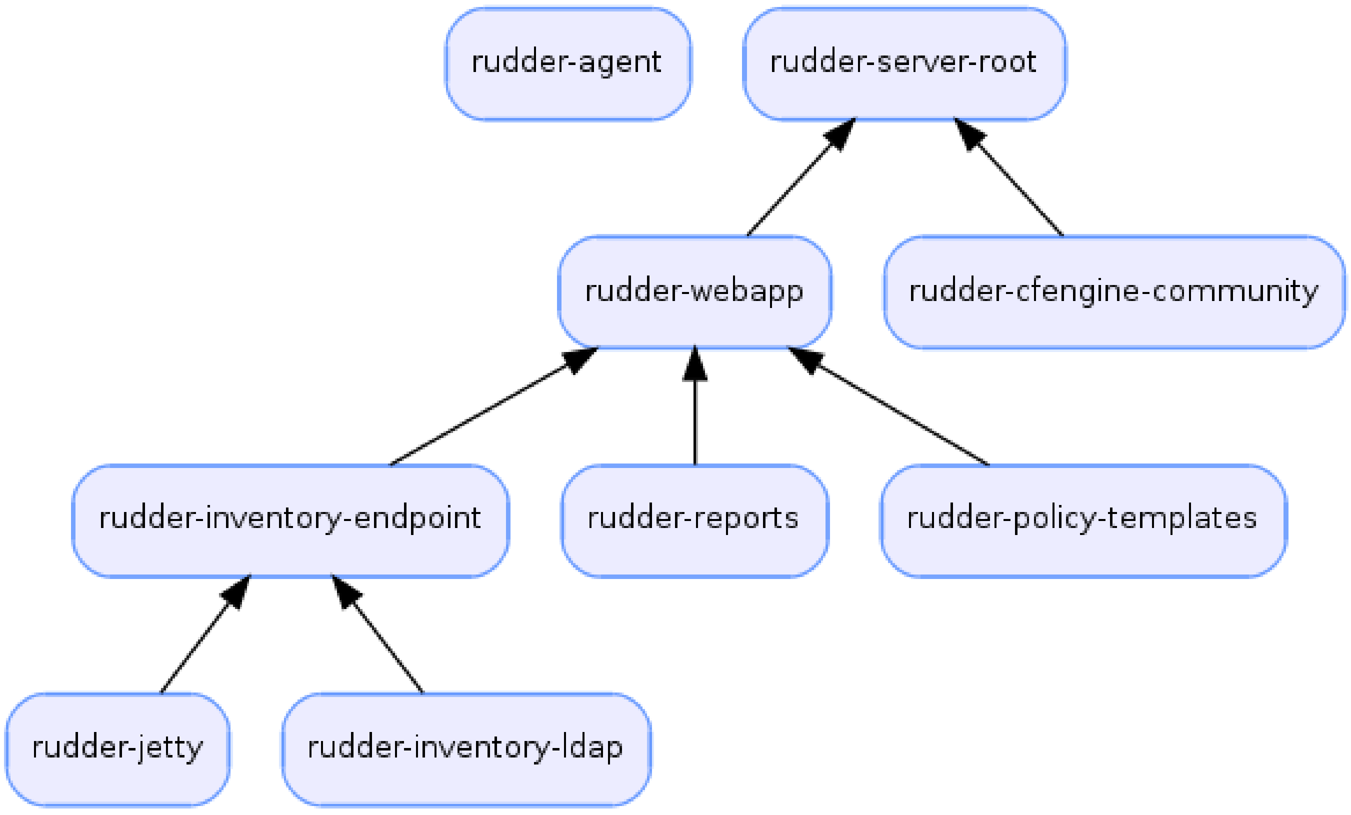


Figure 1.2: Rudder packages and their dependancies

`rudder-webapp` Package for the *Rudder* Web Application. It is the graphical interface for *Rudder*.

`rudder-inventory-endpoint` Package for the inventory reception service. It has no graphical interface. This service is using HTTP as transport protocol. It receives an parses the files sent by *FusionInventory* and insert the valuable data into the LDAP database.

`rudder-jetty` Application server for `rudder-webapp` and `rudder-inventory-endpoint`. Both packages are written in *Scala*. At compilation time, they are converted into `.war` files. They need to be run in an application server. *Jetty* is this application server. It depends on Oracle Java *JRE*.

`rudder-policy-templates` Package for the *Policy Templates*. They are installed in `/opt/rudder/share/policy-templates`. At runtime, the *Policy Templates* are copied into a *git* repository in `/var/rudder`. Therefore, the package depends on the `git` package.

`rudder-inventory-ldap` Package for the database containing the inventory and configuration informations for each pending and validated *Nodes*. This *LDAP* database is build upon *OpenLDAP* server. The *OpenLDAP* engine is contained in the package.

`rudder-reports` Package for the database containing the logs sent by each *Node* and the reports computed by *Rudder*. This is a *PostgreSQL* database using the *PostgreSQL* engine of the distribution. The package has a dependancy on the `postgresl` package, creates the database named `rudder` and installs the inialisation scripts for that database in `/opt/rudder/etc/postgresql/*.sql`.

`rudder-cfengine-community` Package for the *CFEngine* server. This server delivers to the *Nodes* the Applied Policies converted into *CFEngine* promises.

`rudder-server-root` Package to ease installation of every *Rudder* services. This package depends on all above packages. It also

- installs the *Rudder* configuration script:

```
/opt/rudder/bin/rudder-init.sh
```

- installs the initial promises for the Root Server in:

```
/opt/rudder/share/initial-promises/
```

- installs the init scripts (and associated `default` file):

```
/etc/init.d/rudder-server-root
```

- installs the logrotate configuration:

```
/etc/logrotate.d/rudder-server-root
```

`rudder-agent` One single package integrates everything needed for the Rudder Agent. It contains *CFEngine* Commmunity, *FusionInventory*, and the initial promises for a *Node*. It also contains an init script:

```
/etc/init.d/rudder-agent
```

The `rudder-agent` package depends on a few common libraries and utilities:

- `OpenSSL`
- `libpcre`
- `libdb` (4.6 on *Debian*)
- `uuidgen` (utility from `uuid-runtime package` on *Debian*)

1.6.4 Software dependencies and third party components

The *Rudder* Web application requires the installation of *Apache 2 httpd*, Oracle Java *6 JRE*, and *cURL*; the LDAP *Inventory* service needs *rsyslog* and the report service requires *PostgreSQL*.

When available, packages from your distribution are used. These packages are:

Apache The Apache Web server is used as a proxy to give HTTP access to the Web Application. It is also used to give writable WebDAV access for the inventory. The *Nodes* send their inventory to the WebDAV service, the inventory is stored in `/var/rudder/inventories/incoming`.

PostgreSQL The PostgreSQL database is used to store logs sent by the *Nodes* and reports generated by *Rudder*.

rsyslog and rsyslog-pgsql The rsyslog server is receiving the logs from the nodes and insert them into a PostgreSQL database. On SLES, the `rsyslog-pgsql` package is not part of the distribution, it can be downloaded alongside *Rudder* packages.

***Oracle Java* JRE** The *Java* runtime is needed by the Jetty application server. On *Debian*, the package from the distribution is used. On SLES, the package must be dowloaded from *Oracle* website.

curl This package is used to send inventory files from `/var/rudder/inventories/incoming` to the *Rudder* Endpoint.

git The package is not a dependency, but its installation is recommended. The running *Policy Templates* Library is maintained as a git repository in `/var/rudder/policy-templates`. It can be useful to have git installed on the system for maintenance purpose.

1.7 Configure the network

1.7.1 Mandatory flows

The following flows from the *Nodes* to the Rudder *Root Server* has to be allowed:

Port 5309, TCP *CFEngine* communication port, used to communicate the policies to the rudder nodes.

Port 80, TCP, for nodes HTTP communication port, used to send inventory and fetch the id of the Rudder *Server.*

Port 514, TCP Syslog port, used to centralize reports.

Open the following flow from the clients desktop to the Rudder *Root Server:*

Port 80, TCP, for users HTTP communication port, used by the users to access to the web interface.

1.7.2 Optional flows

These flows are used to add features to *Rudder*:

"Big red button" A button, on the right top side of every page of *Rudder* web interface, to command the emergency stop of the agents. This stop will be implicitly done in less than 10 minutes, or can be done immediately if the port 5309 TCP from the Rudder *Root Server* (or each relay server) is open to each nodes. This feature is detailed in the user documentation.

CFEngine Nova Managing *Windows* machines requires the commercial version of *CFEngine*, called *Nova*. It needs to open the port 5308 TCP from the *Node* to the *Rudder* Root Server.

1.7.3 DNS - Name resolution

Currently, *Rudder* relies on the *Node* declared hostnames to identify them. So it is required that each *Node* hostname can be resolved to its *IP address* that will be used to contact the Rudder *Server.* We are aware that it is far from being ideal in most cases (no DNS environement, private sub-networks, NAT, etc. . .), and we are currently working on an alternative solution.

If you do not have the wished name resolution, we advice that you should fill the *IP address* and hostname of the `/etc/hosts` file of the Rudder *Root Server.*

Similarly, each Rudder Node *must be able to resolve the* Rudder *Root Server* hostname given in the step described in Initial configuration of your Rudder Root Server.

Chapter 2

Install Rudder Server

This chapter covers the installation of a Rudder *Root Server*, from the specification of the underlying server, to the initial setup of the application.

2.1 Prepare Rudder Root Server Installation

The following chapter describes the steps to configure the Rudder *Root Server*.

Before all, you need to setup a server according to the server specifications. You should also configure the network. These topics are covered in the Architecture chapter.

Ideally, this machine should have Internet access, but this is not a strict requirement.

2.2 Install Rudder Root server on Debian

2.2.1 Add the main, contrib and non-free components

Ensure that the `apt` package manager is configured to use these three components, by checking that `/etc/apt/sources.list` contains the following lines:

```
deb http://ftp.fr.debian.org/debian/ squeeze main contrib non-free
deb http://security.debian.org/ squeeze/updates main contrib non-free
```

Tip
Your mirror may differ, `ftp.fr.debian.org` is only an example. Also, please adapt the distribution name if needed.
Your Linux distribution may differ too, `squeeze` could be replaced by `lenny`.
The *Rudder* Root server needs a compatible *Java* Runtime Environment to run.
On *Debian Squeeze* and *Debian Lenny*, the available package is *Oracle*™ *Java* 6 JRE, namely sun-java-6-jre, which is in the non-free component.
On *Debian* Wheezy and above, the available package is OpenJDK 7 JRE, namely openjdk-7-jre.
You can get the *Java* JRE 6 here : http://www.java.com

2.2.2 Update the system

Prior to beginning the installation of your Rudder *Server*, we recommend that you update your *Debian* system with the latest versions of available packages. Especially for *Debian* 5 (*Lenny*), since the release of *Debian* 6.0 (*Squeeze*), the signing key of packages repositories has changed. If you haven't already done it, you should force the upgrade of the `debian-archive-keyring` package to fetch the new key:

```
root@rudder-server:~# aptitude update
root@rudder-server:~# aptitude install debian-archive-keyring
root@rudder-server:~# aptitude update
root@rudder-server:~# aptitude safe-upgrade
```

2.2.3 Add the Rudder packages repository

To validate the content of the *Normation* repository, you should import the GPG key used to sign it:

```
root@rudder-server:~# apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 474A19E8
```

If your HTTP Keyserver Protocol (11371/tcp) is blocked you can use an alternate command:

```
root@rudder-server:~# wget --quiet -O- "http://keyserver.ubuntu.com/pks/lookup?op=get& ↩
    search=0x474A19E8" | sudo apt-key add -
```

Then add the URL of the *Normation* repository, by typing:

```
root@rudder-server:~# echo "deb http://www.rudder-project.org/apt-2.3/ $(lsb_release -cs)  ↩
    main" > /etc/apt/sources.list.d/rudder.list
```

Then, update your local package database to retrieve the list of packages available on our repository:

```
root@rudder-server:~# aptitude update
```

2.2.4 Install your Rudder Root Server

To begin the installation, you should simply install the `rudder-server-root` metapackage, which will install the required components:

```
root@rudder-server:~# aptitude install rudder-server-root
```

Rudder uses the *Oracle Java* virtual machine. To install it, you must accept its license.

2.3 Install Rudder Root server on SLES

2.3.1 Configure the package manager

Ensure that the `zypper` package manager is configured, and install the required packages: `rsyslog`, `rsyslog-pgsql` and Oracle Java *JRE*. `rsyslog` and `rsyslog-pgsql` are downloadable along *Rudder* and *Java* is available through *Oracle*'s website: http://www.java.com.

2.3.2 Update the system

Prior to beginning the installation of your Rudder *Server,* we recommend that you update your SLES system with the latest versions of available packages.

```
root@rudder-server:~# zypper up
```

2.3.3 Add the Rudder packages repository

Add the URL of the *Normation* repository, by typing the next command on a SLES 11:

```
root@rudder-server:~# zypper ar -n "Normation RPM Repositories" http://www.rudder-project. ↩
    org/rpm-2.3/SLES_11_SP1/ Normation
```

Or this one on a SLES 10:

```
root@rudder-server:~# zypper sa "http://www.rudder-project.org/rpm-2.3/SLES_10_SP3/"  ↩
    Normation
```

Then, update your local package database to retrieve the list of packages available on our repository:

```
root@rudder-server:~# zypper up
```

2.3.4 Install your Rudder Root Server

To begin the installation, you should simply install the `rudder-server-root` metapackage, which will install the required components:

```
root@rudder-server:~# zypper in rudder-server-root
```

Tip
If you want to manage the *Policy Templates* Library with *git* on a SLES based system, you should dowload the *SDK DVD* and install `git-core` using `yast2` or `zypper`, or get the RPM using another channel.

2.4 Files installed by the application.

`/etc` System-wide configuration files are stored here: init scripts, configuration for apache, logrotate and rsyslog.

`/opt/rudder` Non variable application files are stored here.

`/opt/rudder/etc` Configuration files for *Rudder* services are stored here.

`/var/log/rudder` Log files for *Rudder* services are stored here.

`/var/rudder` Variable data for *Rudder* services are stored here.

`/var/rudder/cfengine-community` Data for *CFEngine Community* are stored here.

`/var/rudder/policy-templates` *Policy Templates* are stored here.

`/var/cfengine` Data for *CFEngine Nova* are stored here.

`/usr/share/doc/rudder*` Documentation about *Rudder* packages.

2.5 Initial configuration of your Rudder Root Server

After the installation, you have to configure some system elements, by launching the following initialization script:

```
/opt/rudder/bin/rudder-init.sh
```

This script will ask you to fill in the following details:

Hostname The hostname that can be used by the client *Nodes* to reach the server. It is used to configure the web interface (so it will be the URL you'll use to access it), and to configure on the client *Node* how to reach the root server.

Allowed networks A list of IP networks authorized to connect to the server. We recommend that you specify all the networks of your infrastructure. The syntax is the standard network/mask notation, for instance `192.168.0.0/24` or `10.0.0.0/8`. To add several networks, first type the first network, then press the return key - the script will ask if you wish to add some more networks.

Server IP The *IP address* of the Rudder *Root Server* on which the *CFEngine* daemon should be contacted by all nodes. If your root server has only one IP address, you should nevertheless type it here.

Demo data Type "yes" if you wish to have the local database filed with demo data. It is usually not recommended if you wish to add your own *Nodes*.

Reset initial promises On an existing Rudder *Server,* you can remove all promises generated by *Rudder* and replace them by the standard initialisation promises. The major effect of this option is that every *Nodes* won't be able to fetch their promises until the next regeneration by *Rudder*.

Tip
In case of typing error, or if you wish to reconfigure these elements, you can execute this script again as many times as you want.

2.6 Validate the installation

Once all these steps have been completed, use your web browser to go to the URL given on the step described in the section about initial configuration.

You should see a loading, then a login screen. Only two demo accounts are configured, without any right restriction as of now.

Chapter 3

Install Rudder Agent

This chapter gives a general presentation of the Rudder Agent, and describes the different configuration steps to deploy the *Rudder* agent on the *Nodes* you wish to manage. Each Operating System has its own set of installation procedures.

The machines managed by *Rudder* are called *Nodes*, and can either be physical or virtual. For a machine to become a managed *Node*, you have to install the Rudder Agent on it. The *Node* will afterwards register itself on the server. And finally, the *Node* should be acknowledged in the Rudder *Server* interface to become a managed *Node*. For a more detailled description of the workflow, please refer to the Advanced Usage part of this documentation.

3.1 Components

This agent contains the following tools:

1. The community version of *CFEngine*, a powerful open source configuration management tool.
2. *FusionInventory*, an inventory software.
3. An initial configuration set for the agent, to bootstrap the Rudder *Root Server* access.

These components are recognized for their reliability and minimal impact on performances. Our tests showed their memory consumption is usually under 10 MB of RAM during their execution. So you can safely install them on your servers.

We grouped all these tools in one package, to ease the Rudder Agent installation.

To get the list of supported Operating systems.please refer to the list of supported Operating Systems for the Nodes.

3.2 Install Rudder Agent on Debian or Ubuntu

3.2.1 Add the Rudder packages repository

To validate the content of the *Normation* repository, you should import the GPG key used to sign it:

```
root@rudder-server:~# apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 474A19E8
```

if your HTTP Keyserver Protocol (11371/tcp) is blocked you can use an alternate command:

```
root@rudder-server:~# wget --quiet -O- "http://keyserver.ubuntu.com/pks/lookup?op=get& ↩
    search=0x474A19E8" | sudo apt-key add -
```

Then add the URL of the *Normation* repository, by typing the next command:

```
root@rudder-server:~# echo "deb http://www.rudder-project.org/apt-2.3/ $(lsb_release -cs)  ↩
    main contrib non-free" > /etc/apt/sources.list.d/rudder.list
```

Then, update your local package database to retrieve the list of packages available on our repository:

```
root@rudder-server:~# aptitude update
```

3.2.2 Install your Rudder Agent

To begin the installation, you should simply install the rudder-agent

```
root@rudder-server:~# aptitude install rudder-agent
```

Type in the *IP address* of the Rudder *Root Server* in the following file

```
echo *root_server_IP_address* > /var/rudder/cfengine-community/policy_server.dat
```

Tip
We advise you to use the `IP address` of the Rudder *Root Server.* The DNS name of this server can also be accepted if you have a complete DNS infrastructure matching the IP of the *Nodes* with their hostnames.

Finally, start the agent:

```
/etc/init.d/rudder-agent start
```

3.3 Install Rudder Agent on RedHat or CentOS

Download the package applicable to your version of *RedHat/CentOS* and to its architecture on

```
http://www.rudder-project.org/rpm-2.3/RHEL/RPMS/
```

Install the package:

```
rpm -Uhv rudder-agent-2.3.0-1.EL.5.x86_64.rpm
```

Type in the *IP address* of the Rudder *Root Server* in the following file

```
echo *root_server_IP_address* > /var/rudder/cfengine-community/policy_server.dat
```

Tip
We advise you to use the `IP address` of the Rudder *Root Server.* The DNS name of this server can also be accepted if you have a complete DNS infrastructure matching the IP of the *Nodes* with their hostnames.

Finally, start the agent:

```
service rudder-agent start
```

3.4 Install Rudder Agent on SuSE

3.4.1 Add the Rudder packages repository

Add the URL of the *Normation* repository, by typing the next command on a SLES 11 node:

```
root@rudder-node:~# zypper ar -n "Rudder RPM Repositories" http://www.rudder-project.org/ ↩
    rpm-2.3/SLES_11_SP1/ Rudder
```

Or this one on a SLES 10 node:

```
root@rudder-node:~# zypper sa "http://www.rudder-project.org/rpm-2.3/SLES_10_SP3/" Rudder
```

Then, update your local package database to retrieve the list of packages available on our repository:

```
root@rudder-node:~# zypper ref
```

3.4.2 Install your Rudder Agent

To begin the installation, you should simply install the rudder-agent:

```
root@rudder-server:~# zypper install rudder-agent
```

Type in the *IP address* of the Rudder *Root Server* in the following file

```
echo *root_server_IP_address* > /var/rudder/cfengine-community/policy_server.dat
```

Tip
We advise you to use the `IP address` of the Rudder *Root Server.* The DNS name of this server can also be accepted if you have a complete DNS infrastructure matching the IP of the *Nodes* with their hostnames.

Finally, start the agent:

```
service rudder-agent start
```

3.5 Validation

Several minutes after the start of the agent, a new *Node* should be pending in the *Rudder* web interface.

You will be able to browse its inventory, and accept it to manage its configuration with *Rudder*.

You may force the agent execution by issuing the following command :

```
/var/rudder/cfengine-community/bin/cf-agent -KI
```

Chapter 4

Rudder Web Interface

This chapter is a general presentation of the *Rudder* Web Interface. You will find how to authenticate in the application, a description of the design of the screen, and some explanations about usage of common user interface items like the search fields and the reporting screens.

4.1 Authentication

When accessing the *Rudder* web interface, a login / password is required. The default accounts are:

- Login: jon.doe, password: secret
- Login: alex.bar, password: secret2

You can change the user accounts by following the User management procedure.

4.2 Presentation of Rudder Web Interface

The web interface is organised according to the concepts described earlier. It is divided in three logical parts: Asset Management, Configuration Management and Administration.

4.2.1 Rudder Home

The home page summarizes the content of the other parts and provides quick links for the most common actions.

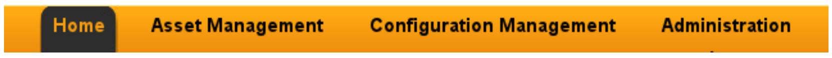


Figure 4.1: Home menu

4.2.2 Asset Management

In the Asset Management section, you will find the validation tool for new *Nodes*, a search engine for validated *Nodes*, and the management tool for groups of *Nodes*.

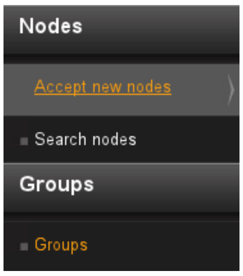


Figure 4.2: Asset Management menu

4.2.3 Configuration Management

In the Configuration Management section, you can select the *Policy Templates*, configure the *Policy Instances* and manage the *Configuration Rules*.

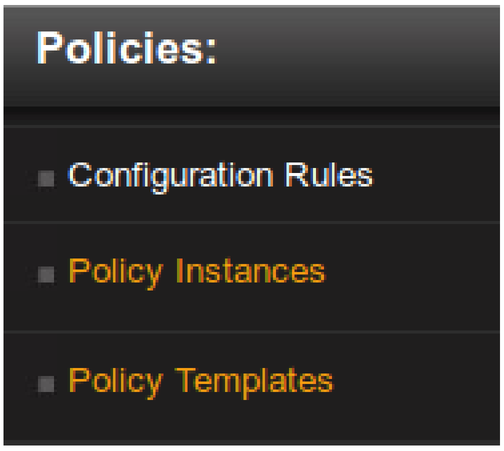


Figure 4.3: Configuration Management menu

4.2.4 Administration

The Administration section provides some general settings: you can setup the available networks for the Policy Server, view the event logs and manage your plugin collection.

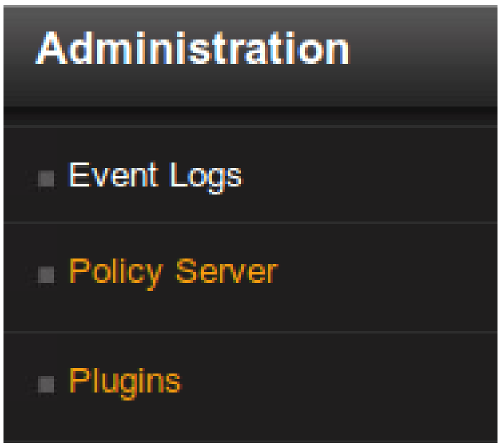


Figure 4.4: Administration menu

4.3 Units supported as search parameters

Some parameters for the advanced search tool allow using units. For example, in the search criterion for RAM size, you can type `512MB` instead of a value in bytes. This paragraph describes supported units by parameter type.

4.3.1 Bytes and multiples

All criteria using a memory size (RAM, hard disk capacity, etc) is by default expected in bytes. If no other unit is specified, all values will be assumed to be in bytes.

4.3.2 Convenience notation

All memory sizes can be written using spaces or underscores (_) to make the numbers easier to read. Numbers must begin with a digit. For example, the following numbers are all valid and all worth `1234`:

```
1234
1 234
1_234
1234_
```

The following number is not valid:

```
_1234
```

4.3.3 Supported units

Units used are non binary units, and a mutliplication factor of 1024 is applied between each unit. Units are case insensitive. Therefore, `Mb` is identical to `mB` or `mb` or `MB`.

In detail, the following units are supported (provided in lower case, see above):

Notation	Alternate	Value
`b`	`o`	bytes (equivalent to not specifying a unit)
`kb`	`ko`	1024 bytes
`mb`	`mo`	1024^2 bytes
`gb`	`go`	1024^3 bytes
`tb`	`to`	1024^4 bytes
`pb`	`po`	1024^5 bytes
`eb`	`eo`	1024^6 bytes
`zb`	`zo`	1024^7 bytes
`yb`	`yo`	1024^8 bytes

Table 4.1: Units supported by Rudder search engine

Chapter 5

Asset Management

5.1 Asset Inventory

Rudder integrates an asset inventory tool which harvest useful informations about the nodes. These informations are used by *Rudder* to handle the nodes, and you can use the inventory informations for Configuration Management purpose: search *Nodes*, create Groups of *Nodes*, determine some configuration management variables.

In the *Rudder* Web Interface, each time you see a *Node* name, you can click on it and display the collection of informations about this *Node*. The inventory is organized as following: first tab is a *summary* of administrative informations about the *Node*; other tabs are specialized for *hardware*, *network* interfaces, and *software* for every *Nodes*; tabs for *reports* and *logs* are added on *Rudder* managed *Nodes*.

The Node *Summary* presents administrative informations like the *Node Hostname*, *Operating System*, Rudder *Client name*, Rudder *ID* and *Date* when the inventory was *last received*. When the *Node* has been validated, some more informations are displayed like the *Node Name* and the *Date first accepted in* Rudder.

The *hardware* informations are organized as following: *General*, *File systems*, *Bios*, *Controllers*, *Memory*, *Port*, *Processor*, *Slot*, *Sound*, *Storage*, *Video*.

Network connexions are detailled as following: *Name* of the interface on the system, IP address, *Network Mask*, usage of *DHCP* or static configuration, *MAC address*, *Type* of connexion, *Speed* of the connexion and *Status*.

And finally, you get the list of every packaged *software* present on the system, including version and description.

On *Nodes* managed by *Rudder*, the *Reports* tab displays informations about the status of latest run of Rudder Agent*,* whereas the *Logs* tab displays informations about changes for the *Node*.

5.2 Accept new Nodes

At the starting point, the Rudder *Server* does'nt know anything about the *Nodes*. After the installation of the Rudder Agent*,* each *Node* register itself to the Rudder *Server,* and sends a first inventory. Every new *Node* must be manually validated in the *Rudder* Web Interface to become part of *Rudder* Managed *Nodes*. This task is performed in the **Asset Management > Accept new *Nodes*** section of the application. You can select *Nodes* waiting for an approval, and determine whether you consider them as valid or not. Click on each *Node* name to display the extended inventory. Click on the magnifying glass icon to display the policies which will be applied after the validation.

Example 5.1 Accept the new Node `debian-node.rudder-project.org`

1. Install and configure the Rudder Agent on the new *Node* `debian-node.rudder-project.org`
2. Wait a few minutes for the first run of the Rudder Agent.
3. Navigate to **Asset Management > Accept new *Nodes***.
4. Select the new *Node* in the list.
5. Validate the *Node*.
6. The *Node* is now integrated in *Rudder*, you can search it using the search tools.

5.3 Search Nodes

You can navigate to **Asset Management > Search *Nodes*** to display information about the *Nodes* which have been already validated, and are managed by *Rudder*.

5.3.1 Quick Search

The easiest search tool is the Quick search: type in the search field the first letters of the Rudder *ID*, *Reference*, or *Hostname*; choose the accurate *Node* in the autocompletion list; validate and look at the *Node* informations. This search tool can be very useful to help you create a new search in the Advanced Search.

Example 5.2 Quick search the Node called `debian-node`

Assuming you have one managed *Node* called `debian-node.rudder-project.org`, which ID in *Rudder* is `d06b1c6-c-f59b-4e5e-8049-d55f769ac33f`.

1. Type in the Quick Search field the `de` or `d0`.
2. Autocompletion will propose you this *Node*: `debian-node.rudder-project.org -- d06b1c6c-f59b-4e5-e-8049-d55f769ac33f [d06b1c6c-f59b-4e5e-8049-d55f769ac33f].`

5.3.2 Advanced Search

In the Advanced Search tool, you create complex research based on Asset *Inventory* informations. The benefit of the Advanced Search tool is to save the query and create a Group of *Nodes* based on the search criteria.

- 1. Select a field

The selection of the field upon which the criteria will apply is a two step process. The list of fields is not displayed unordered and extensively. Fields have been grouped in the same way they are displayed when you look at information about a *Node*. First you choose among these groups: Node, *Network Interface*, *Filesystem*, *Machine*, *RAM*, *Storage*, *BIOS*, *Controller*, *Port*, *Processor*, *Sound Card*, *Video Card*, *Software*; then you choose among the list of fields concerning this theme.

- 2. Select the matching rule

The matching rule can be selected between following possibilities: *Is defined*, *Is not defined*, = or ≠ followed by the term you are searching for presence or absence. Depending on the field, the list of searchable terms is either an free text field, either the list of available terms.

- 3. Add another rule

You can select only one term for each matching rule. If you want to create more complex search, then you can add another rule using the + icon. All rules are using the same operand, either *AND* or *OR*. More complex searches mixing *AND* and *OR* operands are not available at the moment.

Example 5.3 Advanced search for Linux Nodes with `ssh`.

Assuming you want to search every Linux *Nodes* having `ssh` installed. You will create this 2 lines request:

1. Operator: `AND`.
2. First search line: `Node`, `Operating System`, `=`, `Linux`.
3. Second search line: `Software`, `Name`, `=`, `ssh`.

5.4 Group of Nodes

You can create Group of *Nodes* based on search criteria to ease attribution of Rules in Configuration Management. The creation of groups can be done from the `Asset Management > Search Nodes` page, or directly from the Groups list in `Asset Management > Groups`. A group can be either Dynamic or Static.

Dynamic group Group of *Nodes* based on search criteria. The search is replayed every time the group is queried. The list will always contain the nodes that match the criteria, even if the data nodes have changed since the group was created.

Static group Group of *Nodes* based on search criteria. The search is performed once and the resulting list of *Nodes* is stored. Once declared, the list of nodes will not change, except manual change.

Chapter 6

Configuration Management

6.1 Policy Templates

6.1.1 Concepts

A *Policy Template* defines a set of operations and configurations to reach the desired behaviour. This includes the initial set-up, but also a regular check on the parameters, and automatic repairs (when possible).

All the *Policy Templates* are built with the possibility to change only part of a service configuration: each parameter may be either active, either set on the "Don't change" value, that will let the default values or in place. This allows for a progressive deployment of the configuration management.

Finally, the *Policy Templates* will generate a set of reports which are sent to the Rudder *Root Server,* which will let you analyse the percentage of compliance of your policies, and soon, detailed reports on their application.

6.1.2 Manage the Policy Templates

The *Policy Templates* shipped with *Rudder* are presented in a library that you can reorganize in **Administration > *Policy Template* Library Management**. The library is organized in two parts: the available *Policy Templates*, and the selection made by the user.

Reference *Policy Template* Library This is an organized list of every available *Policy Templates*. This list can't be modified: every changes made by an user will be applied to the User Policy Template Library.

User *Policy Template* Library This is an organized list of the *Policy Templates* selected and modified by the user. By default this list is the same as the Reference *Policy Template* Library. *Policy Templates* can be disabled or deleted, and then activated again with a simple drag and drop. Categories can be reorganised according to the desired taxonomy. A *Policy Template* can appear only once in the Library.

Tip
The current version of *Rudder* has only an handful of ***Policy Templates***. We are aware that it considerably limits the use of the application, but we choose to hold back other *Policy Templates* that did not, from our point of view, have the sufficient quality. In the future, there will be some upgrades including more *Policy Templates*.

Warning
The creation of new *Policy Templates* is not covered by the Web interface. This is an advanced task which is currently not covered by this guide.

6.1.3 Available Policy Templates

6.1.3.1 Application management

Apache 2 HTTP server This *Policy Template* will configure the Apache HTTP server and ensure it is running. It will ensure the "apache2" package is installed (via the appropriate packaging tool for each OS), ensure the service is running and start it if not and ensure the service is configured to run on initial system startup. Configuration will create a rudder vhost file.

APT package manager configuration Configure the apt-get and aptitude tools on GNU/Linux *Debian* and *Ubuntu*, especially the source repositories.

OpenVPN client This *Policy Template* will configure the OpenVPN client service and ensure it is running. It will ensure the "openvpn" package is installed (via the appropriate packaging tool for each OS), ensure the service is running and start it if not and ensure the service is configured to run on initial system startup. Configuration will create a rudder.conf file. As of this version, only the PSK peer identification method is supported, please use the "Download File" *Policy Template* to distribute the secret key.

Package management for *Debian* / *Ubuntu* / APT based systems Install, update or delete packages, automatically and consistently on GNU/Linux *Debian* and *Ubuntu*.

Package management for RHEL / *CentOS* / RPM based systems Install, update or delete packages, automatically and consistently on GNU/Linux *CentOS* and *RedHat*.

6.1.3.2 Distributing files

Copy a file Copy a file on the machine

Distribute ssh keys Distribute ssh keys on servers

Download a file Download a file for a standard URL (HTTP/FTP), and set permissions on the downloaded file.

6.1.3.3 File state configuration

Set the permissions of files Set the permissions of files

6.1.3.4 System settings: Miscellaneous

Time settings Set up the time zone, the NTP server, and the frequency of time synchronisation to the hardware clock. Also ensures that the NTP service is installed and started.

6.1.3.5 System settings: Networking

Hosts settings Configure the contents of the hosts filed on any operating system (Linux and *Windows*).

IPv4 routing management Control IPv4 routing on any system (Linux and *Windows*), with four possible actions: add, delete (changes will be made), check presence or check absence (a warning may be returned, but no changes will be made) for a given route.

Name resolution Set up the *IP address* of the DNS server name, and the default search domain.

NFS Server Configure a NFS server

6.1.3.6 System settings: Process

Process Management Enforce defined parameters on system processes

6.1.3.7 System settings: Remote access

OpenSSH server Install and set up the SSH service on Linux nodes. Many parameters are available.

6.1.3.8 System settings: User management

Group management This *Policy Template* manages the target host(s) groups. It will ensure that the defined groups are present on the system.

Sudo utility configuration This *Policy Template* configures the sudo utility. It will ensure that the defined rights for given users and groups are correctly defined.

User management Control users on any system (Linux and *Windows*), including passwords, with four possible actions: add, delete (changes will be made), check presence or check absence (a warning may be returned, but no changes will be made) for a given user.

6.2 Policy Instances

Once you have selected and organized your *Policy Templates*, you can create your configurations in the **Configuration Management > *Policy Instances*** section.

Policy Instance This is an instance of a *Policy Template,* which allows to set values for the parameters of the latter. Each *Policy Instance* can have an unique name. A Policy Instance should be completed with a short and a long description, and a collection of parameters for the variables defined by the *Policy Template.*

The screen is divided in three parts:

- on the left, your list of *Policy Templates* and *Policy Instances*,
- on the right the description of the selected *Policy Template* or Policy Instance.
- at the bottom, the configuration items of the selected *Policy Instance.*

Click on the name of a *Policy Template* or to see its description.

Click on the name of a *Policy Instance* to see the Policy Summary containing the description of the *Policy Template* its derived from, and the configuration items of the *Policy Instance.*

Example 6.1 Create a Policy Instance for Name resolution

Use the *Policy Template Name resolution* to create a new *Policy Instance* called `Google DNS Servers`, and shortly described as *Use Google DNS Server*. Check in the options *Set nameservers* and *Set DNS search suffix*. Set the value of the variable *DNS resolver* to `8.8.8.8` and of *Domain search suffix* according to your organization, like `rudder-project.org`.

Manual escaping of value

The values you are inserting within the *Policy Instance* fields need to be manually escaped most of the time, or else you will have deployment error. Every quote (") need to be prefixed by a backslash (\"), and backslash should be prefixed by another backslash (\\ ). Indeed, *CFEngine* will consider a non-prefixed quote as the end of value, and it considers \\ to mean \

However, there are no other escape character (\n will not be considered as a new line, etc).

6.3 Configuration rules

Configuration Rule It is the application of a policy to a group of nodes. It is the glue between both Asset Management and Configuration Management parts of the application.

When a *Configuration Rule* is created or modified, the promises for the target nodes are generated. Rudder *computes all the promises each nodes must have, and makes them available for the nodes. This process can take up to several minutes, depending on the number of managed nodes and the Policy Server* configuration. During this time, the "Regenerate now" button is replaced by a moving bar and a message stating "Generating configuration rules". You can also press the "Regenerate now" button on the top of the interface if you feel the generated promises should be modified (for instance, if you changed the configuration of *Rudder*) === Compliance

A *Policy Instance* contains one or multiple components. Each component generates one ore multiple reports, based on the number of keys in this component. For example, for a Sudoers *Policy Instance,* each user is a key. These states are available in reports:

Success The system is already in the desired state. No change is needed. Conformity is gained.

Repaired The system was not in the desired state. *Rudder* applied some change and repaired what was not correct. Now the system is in the desired state. Conformity is gained.

Error The system is not in the desired state. *Rudder* couldn't repair the system.

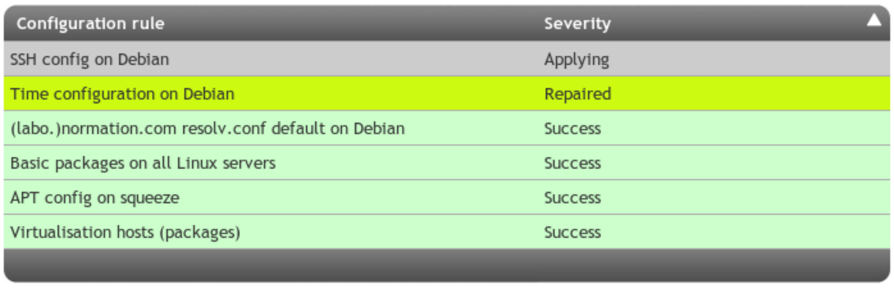
Applying When a *Policy Instance* is applied, *Rudder* waits during 10 minutes for a report. During this period, the *Policy Instance* is said *Applying*.

No answer The system didn't sent any reports. *Rudder* waited for 10 minutes and no report was received.

A *Policy Instance* has gained conformity on a *Node* is every reports for each components, for each key, are in *Success* state. This is the only condition.

Based on these facts, the compliance of a *Configuration Rule* is calculated like this :

Number of *Nodes* for which conformity is reached for every *Policy Instance* of the *Configuration Rule* / Total number of *Nodes* on which the *Configuration Rule* has been applied



Configuration rule	Severity
SSH config on Debian	Applying
Time configuration on Debian	Repaired
(labo.)normation.com resolv.conf default on Debian	Success
Basic packages on all Linux servers	Success
APT config on squeeze	Success
Virtualisation hosts (packages)	Success

Figure 6.1: Reports

Chapter 7

Administration

This chapter covers basic administration task of *Rudder* services like configuring some parameters of the *Rudder* policy server, reading the services log, and starting, stopping or restarting *Rudder* services. === Basic administration of *Rudder* services

This chapter covers basic administration task of *Rudder* services like configuring some parameters of the *Rudder* policy server, reading the services logg, and start, stop or restart *Rudder* services.

7.1 Policy Server Management

The **Administartion > Policy Server Management** section sum-up information about *Rudder* policy server and its parameters.

7.1.1 Configure allowed networks

Here you can configure the networks from which nodes are allowed to connect to *Rudder* policy server to get their updated configuration rules.

You can add as many network as you want, the expected format is: `networkip/mask`, for example `42.42.0.0/16`.

7.2 View Event Logs

The **Administration > View Event Logs** section allows to see last events logged in *Rudder*.

7.3 Plugin Management

Rudder is an extensible software. The **Administration > Plugin Management** section sum-up information about loaded plugins and their configuration.

7.4 Restart the agent of the node

To restart the Rudder Agent, use following command on a node:

```
/etc/init.d/rudder-agent restart
```

Tip
This command can take more than one minute to restart the *CFEngine* daemon. This is not a bug, but an internal protection system of *CFEngine*.

7.5 Restart the root rudder service

7.5.1 Restart everything

You can restart all components of the Rudder *Root Server* at once:

```
/etc/init.d/rudder-server-root restart
```

7.5.2 Restart only one component

Here is the list of the components of the root server with a brief description of their role, and the command to restart them:

***CFEngine* server** Distribute the *CFEngine* configuration to the nodes.

```
/etc/init.d/cfengine-community restart
```

Web server application Execute the web interface and the server that handles the new inventories.

```
/etc/init.d/jetty restart
```

Web server front-end Handle the connection to the Web interface, the received inventories and the sharing of the UUID Rudder *Root Server*.

```
/etc/init.d/apache2 restart
```

LDAP server Store the inventories and the *Node* configurations.

```
/etc/init.d/slapd restart
```

SQL server Store the received reports from the nodes.

```
/etc/init.d/postgresql* restart
```

7.6 Policy Template upgrade

New versions of the *Policy Template* library are made available as packages, named rudder-policy-templates, for the 2.3 version of *Rudder*. Many bug fixes and new *Policy Templates* are added all the time. To benefit from these, we recommend you upgrade your *Policy Template* library from time to time.

Updates are available from rudder-project.org, as standard OS package downloads. Please note that nightly builds are also available, and may provide the most up to date set of *Policy Templates*. See http://www.rudder-project.org/foswiki/Download/ for full details.

When you upgrade the *Rudder Policy Templates* packages to a new version, a new version of the *Policy Template* library is installed to /opt/rudder/share/policy-templates.

The *Policy Template* library used by *Rudder* is managed using a GIT tree, located in /var/rudder/configuration-repository/policy-templates. Therefore, you can not simply copy the files from /opt/rudder/share/policy-templates to *Rudder*'s storage, you also have to follow this simple procedure:

Tip
Please make sure that any changes you make are on a new version of a *Policy Template,* or you are likely to have your changes replaced by the reference library! Of course, GIT will keep history if your modifications are already commited but this would be an annoyance.

- Jump to the *Rudder Policy Template* tree

```
cd /var/rudder/configuration-repository/policy-templates
```

- Copy the reference *Policy Template* library to your local tree

```
cp -a /opt/rudder/share/policy-templates/* .
```

- Update the GIT repository to match the new tree state

```
git commit -am "Upgraded the Policy Template library (by $USER)"
```

- Finally, return to the web interface and go the Configuration Management menu, then click on the *Policy Templates* menu item on the left. In the screen that appears, click the "Reload" button next to "You can load the last available version of the policy template library" at the top of the screen. === Usecases

This chapter gives a few examples for using *Rudder*. We have no doubt that you'll have your own ideas, that we're impatient to hear about…

7.6.1 Dynamic groups by operating system

Create dynamic groups for each operating system you administer, so that you can apply specific policies to each type of OS. When new nodes are added to *Rudder*, these policies will automatically be enforced upon them.

7.6.2 Library of preventive policies

Why not create policies for emergency situations in advance? You can then put your IT infrastructure in "panic" mode in just a few clicks.

For example, using the provided *Policy Templates*, you could create a Name resolution policy to use your own internal DNS servers for normal situations, and a second, alternative policy, to use Google's public DNS servers, in case your internal DNS servers are no longer available.

7.6.3 Standardizing configurations

You certainly have your own best practices (let's call them good habits) for setting up your SSH servers.

But is that configuration the same on all your servers? Enforce the settings your really want using an OpenSSH server policy and apply it to all your Linux servers. SSH servers can then be stopped or reconfigured manually many times, *Rudder* will always restore your preferred settings and restart the SSH server in less than 5 minutes. == Advanced usage

This chapter describe advanced usage of *Rudder*.

7.7 User management

Change the users authorized to connect to the application

7.7.1 Configuration of the users using a XML file

7.7.1.1 Generality and uses of clear text password

The credentials of a user are defined in the XML file `/opt/rudder/etc/rudder-users.xml`. This file expects the following format:

```
<authentication>
  <user name="jon.doe" password="secret"/>
  <user name="alex.bar" password="secret2"/>
</authentication>
```

The name and password attributes are mandatory (non empty) for the user tags. Only these attributes are recognized.

Every modification of this file should be followed by a restart of the *Rudder* web application to be taken into account:

```
/etc/init.d/jetty restart
```

7.7.1.2 Use of hashed passwords

The authentication tag may have the hash attribute. If defined, the password will be stored as hashes.

The algorithm used to create the hash (and verify it during authentication) depend on the value of the hash attribute. The possible values, the corresponding algorithm and the Linux shell command need to obtain the hash of the "secret" password for this algorithm are listed here:

Value	**Algorithm**	**Linux command to hash the password**
"md5"	MD5	`read mypass; echo -n $mypass | md5sum`
"sha" or "sha1"	SHA one	`read mypass; echo -n $mypass | shasum`
"sha256" or "sha-256"	SHA, 256 bytes	`read mypass; echo -n $mypass | sha256sum`
"sha512" or "sha-512"	SHA, 512 bytes	`read mypass; echo -n $mypass | sha512sum`

Table 7.1: Hashed passwords algorithms list

When using the suggested commands to hash a password, you must enter the command, then type your password, and hit return. The hash will then be displayed in your terminal. This avoids storing the password in your shell history.

Here is an example of authentication file with hashed password:

```
<authentication hash="sha">
  <user name="jon.doe"  password="e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4"/>
  <user name="alex.bar" password="c636e8e238fd7af97e2e500f8c6f0f4c0bedafb0"/>
</authentication>
```

7.7.2 Going further

Rudder aims at integrating with your IT system transparently, so it can't force its own authentication system.

To meet this need, *Rudder* relies on the modular authentication system Spring Security that allows to easily integrate with databases, LDAP directory, or an entreprise SSO like CAS, OpenID or SPNEGO. The documentation for this integration is not yet available, but don't hesitate to reach us on this topic.

7.8 Password management

You might want to change the default passwords used in *Rudder*'s managed daemons for evident security reasons.

7.8.1 Configuration of the postgres database password

You will have to adjust the postgres database and the rudder-web.properties file.

Here is a semi-automated procedure:

- Generate a decently fair password. You can use an arbitrary one too.

```
PASS=`dd if=/dev/urandom count=128 bs=1 2>&1 | md5sum | cut -b-12`
```

- Update the Postgres database user

```
su - postgres -c "psql -q -c \"ALTER USER blah WITH PASSWORD '$PASS'\""
```

- Insert the password in the rudder-web.properties file

```
sed -i "s%^rudder.jdbc.password.*$%rudder.jdbc.password=$PASS%" /opt/rudder/etc/rudder-web. ↩
    properties
```

7.8.2 Configuration of the OpenLDAP manager password

You will have to adjust the OpenLDAP and the rudder-web.properties file.

Here is a semi-automated procedure:

- Generate a decently fair password. You can use an arbitrary one too.

```
PASS=`dd if=/dev/urandom count=128 bs=1 2>&1 | md5sum | cut -b-12`
```

- Update the password in the slapd configuration

```
HASHPASS=`/opt/rudder/sbin/slappasswd -s $PASS`
sed -i "s%^rootpw.*$%rootpw          $HASHPASS%" /opt/rudder/etc/openldap/slapd.conf
```

- Update the password in the rudder-web.properties file

```
sed -i "s%^ldap.authpw.*$%ldap.authpw=$PASS%" /opt/rudder/etc/rudder-web.properties
```

7.8.3 Configuration of the WebDAV access password

This time, the procedure is a bit more tricky, as you will have to update the *Policy Template* library as well as a configuration file.

Here is a semi-automated procedure:

- Generate a decently fair password. You can use an arbitrary one too.

```
PASS=`dd if=/dev/urandom count=128 bs=1 2>&1 | md5sum | cut -b-12`
```

- Update the password in the apache htaccess file

Tip
On some systems, especially *SuSE* ones, htpasswd is called as "htpasswd2"

```
htpasswd -b /opt/rudder/etc/htpasswd-webdav rudder $PASS
```

- Update the password in *Rudder*'s system *Policy Templates*

```
cd /var/rudder/configuration-repository/policy-templates/system/common/1.0/
sed -i "s%^.*davpw.*$%   \"davpw\" string => \"$PASS\"\;%" site.st
git commit -m "Updated the rudder WebDAV access password" site.st
```

- Update the *Rudder Policy Instances* by either reloading them in the web interface (in the "Configuration Management/*Policy Templates*" tab) or restarting jetty (NOT recommended) === Reinitialize policies for a *Node*

To reinitialize the policies for a *Node*, delete the local copy of the Applied Policies fetched from the Rudder *Server,* and create a new local copy of the initial promises.

```
root@node:~# rm -rf /var/rudder/cfengine-community/inputs/*
root@node:~# cp -a /opt/rudder/share/initial-promises/* /var/rudder/cfengine-community/ ↩
    inputs/
```

At next run of the Rudder Agent (it runs every five minuts), the initial promises will be used.

Caution
Use this procedure with caution: the *Applied Polici*es of a *Node* should never get broken, unless some major change has occured on the *Rudder* infrastructure, like a full reinstallation of the Rudder *Server.*

7.9 Optimize Postgres

The default (out-of-the-box) configuration of Postgres is really not compliant for large (or normal) machines. It uses a really small amount of memory

The location of the file is:

1. /etc/postgresql/8.x/main/postgresql.conf
2. /var/lib/pgsql/data/postgresql.conf (on a *SuSE* system)

7.9.1 Suggested values on a large node

7.9.1.1 Amount of System V shared memory

A reasonable starting value for shared_buffers is 1/4 of the memory in your system:

1. shared_buffers = 1GB

Note
Note : you may need to set the proper amount on the system

```
$ sysctl -w kernel.shmmax=1073741824
```

Ref http://www.postgresql.org/docs/8.4/interactive/kernel-resources.html#SYSVIPC

7.9.1.2 Memory for complex operations

Complex query:

1. work_mem = 24MB
2. max_stack_depth = 4MB

Complex maintenance (index, vacuum):

1. maintenance_work_mem = 240MB

7.9.1.3 Write ahead log

Size of the write ahead log:

1. wal_buffers = 4MB

7.9.1.4 Query planner

Gives hint to the query planner about the size of disk cache
Setting effective_cache_size to 1/2 of total memory would be a normal conservative setting:

1. effective_cache_size = 1024MB

7.9.2 Suggested values on a not so large node

1. shared_buffers = 128MB
2. work_mem = 8MB
3. max_stack_depth = 3MB
4. maintenance_work_mem = 64MB
5. wal_buffers = 1MB
6. effective_cache_size = 128MB === Rudder Agent workflow

In this chapter, we will have a more detailled view of the Rudder Agent workflow. What files and processes are created or modified at the installation of the Rudder Agent*?* What is happening when a new *Node* is created? What are the recurrent tasks performed by the *Rudder Agent*? *How does the* Rudder *Server* handle the requests coming from the Rudder Agent*? The* Rudder Agent workflow schema summarizes the process that will be described in the next pages.

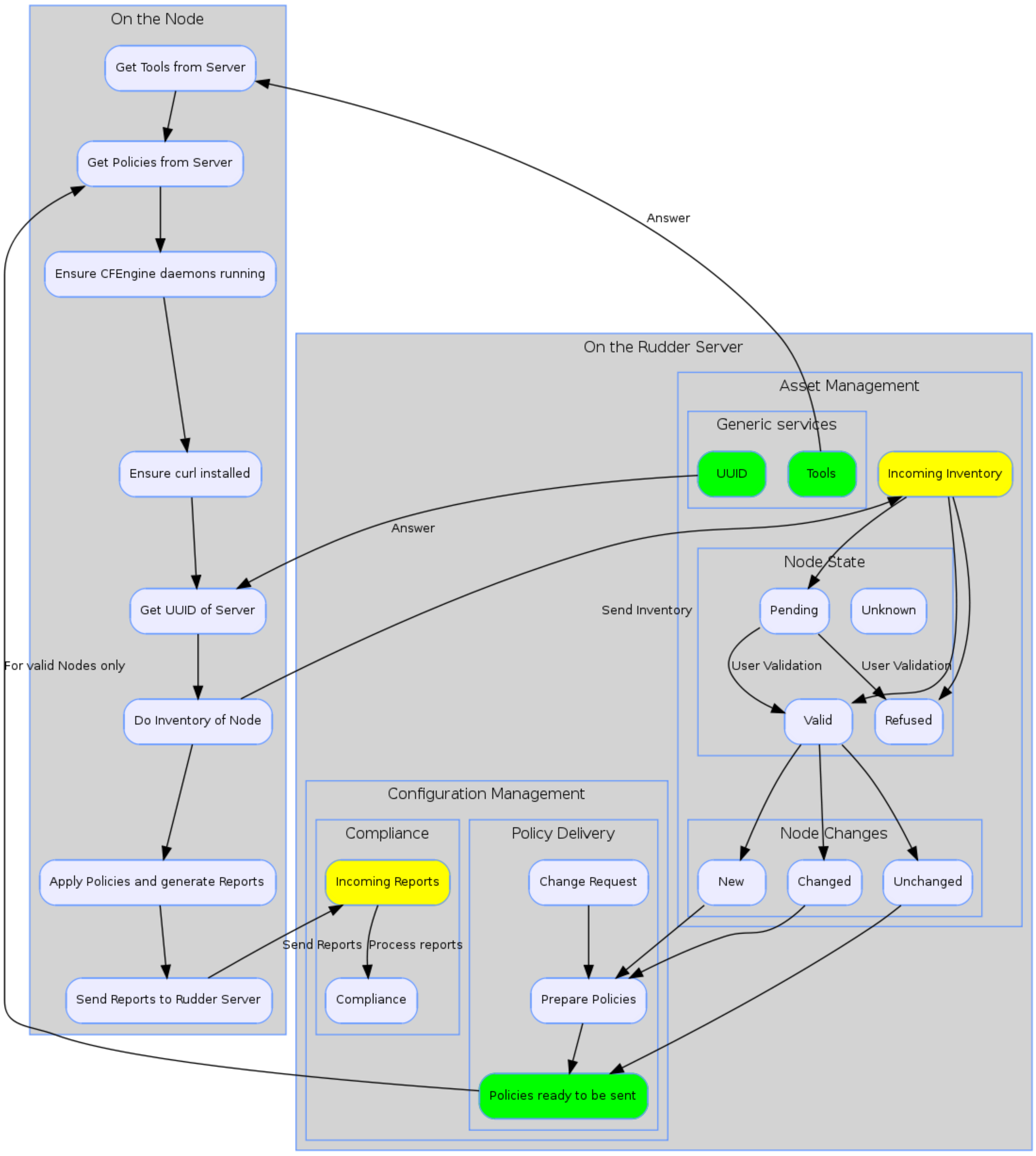


Figure 7.1: Rudder Agent workflow

7.9.3 Installation of the Rudder Agent

7.9.3.1 Static files

At installation of the Rudder Agent, files and directories are created in following places:

`/etc` Scripts to integrate Rudder Agent in the system (init, cron).

`/opt/rudder/share/initial-promises` Initialization promises for the *Rudder Agent*. These promises are used until the *Node* has been validated in *Rudder*. They are kept available at this place afterwards.

`/opt/rudder/lib/perl5` The *FusionInventory Inventory* tool and its Perl dependencies.

`/opt/rudder/bin/run-inventory` Wrapper script to launch the inventory.

`/opt/rudder/sbin` Binaries for *CFEngine Community*.

`/var/rudder/cfengine-community` This is the working directory for *CFEngine Community*.

7.9.3.2 Generated files

At the end of installation, the *CFEngine Community* working directory is populated for first use, and unique identifiers for the *Node* are generated.

`/var/rudder/cfengine-community/bin/` *CFEngine Community* binaries are copied there.

`/var/rudder/cfengine-community/inputs` Contains the actual working *CFEngine Community* promises. Initial promises are copied here at installation. After validation of the *Node*, *Applied Polic*ies, which are the *CFEngine* promises generated by *Rudder* for this particular *Node*, will be stored here.

`/var/rudder/cfengine-community/ppkeys` An unique SSL key generated for the *Node* at installation time.

`/opt/rudder/etc/uuid.hive` An unique identifier for the *Node* is generated into this file.

7.9.3.3 Services

After all of these files are in place, the *CFEngine Community* daemons are launched:

`cf-execd` This *CFEngine Community* daemon is launching the *CFEngine Community Agent* `cf-agent` every 5 minutes.

`cf-serverd` This *CFEngine Community* daemon is listening on the network for a forced launch of the *CFEngine Community Agent* coming from the Rudder *Server*'s Big Red Button.

7.9.3.4 Configuration

At this point, you should configure the Rudder Agent to actually enable the contact with the server. Type in the *IP address* of the Rudder *Root Server* in the following file:

```
echo *root_server_IP_address* > /var/rudder/cfengine-community/policy_server.dat
```

7.9.4 Tasks executed by Rudder Agent on the Node

Let's consider the Rudder Agent is installed and configured on the new *Node*.

The Rudder Agent is regularly launched and performs following tasks sequencially, in this order:

7.9.4.1 Request data from Rudder Server

The first action of Rudder Agent is to fetch the `tools` directory from *Rudder* Server. This directory is located at `/opt/rudder/share/tools` on the *Rudder* Server and at `/var/rudder/tools` on the *Node*. If this directory is already present, only changes will be updated.

The agent then try to fetch new *Applied Polic*ies from Rudder *Server.* Only requests from valid *Nodes* will be accepted. At first run and until the *Node* has been validated in *Rudder*, this step fails.

7.9.4.2 Launch processes

Ensure that the *CFEngine* community daemons `cf-execd` and `cf-serverd` are running. Try to start these daemons if they are not already started.

Daily between 5:00 and 5:05, relaunch the *CFEngine Community* daemons `cf-execd` and `cf-serverd`.

Add a line in `/etc/crontab` to launch `cf-execd` if it's not running.

Ensure again that the *CFEngine* community daemons `cf-execd` and `cf-serverd` are running. Try to start these daemons if they are not already started.

7.9.4.3 Identify Rudder Root Server

Ensure the `curl` package is installed. Install the package if it's not present.

Get the identifier of the Rudder *Root Server,* necessary to generate reports. The URL of the identifier is http://*Rudder*_root_server/uuid

7.9.4.4 Inventory

If no inventory has been sent since 8 hours, or if a forced inventory has been requested (class `force_inventory` is defined), do and send an inventory to the server.

```
user@node:~$ sudo /var/rudder/cfengine-community/bin/cf-agent -KI -Dforce_inventory=true
```

No reports are generated until the *Node* has been validated in Rudder *Server.*

7.9.4.5 Syslog

After validation of the *Node*, the system log service of the *Node* is configured to send reports regularly to the server. Supported system log providers are: `syslogd`, `rsyslogd` and `syslog-ng`.

7.9.4.6 Apply Policy Instances

Apply other policies and write reports locally.

7.9.5 Rudder Agent interactive

You can force the Rudder Agent to run from the console and observe what happens.

```
user@node:~$ sudo /var/rudder/cfengine-community/bin/cf-agent -KI
```

Error: the name of the Rudder Root Server can't be resolved
If the *Rudder* Root Server *name is not resolvable, the* Rudder Agent will issue this error:

```
user@node:~$ sudo /var/rudder/cfengine-community/bin/cf-agent -KI

Unable to lookup hostname (rudder-root) or cfengine service: Name or service not ←
    known
```

To fix it, either you set up the agent to use the IP adress of the *Rudder* root server instead of its Domain name, either you set up accurately the name resolution of your Rudder *Root Server,* in your DNS server or in the hosts file.
The Rudder *Root Server* name is defined in this file

```
root@node:~# echo *IP_of_root_server* > /var/rudder/cfengine-community/policy_server ←
    .dat
```

Error: the CFEngine service is not responding on the Rudder Root Server
If the *CFEngine* is stopped on the Rudder *Root Server* you will get this error:

```
user@node:~$ sudo /var/rudder/cfengine-community/bin/cf-agent -KI
 !! Error connecting to server (timeout)
 !!! System error for connect: "Operation now in progress"
 !! No server is responding on this port
Unable to establish connection with rudder-root
```

Restart the *CFEngine* service:

```
user@rudder-root:~$ sudo /var/rudder/cfengine-community/bin/cf-serverd
```

7.9.6 Processing new inventories on the server

7.9.6.1 Verify the inventory has been received by the Rudder Root Server

There is some delay between the time when the first inventory of the *Node* is sent, and the time when the *Node* appears in the New *Nodes* of the web interface. For the brave and impatient, you can check if the inventory was sent by listing incoming *Nodes* on the server:

```
ls /var/rudder/inventories/incoming/
```

7.9.6.2 Process incoming inventories

On the next run of the *CFEngine* agent on Rudder *Root Server,* the new inventory will be detected and sent to the *Inventory* Endpoint. The inventory will be then moved in the directory of received inventories. The the *Inventory* Endpoint do its job and the new *Node* appears in the interface.

You can force the execution of *CFEngine* agent on the console:

```
user@rudder-root:~$ sudo /var/rudder/cfengine-community/bin/cf-agent -KI
```

7.9.6.3 Validate new Nodes

User interaction is required to validate new *Nodes*.

7.9.6.4 Prepare policies for the Node

Policies are not shared between the *Nodes* for obvious security and confidentiality reasons. Each *Node* has its own set of policies. Policies are generated for *Nodes* according in the following states:

1. *Node* is new;
2. *Inventory* has changed;
3. *Policy Template* has changed;
4. *Policy Instance* has changed;
5. Group of *Node* has changed;
6. *Configuration Rule* has changed;
7. Regeneration was forced by the user.

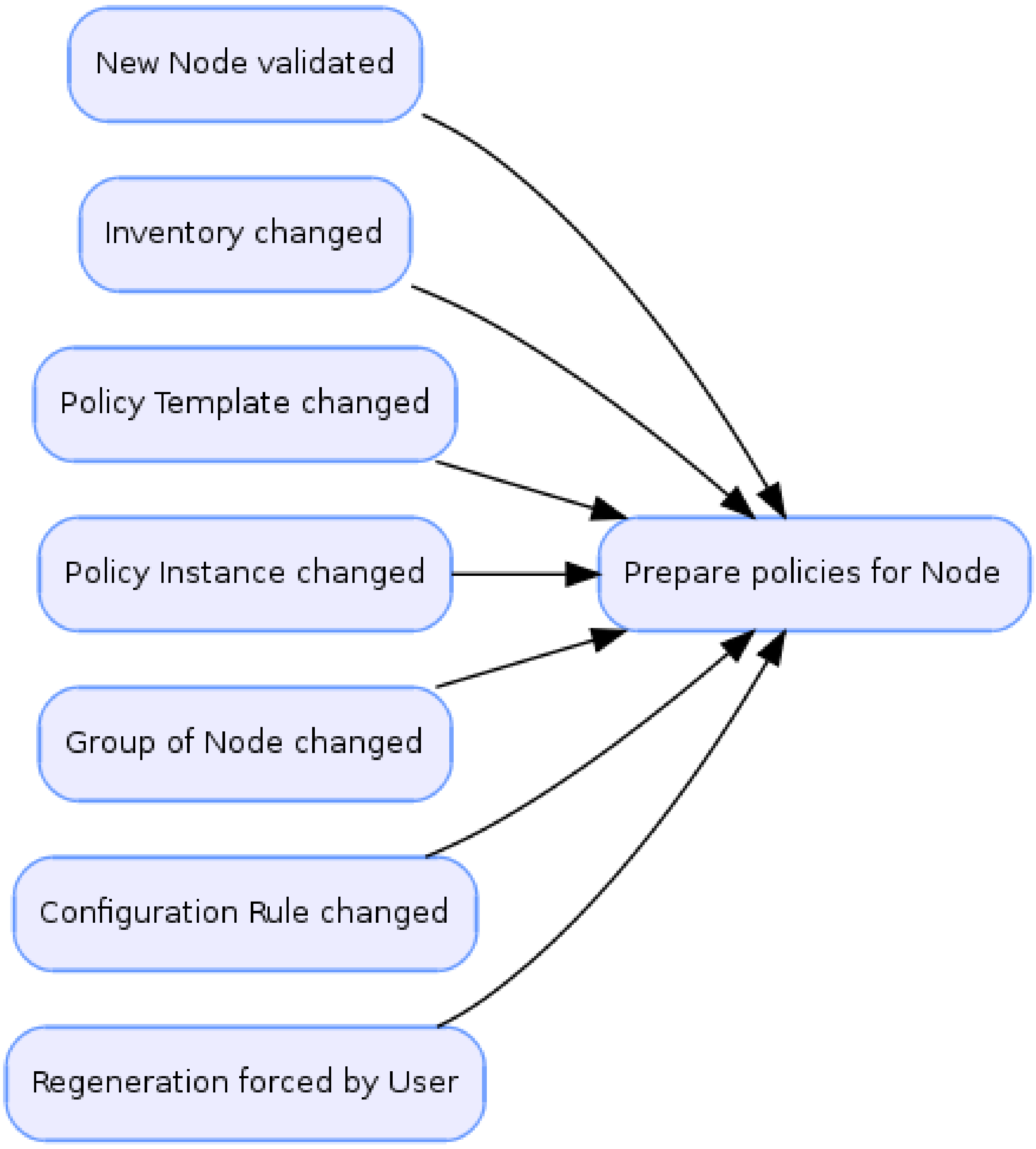


Figure 7.2: Generate policy workflow

7.10 Configuration files

/etc/default/rudder-agent

```
#=====================================================================
# Configuration sample for Cfengine Community init script
#=====================================================================

# Cfengine Community directory and files
CFENGINE_COMMUNITY_PATH="/opt/rudder"
CFENGINE_COMMUNITY_VAR_PATH="/var/rudder/cfengine-community"
```

```
CFENGINE_COMMUNITY_RUN[CFEXECD]="1"
CFENGINE_COMMUNITY_RUN[CFSERVERD]="1"
CFENGINE_COMMUNITY_RUN[CFMONITORD]="0"
CFENGINE_COMMUNITY_BIN[CFEXECD]="$CFENGINE_COMMUNITY_VAR_PATH/bin/cf-execd"
CFENGINE_COMMUNITY_BIN[CFSERVERD]="$CFENGINE_COMMUNITY_VAR_PATH/bin/cf-serverd"
CFENGINE_COMMUNITY_BIN[CFMONITORD]="$CFENGINE_COMMUNITY_VAR_PATH/bin/cf-monitord"
CFENGINE_COMMUNITY_PARAMS[CFEXECD]=""
CFENGINE_COMMUNITY_PARAMS[CFSERVERD]=""
CFENGINE_COMMUNITY_PARAMS[CFMONITORD]=""
CFENGINE_COMMUNITY_PID_FILE[CFEXECD]="$CFENGINE_COMMUNITY_VAR_PATH/cf-execd.pid"
CFENGINE_COMMUNITY_PID_FILE[CFSERVERD]="$CFENGINE_COMMUNITY_VAR_PATH/cf-serverd.pid"
CFENGINE_COMMUNITY_PID_FILE[CFMONITORD]="$CFENGINE_COMMUNITY_VAR_PATH/cf-monitord.pid"

# Other
TIMEOUT="60" # Max time to start/stop processes
SYSLOG_FACILITY="local6"
PS_COMMAND="ps -efww"   # This ensures full width for ps output but doesn’t work on Solaris ↩
     - use "ps -ef"
```

/opt/rudder/etc/htpasswd-webdav

```
rudder:vHBLbrOyfEWFg
```

/opt/rudder/etc/inventory-web.properties

```
##
# Default configuration file for the application.
# You can define the location of this file by
# setting "inventoryweb.configFile" JVM property,
# for example:
# java .... -Dinventoryweb.configFile=/opt/rudder/etc/inventory-web.conf
##

#
## LDAP related configuration
#

#  LDAP directory connection information
ldap.host=localhost
ldap.port=389
ldap.authdn=cn=Manager,cn=rudder-configuration
ldap.authpw=secret

# inventories information
ldap.inventories.software.basedn=ou=Inventories,cn=rudder-configuration
ldap.inventories.accepted.basedn=ou=Accepted Inventories,ou=Inventories,cn=rudder- ↩
    configuration
ldap.inventories.pending.basedn=ou=Pending Inventories,ou=Inventories,cn=rudder- ↩
    configuration

# where to store LDIF inventory versions
history.inventories.rootdir=/var/rudder/inventories/historical

# where to store debug information about LDAP modification requests
ldif.tracelog.rootdir=/var/rudder/inventories/debug
```

/opt/rudder/etc/logback.xml

```
<configuration>
  <!--
    This is the default logging configuration file. It will be used if you
    didn’t specify the "logback.configurationFile" JVM option.
    For example, to use a loggin configuration file in "/etc/rudder":
    java ... -Dlogback.configurationFile=/etc/rudder/logback.xml

    Full information about the file format is available on the project
    web site: http://logback.qos.ch/manual/configuration.html#syntax
   -->

  <!--
    Appender configuration - where&how to write logs in SLF4J speaking.
    ====================================================================
    Our default configuration : log on stdout appender so that our logs
    are managed by the container log system (and so, if Tomcat/Jetty/etc
    logs are stored in files and rotated, so are our log information).

    Log format is:
    - date/time/thread of the log on 30 chars (fixed)
    - log level on 5 char (fixed)
    - name of the logger (and so the class) on 36 chars, with
      package name folding
    - log message follows
    - limit exception trace to 30 calls

    You should not have to modify that.
  -->
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <Pattern>%-30(%d{HH:mm:ss.SSS} [%thread]) %-5level %logger{36} - %msg%n%xEx{30}</ ↩
          Pattern>
    </encoder>
  </appender>

  <!--
    Manage the global log level of the application.
    ===============================================

    That level will be used for all logs that are not
    more precisely defined below (i.e for whom there is
    no <logger name="...." level="..."/> defined)

    Available log levels are:
         trace < debug < info < warn < error < off
    "off" completely shut down logging for the given logger

    Do not modify the appender part if you don’t know what you
    are doing.
  -->

  <root level="info">
    <appender-ref ref="STDOUT" />
  </root>

  <!--
    Debug LDAP write operations
    ===========================

    This logger allow to trace LDAP writes operation and
    to output them in LDIF file (the output directory path
    is configured in the main configuration file)
    The trace is done only if level is set to "trace"
    WARNING: setting the level to trace may have major
    performance issue, as A LOT of LDIF files will have
    to be written.
    You should activate that log only for debugging purpose.
  -->

  <logger name="trace.ldif.in.file" level="off" />


  <!-- ==================================================== -->
  <!-- YOU SHOULD NOT HAVE TO CHANGE THINGS BELOW THAT LINE -->
  <!-- ==================================================== -->

  <!--
    Display AJAX information of the Web interface
    =============================================
    Whatever the root logger level is, you are likely
    to not wanting these information.
    Set the level to debug if you are really interested
    in AJAX-related debug messages.
  -->
  <logger name="comet_trace" level="info" />

  <!--
    Spring Framework log level
    ==========================
    We really don't want to see SpringFramework debug info,
    whatever the root logger level is - it's an internal
    component only.
  -->
  <logger name="org.springframework" level="warn" />

  <!--
    We don't need to have a timing information for each
    HTTP request.
    If you want to have these information, set the log
    level for that logger to (at least) "info"
   -->
  <logger name="net.liftweb.util.TimeHelpers" level="warn" />

</configuration>
```

/opt/rudder/etc/openldap/slapd.conf

```
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include         /opt/rudder/etc/openldap/schema/core.schema
include         /opt/rudder/etc/openldap/schema/cosine.schema
include         /opt/rudder/etc/openldap/schema/nis.schema
include         /opt/rudder/etc/openldap/schema/dyngroup.schema
include         /opt/rudder/etc/openldap/schema/inventory.schema
include         /opt/rudder/etc/openldap/schema/rudder.schema

loglevel none stats

# Define global ACLs to disable default read access.

# Do not enable referrals until AFTER you have a working directory
# service AND an understanding of referrals.
#referral       ldap://root.openldap.org

pidfile         /var/rudder/run/slapd.pid
argsfile        /var/rudder/run/slapd.args
```

```
# Load dynamic modules for backends and overlays:
modulepath      /opt/rudder/libexec/openldap/
moduleload      back_hdb.la
moduleload      back_monitor.la
moduleload  dynlist.la

# Sample security restrictions
#       Require integrity protection (prevent hijacking)
#       Require 112-bit (3DES or better) encryption for updates
#       Require 63-bit encryption for simple bind
# security ssf=1 update_ssf=112 simple_bind=64

# Sample access control policy:
#       Root DSE: allow anyone to read it
#       Subschema (sub)entry DSE: allow anyone to read it
#       Other DSEs:
#               Allow self write access
#               Allow authenticated users read access
#               Allow anonymous users to authenticate
#       Directives needed to implement policy:
# access to dn.base="" by * read
# access to dn.base="cn=Subschema" by * read
# access to *
#       by self write
#       by users read
#       by anonymous auth
#
# if no access controls are present, the default policy
# allows anyone and everyone to read anything but restricts
# updates to rootdn.  (e.g., "access to * by * read")
#
# rootdn can always read and write EVERYTHING!

##############################################
# Global overlays (available on all databases)
##############################################
overlay dynlist
dynlist-attrset dynGroup memberURL

#######################################################################
# BDB database definitions
#######################################################################

database        hdb
suffix          "cn=rudder-configuration"
rootdn          "cn=Manager,cn=rudder-configuration"
# Cleartext passwords, especially for the rootdn, should
# be avoid.  See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw          secret
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory   /var/rudder/ldap/openldap-data
# Checkpoint database every 128k written or every 5 minutes
checkpoint      0       1
# Indices to maintain
index   objectClass     eq
index   confirmed       eq
index   uuid,machineUuid,nodeId,machine,hostedVm,container,node,software eq
index   mountPoint,softwareVersion,cn   eq
index   member eq

database monitor
```

/opt/rudder/etc/postgresql/reportsSchema.sql

```
-- SQL schema for the reports data

-- set the report to warnings
set client_min_messages='warning';


-- Create the sequences
Create SEQUENCE confSerialId START 1;

Create SEQUENCE confVersionId START 1;

-- Create the table for the configuration reports information
create table configurationReportsInfo (
        serialId integer PRIMARY KEY DEFAULT nextval('confSerialId'),
        versionId integer NOT NULL,
        configurationRuleId text NOT NULL CHECK (configurationRuleId <> ''),
        serial integer NOT NULL,
        policyInstanceId text NOT NULL CHECK (policyInstanceId <> ''),
        component text NOT NULL CHECK (component <> ''),
        cardinality integer NOT NULL,
        componentsValues text NOT NULL, -- this is the serialisation of the expected values
        beginDate timestamp with time zone NOT NULL,
        endDate timestamp with time zone
);

create index configuration_versionId on configurationReportsInfo (versionId);
create index configuration_serialId on configurationReportsInfo (configurationRuleId,  ↩
    serial);

create table configurationServerList (
        versionId integer NOT NULL ,
        serverUuid varchar(50) NOT NULL  CHECK (serverUuid <> ''),
        primary key (versionId, serverUuid)
);

create index configurationServerList_versionId on configurationServerList (versionId);


-- create the table for the reports sent

create sequence serial START 101;

CREATE TABLE RudderSysEvents (
id integer PRIMARY KEY default nextval('serial'),
executionDate timestamp with time zone NOT NULL,
nodeId text NOT NULL CHECK (nodeId <> ''),
policyInstanceId text NOT NULL CHECK (policyInstanceId <> ''),
configurationRuleId text NOT NULL CHECK (configurationRuleId <> ''),
serial integer NOT NULL,
component text NOT NULL CHECK (component <> ''),
keyValue text,
executionTimeStamp timestamp with time zone NOT NULL,
eventType varchar(64),
policy text,
msg text
);
```

```
create index nodeid_idx on RudderSysEvents (nodeId);
create index date_idx on RudderSysEvents (executionDate);
create index policyInstanceId_idx on RudderSysEvents (policyInstanceId);
create index configurationRuleId_idx on RudderSysEvents (configurationRuleId);
create index configurationRuleId_node_idx on RudderSysEvents (configurationRuleId, nodeId);
create index configurationRuleId_serialed_idx on RudderSysEvents (configurationRuleId,  ↩
    serial);
create index composite_idx on RudderSysEvents (configurationRuleId, policyInstanceId,  ↩
    serial, executionTimeStamp);


-- Log event part

CREATE SEQUENCE eventLogIdSeq START 1;


CREATE TABLE EventLog (
    id integer PRIMARY KEY  DEFAULT nextval('eventLogIdSeq'),
    creationDate timestamp with time zone NOT NULL DEFAULT 'now',
    severity integer,
    causeId integer,
    principal varchar(64),
    eventType varchar(64),
    data xml
);

create index eventType_idx on EventLog (eventType);
create index causeId_idx on EventLog (causeId);



create sequence GroupsId START 101;


CREATE TABLE Groups (
id integer PRIMARY KEY default nextval('GroupsId'),
groupId text NOT NULL CHECK (groupId <> ''),
groupName text,
groupDescription text,
nodeCount int,
startTime timestamp with time zone default now(),
endTime timestamp with time zone
);



create index groups_id_start on Groups (groupId, startTime);
create index groups_end on Groups (endTime);


create sequence PolicyInstancesId START 101;


CREATE TABLE PolicyInstances (
id integer PRIMARY KEY default nextval('PolicyInstancesId'),
policyInstanceId text NOT NULL CHECK (policyInstanceId <> ''),
policyInstanceName text,
policyInstanceDescription text,
priority integer NOT NULL,
policyPackageName text,
policyPackageVersion text,
policyPackageDescription text,
startTime timestamp with time zone NOT NULL,
endTime timestamp with time zone
);


create index pi_id_start on PolicyInstances (policyInstanceId, startTime);
create index pi_end on PolicyInstances (endTime);

create sequence ConfigurationRulesId START 101;


CREATE TABLE ConfigurationRules (
id integer PRIMARY KEY default nextval('ConfigurationRulesId'),
configurationRuleId text NOT NULL CHECK (configurationRuleId <> ''),
serial integer NOT NULL,
name text,
shortdescription text,
longdescription text,
isActivated boolean,
startTime timestamp with time zone NOT NULL,
endTime timestamp with time zone
);

CREATE TABLE ConfigurationRulesGroups (
CrId integer, -- really the id of the table ConfigurationRules
groupId text NOT NULL CHECK (groupId <> ''),
PRIMARY KEY(CrId, groupId)
);

CREATE TABLE ConfigurationRulesPolicyInstance (
CrId integer, -- really the id of the table ConfigurationRules
policyInstanceId text NOT NULL CHECK (policyInstanceId <> ''),
PRIMARY KEY(CrId, policyInstanceId)
);


create index cr_id_start on ConfigurationRules (configurationRuleId, startTime);
create index cr_end on ConfigurationRules (endTime);
```

/opt/rudder/etc/reportsInfo.xml

```
<ReportsInfoStore>
</ReportsInfoStore>
```

/opt/rudder/etc/rudder-users.xml

```
<!--
 The "authentication" tag can have a "hash" argument, with these allowed values:
 "md5", "sha1", "sha256", "sha-256", "sha512", "sha-512"

 For example: <authentication hash="sha">

 To hash passwords for this format, run these commands:
 "md5"                    read mypass; echo -n $mypass | md5sum
 "sha" or "sha1"          read mypass; echo -n $mypass | shasum
 "sha256" or "sha-256"    read mypass; echo -n $mypass | sha256sum
 "sha512" or "sha-512"    read mypass; echo -n $mypass | sha512sum

  After changing this file, the rudder webapp must be restarted to take changes
  into account: /etc/init.d/jetty restart
 -->

<!-- example with hash -->
<!-- <authentication hash="sha"> -->
<authentication>
  <user name="jon.doe"  password="secret"/>
  <user name="alex.bar" password="secret2"/>
  <!--  exemple of bad lines -->
  <!--  <user name="" password="secret2"/>-->
  <!--  <user name="name" password=""/>-->
</authentication>
```

/opt/rudder/etc/rudder-web.properties

```
##
# Default configuration file for the application.
# You can define the location of the file by
# setting "rudder.configFile" JVM property,
# for example:
# java .... -Drudder.configFile=/opt/rudder/etc/rudder-web.conf
##


##
# Application information
##
#define that property if you are behind a proxy
#or anything that make the URL served by the
#servlet container be different than the public one
#note: if defined, must not end with /
#let blank to use default value
base.url=http://rudder-debian/rudder

##
#  LDAP properties
##

#  LDAP directory connection information
ldap.host=localhost
ldap.port=389
ldap.authdn=cn=manager,cn=rudder-configuration
ldap.authpw=secret

#inventories information
ldap.inventories.software.basedn=ou=Inventories, cn=rudder-configuration
ldap.inventories.accepted.basedn=ou=Accepted Inventories, ou=Inventories, cn=rudder- ↩
    configuration
ldap.inventories.pending.basedn=ou=Pending Inventories, ou=Inventories, cn=rudder- ↩
    configuration

#Base DN for Rudder Data
ldap.rudder.base=ou=Rudder, cn=rudder-configuration

#Base DN (the ou=Node is already given by the DIT)
ldap.node.base=cn=rudder-configuration

#  directory where LDIF trace of LDAP modify request are
#  stored when loglevel is 'trace'
ldif.tracelog.rootdir=/var/rudder/inventories/debug



##
```

```
# Other Rudder Configuration properties
##

#
# directory used as root directory to store LDIF dump
# of historised inventories
history.inventories.rootdir=/var/rudder/inventories/historical

##
#  Upload directory
##
#  directory where new uploaded files are stored
upload.root.directory=/var/rudder/files/

##
#  Emergency stop
##
#  path to the script/binary that allows emergency orchestrator stop
bin.emergency.stop=/opt/rudder/bin/cfe-red-button.sh


##
#  Promise writer directory configuration
##
rudder.dir.config=/opt/rudder/etc/
rudder.dir.policyPackages=/opt/rudder/share/policy-templates
rudder.dir.licensesFolder=/opt/rudder/etc/licenses
rudder.dir.policies=/var/rudder/
rudder.dir.backup=/var/rudder/backup/
rudder.dir.dependencies=/var/rudder/tools/
rudder.dir.sharing=/var/rudder/files/
rudder.dir.lock=/var/rudder/lock/
rudder.endpoint.cmdb=http://localhost:8080/endpoint/upload/

# Port used by the community edition
rudder.community.port=5309


rudder.jdbc.driver=org.postgresql.Driver
rudder.jdbc.url=jdbc:postgresql://localhost:5432/rudder
rudder.jdbc.username=rudder
rudder.jdbc.password=Normation


#
# Destination directory for files distributed
# with the copyFile policy
#
policy.copyfile.destination.dir=/some/default/destination/directory/

#
# Command line to check the promises generated
#
rudder.community.checkpromises.command=/var/rudder/cfengine-community/bin/cf-promises
rudder.nova.checkpromises.command=/bin/true


#
# Interval of time between two dynamic group update batch
# Expect an int (amount of minutes)
#
rudder.batch.dyngroup.updateInterval=5
```

```
#
# Interval of time (in seconds) between two checks
# for a policy template library update (a commit)
# 300s = 5minutes
#
rudder.batch.ptlib.updateInterval=300


#
# Configure the refs path to use for the git repository for
# the Policy Template Reference Library.
# The default is to use "refs/heads/master" (the local master
# branche).
# You have to use the full ref path.
rudder.ptlib.git.refs.path=refs/heads/master
```

Chapter 8

Glossary

Applied Policy This is the result of the conversion of a *Policy Instance* into a set of *CFEngine* Promises for a particular *Node*.

"Big red button" A button, on the right top side of every page of *Rudder* web interface, to command the emergency stop of the agents. This stop will be implicitly done in less than 10 minutes, or can be done immediately if the port 5309 TCP from the Rudder *Root Server* (or each relay server) is open to each nodes. This feature is detailed in the user documentation.

`cf-execd` This *CFEngine Community* daemon is launching the *CFEngine Community Agent* `cf-agent` every 5 minutes.

`cf-serverd` This *CFEngine Community* daemon is listening on the network for a forced launch of the *CFEngine Community Agent* coming from the Rudder *Server*'s Big Red Button.

CFEngine Nova Managing *Windows* machines requires the commercial version of *CFEngine*, called *Nova*. It needs to open the port 5308 TCP from the *Node* to the *Rudder* Root Server.

***CFEngine* server** Distribute the *CFEngine* configuration to the nodes.

CFEngine *CFEngine* is a configuration management software. *CFEngine* comes from a contraction of “ConFiguration Engine”.

Configuration Rule It is the application of a policy to a group of nodes. It is the glue between both Asset Management and Configuration Management parts of the application.

Dynamic group Group of *Nodes* based on search criteria. The search is replayed every time the group is queried. The list will always contain the nodes that match the criteria, even if the data nodes have changed since the group was created.

LDAP server Store the inventories and the *Node* configurations.

Policy Instance This is an instance of a *Policy Template,* which allows to set values for the parameters of the latter. Each *Policy Instance* can have an unique name. A Policy Instance should be completed with a short and a long description, and a collection of parameters for the variables defined by the *Policy Template.*

Policy Template This is a configuration skeleton, adapted to a function or a particular service (eg DNS resolver configuration). This skeleton includes the configuration logic for this function or service, and can be set according to a list of variables (in the same example: *IP address*es of DNS servers, the default search box, . . .)

Port 514, TCP Syslog port, used to centralize reports.

Port 5308, TCP *Nova* communication port, used by the commercial version of *CFEngine*, which is required to manage *Windows* nodes.

Port 5309, TCP *CFEngine* communication port, used to communicate the policies to the rudder nodes.

Port 80, TCP, for nodes HTTP communication port, used to send inventory and fetch the id of the Rudder *Server.*

Port 80, TCP, for users HTTP communication port, used by the users to access to the web interface.

Reference *Policy Template* Library This is an organized list of every available *Policy Templates*. This list can't be modified: every changes made by an user will be applied to the User Policy Template Library.

Rudder Node A *Node* is client computer managed by *Rudder*. To be managed, a *Node* must first be accepted as an authorized node.

Rudder *Relay Server* Relay servers are not available in the current version. In a future version, these optional servers will let you adapt your *Rudder* architecture to your existing network topology, by acting as a proxy for flows exchanged between managed nodes and the root server.

Rudder *Root Server* This is the core of the *Rudder* infrastructure. This server must be a dedicated machine (either virtual of physical), and contains the main application components: the web interface, databases, configuration data, logs…

Rudder *Rudder* is a Drift Assesment software. *Rudder* associates Asset Management and Configuration Management. *Rudder* is a Free Software developped by *Normation*.

SQL server Store the received reports from the nodes.

Static group Group of *Nodes* based on search criteria. The search is performed once and the resulting list of *Nodes* is stored. Once declared, the list of nodes will not change, except manual change.

User *Policy Template* Library This is an organized list of the *Policy Templates* selected and modified by the user. By default this list is the same as the Reference *Policy Template* Library. *Policy Templates* can be disabled or deleted, and then activated again with a simple drag and drop. Categories can be reorganised according to the desired taxonomy. A *Policy Template* can appear only once in the Library.

Web server application Execute the web interface and the server that handles the new inventories.

Web server front-end Handle the connection to the Web interface, the received inventories and the sharing of the UUID Rudder *Root Server*.

License