Rudder 3.0 - User Manual

COLLABORATORS

	TITLE : Rudder 3.0 - User Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jonathan Clarke, Nicolas Charles, Fabrice Flore-Thebault, Matthieu Cerda, Nicolas Perron, Arthur Anglade, Vincent Membré, and François Armand	Nov 2014	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
3.0.0	Nov 2014		N

Contents

List of Figures

List of Tables

Chapter 1

Online version

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

Chapter 2

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.

2.1 Concepts

2.1.1 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 information produced by the asset management function to identify these *Nodes* and evaluate some specific information about them.

2.1.2 Asset management concepts

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

2.1.2.1 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*.

2.1.2.2 Search Nodes

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

2.1.2.3 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.

2.1.3 Configuration management concepts

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

Technique This is a configuration skeleton, adapted to a function or a particular service (e.g. 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 addresses of DNS servers, the default search box, . . .)

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

Rule It is the application of one or more directives to a group of nodes. It is the glue between both Asset Management and Configuration Management parts of the application.

Applied Policy 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 rules are linking the functions of inventory management and configuration management.

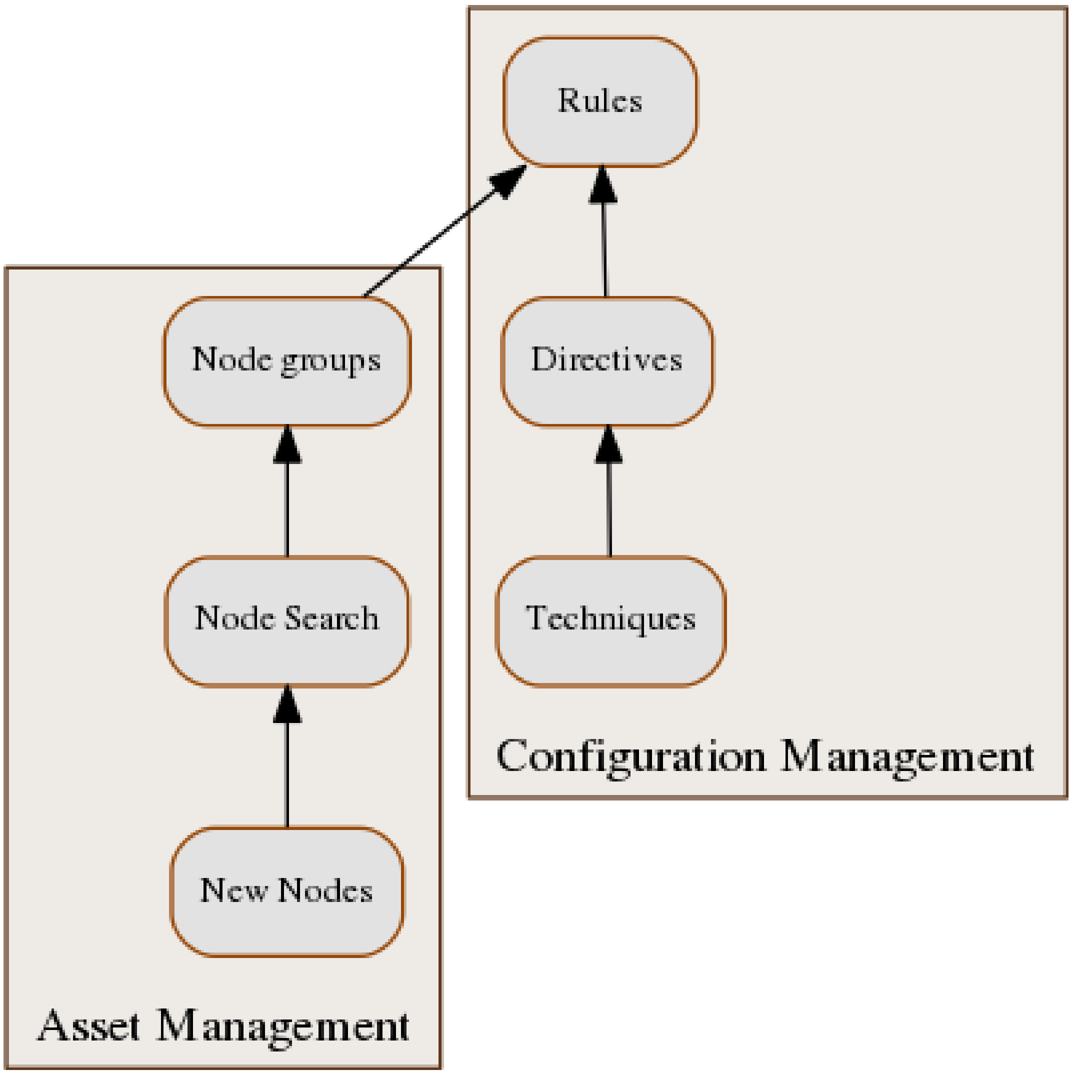


Figure 2.1: Concepts diagram

2.2 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 an optional component in a *Rudder* architecture. They can act as a proxy for all network communications between *Rudder* agents and a *Rudder* server. This enables them to be installed in a remote datacenter, or inside a restricted network zone, to limit the network flows required to use *Rudder*.

Chapter 3

Installation

3.1 Requirements

3.1.1 Networking

3.1.1.1 Mandatory flows

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

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

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

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

And this one is optional:

Port 5310, TCP *CFEngine* communication port, used to communicate the policies to the *Rudder* nodes when debugging communication between a *Node* and a policy server with the `rudder server debug` command.

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

Port 443, TCP, for users HTTP/S communication port, used to access the *Rudder* web interface.

3.1.1.2 Optional flows

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

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

This version used to be called Nova before.

3.1.1.3 DNS - Name resolution

By default, *Rudder* relies on the *Node* declared hostnames to identify them, for security reasons. It is required that each *Node* hostname can be resolved to its IP address that will be used to contact the *Rudder 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.

If you can not make every node resolution consistent, it is possible to remove this constraint by unticking "Use reverse DNS lookups on nodes to reinforce authentication to policy server:" in the Administration - Settings tab of the *Rudder* web interface.

3.1.2 Supported Operating Systems

3.1.2.1 For Rudder Nodes

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

GNU/Linux:

- *Debian* 5 to 8
- RedHat *Enterprise* Linux (*RHEL*) / *RHEL*-like 3 to 7
- *Fedora* 18
- *SuSE* Linux *Enterprise* Server (SLES) 10 SP3, 11 SP1, 11 SP3
- *Ubuntu* 10.04 LTS (Lucid), 12.04 LTS (Precise), 12.10 (Quantal), 14.04 LTS (Trusty)

Other Unix systems:

- IBM AIX 5.3, 6.1 and 7.1

Windows:

- Microsoft Windows Server 2000, 2003, 2008, 2008 R2, 2012

Windows and AIX Nodes

- On *Windows*, installing *Rudder* requires the commercial version of *CFEngine* (named *CFEngine Enterprise*)
- For IBM AIX, pre-built RPM packages are distributed by *Normation* only

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 these 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, some *Techniques* may not work properly. If you wish to try *Rudder* on other systems, please get in touch with us!

A reference about how to manually build a *Rudder* agent is available on *Rudder*'s website here: http://www.rudder-project.org/foswiki/Development/*Agent*Build

3.1.2.2 For Rudder Root Server

The following operating systems are supported as a Root server:

GNU/Linux:

- *Debian* 7 and 8
- RedHat *Enterprise* Linux (*RHEL*) / *RHEL*-like 6 and 7
- *SuSE* Linux *Enterprise* Server (SLES) 11 SP1, 11 SP3
- *Ubuntu* 12.04 LTS (Precise), 14.04 LTS (Trusty)

3.1.3 Hardware specifications and sizing for Rudder Root Server

A dedicated server is strongly recommended, either physical or virtual with at least one dedicated core. *Rudder Server* runs on both 32 (if available) and 64 bit versions of every supported Operating System.

3.1.3.1 Memory

The required amount of RAM mainly depends on the number of managed nodes. A general rule for the minimal value is:

- less than 50 nodes: 2GB
- between 50 and 1000 nodes: 4GB
- more than 1000 nodes: 4GB + 1GB of RAM by 500 nodes above 1000.

When managing more than 1000 nodes, we also recommend you to use a multiserver installation for *Rudder* as described in chapter Multiserver Rudder.

For large installations, you should also tune the amount of RAM given to:

- the web application, as explained in the section about webapplication RAM configuration
- PostgresSQL, as explained in the Optimize PostgreSQL Server section

3.1.3.2 Disk

The PostgreSQL database will take up most disk space needed by *Rudder*. The storage necessary for the database can be estimated by counting around 150 to 400 kB by *Directive*, by *Node* and by day of retention of node's execution reports (the default is 4 days):

```
max_space = number of Directives * number of Nodes * retention duration in days * 400 kB
```

For example, a default installation with 500 nodes and an average of 50 *Directives* by node, should require between 14 GB and 38 GB of disk space for PostgreSQL.

Follow the Reports Retention section to configure the retention duration.

3.2 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.

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.

As *Rudder* data can grow really fast depending on your number of managed nodes and number of rules, it is advised to separate partitions to prevent /var getting full and break your system. Special attention should be given to:

/var/lib/pgsql (OS dependent). Please see the database maintenance chapter for more details about the PostgreSQL database size estimation.

/var/rudder Contains most of your server information, the configuration-repository, *LDAP* database, etc… *Rudder* application-related files should stay under 1GB, but the size of the configuration-repository will depend of the amount of data you store in it, especially in the shared-files folder (files that will get distributed to the agents using the "Download a file for the shared folder" *Technique*).

/var/log/rudder Report logs (/var/log/rudder/reports) size will depend on the amount of nodes you manage. It is possible to reduce this drastically by unticking "Log all reports received to /var/log/rudder/reports/all.log" under the Administration - Settings tab in the *Rudder* web interface. This will prevent *Rudder* from recording this logs in a text file on disk, and will only store them in the SQL database. This saves on space, and doesn't remove any functionality, but does however make debugging harder.

3.2.1 Install Rudder Root server on Debian or Ubuntu

3.2.1.1 Add the Rudder packages repository

Run the following commands as root to add the package repository GPG key and the repository itself:

```
apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 474A19E8
echo "deb http://www.rudder-project.org/apt-3.0/ $(lsb_release -cs) main" > /etc/apt/ ↩
    sources.list.d/rudder.list
apt-get update
```

Tip
If the HTTP Keyserver Protocol (11371/tcp) port is blocked on your network you can use this alternate command:

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

3.2.1.2 Install your Rudder Root Server

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

```
apt-get install rudder-server-root
```

Warning
Any nodes running **syslogd** (not syslog-ng or rsyslog) will **fail** to send any reports about the configuration rules they have applied to a *Rudder Server* running on *Ubuntu* (and only on *Ubuntu*). *Rudder* will apply rules on nodes but will never get reports from them. Therefore *Rudder* will not be able to calculate compliance.
The only supported platform using syslogd by default is ***RHEL***/***CentOS*** **5**, and several workarounds are available to fix this:

1. Install another syslog server on your nodes, such as rsyslog or syslog-ng.
2. Change the rsyslog configuration on the *Rudder* server (running *Ubuntu* 12.04 or later) to use port 514 and authorize this in the rsyslog configuration.
3. Setup iptables on the node to send syslog traffic to the correct port on your *Rudder* server.
4. Use a different OS for your *Rudder* server that *Ubuntu* Server 12.04 or later.

Now jump to the next section to configure your server.

3.2.2 Install Rudder Root server on SLES

3.2.2.1 Configure the package manager

Rudder requires three packages that are not always packaged by *SuSE* on all versions:

- rsyslog-psql
- PostgreSQL 9
- *Java* RE (version 7 at least).

The first one is present in *Rudder*'s repositories and can be downloaded alongside *Rudder* as a dependency, but you will need to install PostgeSQL 9 and a *Java* RE separately.

PostgreSQL 9 can be installed through the Open*SuSE* build service: https://build.opensuse.org/project/show/server:database:postgresql or through the system repositories, on post-SP1 systems.

The *Java* RE 7 can be found either using the Open*SuSE* build service, or through *Oracle*'s website: http://www.java.com

Also, *Rudder* server requires the `git` software, that can be found on SLES SDK DVD under the name `git-core`.

Warning
SLES 11 will try to install PostgreSQL 8.3 by default, which is not supported by *Rudder* and will cause various glitches in the web interface, as well as reporting failures.
It is really mandatory to either add the Open*SuSE* build service repository, or install postgresql91-server (if available) beforehand to prevent the system from choosing the default PostgreSQL implementation.

Warning
You might not be able to install *Rudder* rpm files locally with Zypper (for example with *zypper install rudder-agent-version.release-1.SLES.11.x86_64.rpm*), due to a bug (bnc#929483 on *SuSE* bugtracker) in its RPM headers parsing causing a segmentation fault. You can either:

- Install the packages directly from the repository, as described below
- Upgrade your libzypp package to a version including the fix provided by *SuSE* (upgrade for SLES11SP3 and for SLES12)
- Use the rpm command to install packages locally (for example with *rpm -i rudder-agent-version.release-1.SLES.11.x86_64.rpm*)

3.2.2.2 Add the Rudder packages repository

Run the following commands as root:

```
zypper ar -n "Rudder SLES repository" http://www.rudder-project.org/rpm-3.0/SLES_11/ Rudder
zypper ref
```

This will add the *Rudder* package repository, then update the local package cache.

3.2.2.3 Install your Rudder Root Server

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

```
zypper in rudder-server-root
```

Warning
Zypper seems to be quite tolerant to missing dependencies and will let you install `rudder-server-root` even if you are missing something like `git-core` for example, if nothing provides it or you did not install it beforehand.
Special care should be taken during initial installation not to say "Continue anyway" if Zypper does complain a dependency can not be resolved and asks what to do.

Now jump to the next section to configure your server.

3.2.3 Install Rudder Root server on RHEL-like systems

3.2.3.1 Add the Rudder packages repository

Run the following command as root:

```
version=$(grep "release [0-9]" /etc/redhat-release | sed "s/^.*release \([0-9]\).*$/\1/")
echo "[Rudder_3.0]
name=Rudder 3.0 EL repository
baseurl=http://www.rudder-project.org/rpm-3.0/RHEL_${version}/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-3.0/RHEL_${version}/repodata/repomd.xml.key" > / ↩
    etc/yum.repos.d/rudder.repo
```

3.2.3.2 Install your Rudder Root Server

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

```
yum install rudder-server-root
```

On Red Hat-like systems, a firewall setup is enabled by default, and would need to be adjusted for *Rudder* to operate properly. You have to allow all the flows described in the Network section.

Warning

Rudder server doesn't support SELinux yet (see http://www.rudder-project.org/redmine/issues/2882), so you should set it to be permissive with these commands:

```
sed -i "s%^\s*SELINUX=.*%SELINUX=disabled%" /etc/sysconfig/selinux
setenforce 0
```

Tip

On EL6, the /etc/sysconfig/iptables file configures the firewall:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
# Allow SSH access (Maintenance)
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
# Allow HTTPS access (Rudder)
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

The important line to have access to the Web interface being:

```
# Allow HTTPS access (Rudder)
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
```

Tip

On EL7, the default firewall is firewalld, and you can enable HTTP/S access by running

```
firewall-cmd --permanent --zone=public --add-port=443/tcp
```

Now jump to the next section to configure your server.

3.2.4 Initial configuration of your Rudder Root Server

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

```
/opt/rudder/bin/rudder-init
```

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

Allowed networks A list of IP networks authorized to connect to the server. It uses the network/CIDR 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. Also, the allowed networks can be adjusted later in the web interface in the Administration - Settings tab without having to run the script again.

Tip

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

3.2.5 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 screen, then a login prompt. The default login is "admin" with password "admin", authenticating you in the *Rudder* web interface with full administrative privileges.

The setup of the *Rudder* server is now over. If you plan to manage hundreds or thousands of *Nodes*, please note that some performance tuning can be necessary on the system.

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/configuration-repository/techniques` *Techniques* are stored here.

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

`/var/cfengine` Data for *CFEngine Enterprise* is stored here.

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

3.3 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 detailed description of the workflow, please refer to the Advanced Usage part of this documentation.

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.

3.3.1 Install Rudder Agent on Debian or Ubuntu

Validate the content of the *Rudder* project repository by importing the GPG key used to sign it:

```
apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 474A19E8
```

If the HTTP Keyserver Protocol (11371 TCP) is blocked, you can try an alternate command:

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

Add *Rudder* project repository:

```
echo "deb http://www.rudder-project.org/apt-3.0/ $(lsb_release -cs) main" > /etc/apt/ ↩
    sources.list.d/rudder.list
```

Update your local package database to retrieve the list of packages available on our repository:

```
sudo apt-get update
```

Install the `rudder-agent` package:

```
sudo apt-get install rudder-agent
```

3.3.2 Install Rudder Agent on RHEL-like systems

Define a yum repository for *Rudder*:

```
version=$(grep "release [0-9]" /etc/redhat-release | sed "s/^.*release \([0-9]\).*$/\1/")
echo "[Rudder_3.0]
name=Rudder 3.0 EL repository
baseurl=http://www.rudder-project.org/rpm-3.0/RHEL_${version}/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-3.0/RHEL_${version}/repodata/repomd.xml.key" > / ↩
    etc/yum.repos.d/rudder.repo
```

Tip
The RPM can be directly downloaded for a standalone installation, from the following URL: http://www.rudder-project.org/rpm-3.0/*RHEL_7*/ (or *RHEL_6*, *RHEL_5*, etc, depending on your host's OS version)

Install the package:

```
yum install rudder-agent
```

Or:

```
yum install rudder-agent-3.0.0-1.EL.7.x86_64.rpm
```

3.3.3 Install Rudder Agent on SLES

Following commands are executed as the `root` user.

Warning
You might not be able to install *Rudder* rpm files locally with Zypper (for example with *zypper install rudder-agent-version.release-1.SLES.11.x86_64.rpm*), due to a bug (bnc#929483 on *SuSE* bugtracker) in its RPM headers parsing causing a segmentation fault. You can either:

- Install the packages directly from the repository, as described below
- Upgrade your libzypp package to a version including the fix provided by *SuSE* (upgrade for SLES11SP3 and for SLES12)
- Use the rpm command to install packages locally (for example with *rpm -i rudder-agent-version.release-1.SLES.11.x86_64.rpm*)

Add the *Rudder* packages repository:

- on SLES 11:

```
zypper ar -n 'Rudder SLES repository' http://www.rudder-project.org/rpm-3.0/SLES_11_SP1/  ↩
    Rudder
```

- on SLES 10:

```
zypper sa 'http://www.rudder-project.org/rpm-3.0/SLES_10_SP3/' Rudder
```

Update your local package database to retrieve the list of packages available on our repository:

```
zypper ref
```

Install the `rudder-agent` package:

```
zypper install rudder-agent
```

Tip
The use the the `rug` package manager on SLES 10 is strongly discouraged, due to poor performance and possible stability issues.

3.3.4 Configure and validate

3.3.4.1 Configure Rudder Agent

Configure the IP address or hostname of the *Rudder Root Server* in the following file

```
echo '<rudder server ip or hostname>' > /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 trusted DNS infrastructure with proper reverse resolutions.

You can now start the *Rudder* service with:

```
service rudder-agent start
```

3.3.4.2 Validate new Node

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 to run and send an inventory by issuing the following command:

```
rudder agent inventory
```

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

```
rudder agent run
```

3.4 Install Rudder Relay (optional)

Relay servers can be added to *Rudder*, for example to manage a DMZ or to isolate specific nodes from the main environment for security reasons.

Relay server's purpose is to solve a simple problem: sometimes, one would want to manage multiple networks from *Rudder*, without having to allow all the subnet access to the other for security reasons. A solution for this would be to have a kind of "*Rudder*" proxy that would be relaying information between the subnet and the main *Rudder* server. This is the reason relay servers were created.

Using a relay, you are able to:

- Separate your *Rudder* architecture into separate entities that still report to one server
- Prevent laxist security exceptions to the *Rudder* server
- Ease maintenance

The first part is to be done on the machine that will become a relay server. The procedure will:

- Add the machine as a regular node
- Configure the relay components (Syslog, *Apache* HTTPd, *CFEngine*)
- Switch this node to the relay server role (from the root server point of view)

3.4.1 On the relay

To begin, please install a regular *Rudder* agent on the OS, following the installation instructions, and install the *rudder-server-relay* package in addition to the *rudder-agent* package.

To complete this step, please make sure that your node is configured successfully and appears in your *Rudder* web interface.

3.4.2 On the root server

You have to tell the *Rudder Root* server that a node will be a relay. To do so, launch the rudder-node-to-relay script on the root server, supplying the UUID of the host to be considered as a relay. You can find the UUID of your node in */opt/rudder/etc/uuid.hive*.

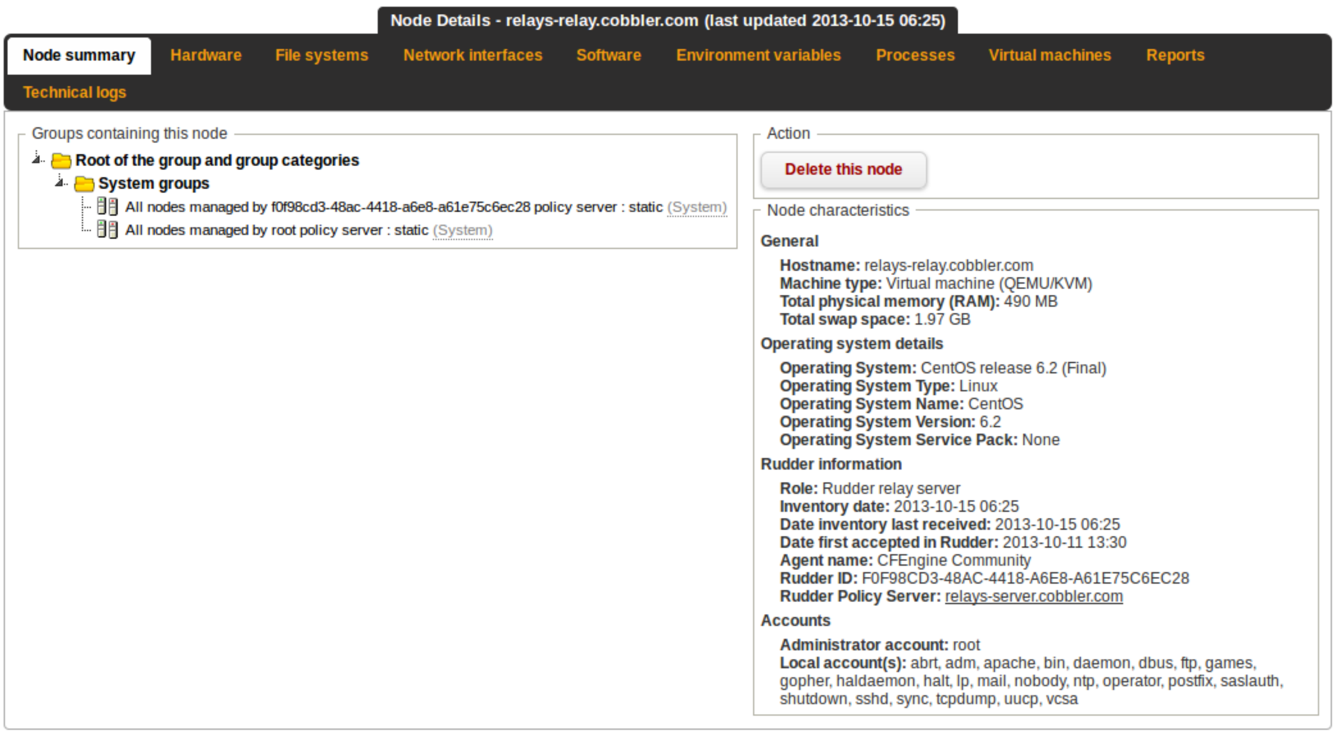
```
/opt/rudder/bin/rudder-node-to-relay aaaaaaaa-bbbb-cccc-dddd-eeeeeeee
```

3.4.3 Validation

When every step has completed successfully:

- The *Rudder* root server will recognize the new node as a relay
- It will generate specific promises for the relay
- The relay will update and switch to his new role

This is an example of node details pane showing a relay server. Note the "Role: *Rudder* relay server" part that shows that the machine has successfully changed from a node to a relay.



Root of the group and group categories
System groups
All nodes managed by f0f98cd3-48ac-4418-a6e8-a61e75c6ec28 policy server : static (System)
All nodes managed by root policy server : static (System)

General
Hostname: relays-relay.cobbler.com
Machine type: Virtual machine (QEMU/KVM)
Total physical memory (RAM): 490 MB
Total swap space: 1.97 GB
Operating system details
Operating System: CentOS release 6.2 (Final)
Operating System Type: Linux
Operating System Name: CentOS
Operating System Version: 6.2
Operating System Service Pack: None
Rudder information
Role: Rudder relay server
Inventory date: 2013-10-15 06:25
Date inventory last received: 2013-10-15 06:25
Date first accepted in Rudder: 2013-10-11 13:30
Agent name: CFEngine Community
Rudder ID: F0F98CD3-48AC-4418-A6E8-A61E75C6EC28
Rudder Policy Server: relays-server.cobbler.com
Accounts
Administrator account: root
Local account(s): abrt, adm, apache, bin, daemon, dbus, ftp, games, gopher, haldaemon, halt, lp, mail, nobody, ntp, operator, postfix, saslauth, shutdown, sshd, sync, tcpdump, uucp, vcsa

Figure 3.1: Rudder relay node

3.4.4 Adding nodes to a relay server

When you have at least one relay, you will likely want to add nodes on it.

You then have two possible cases:

- You want to switch an already existing node to the relay
- You want to add a new one

The procedure on both cases is the same, you have to:

- Create / update the file /var/rudder/cfengine-community/policy_server.dat with the IP address or the fully qualified domain name of the relay server (instead of the root server)

```
echo "rudder-relay.example.com" > /var/rudder/cfengine-community/policy_server.dat
```

- Trigger an inventory immediately to make sure the node is registered correctly

```
rudder agent inventory
```

After those steps, the node should be registered correctly on your *Rudder* infrastructure.

Chapter 4

Upgrade

This short chapter covers the upgrade of the *Rudder Server* Root and *Rudder* Agent from older versions to the latest version, 3.0.

Please note that you can upgrade directly from *Rudder* 2.10.x or 2.11.x to *Rudder* 3.0. However, direct upgrades from 2.9.x and older are no longer supported. If you are still running one of those, please first upgrade to one of the supported versions above.

The upgrade is quite similar to the installation.

A big effort has been made to ensure that all upgrade steps are performed automatically by packaging scripts. Therefore, you shouldn't have to do any upgrade procedures manually, but you will note that several data migrations occur during the upgrade process.

4.1 Caution cases

4.1.1 Upgrading from Rudder 2.11

Rudder 2.11.* use a compatible *CFEngine* version. Therefore, *Rudder* agent 2.11.* are fully compatible with *Rudder* server 3.0.**, so it is not necessary to update your agents to 3.0.**.

To have a successful upgrade, you only need to upgrade *Rudder* server to 3.0.

4.1.2 Upgrading from Rudder 2.6 or 2.10

Rudder 2.6.* and 2.10.* contain an older *CFEngine* version. To have a successful upgrade these steps should be followed:

- Make sure the *Rudder* server to be upgraded to the latest 2.6 or 2.10 version before attempting to upgrade to 3.0.*
- Ensure that all node's promises have been regenerated since the server upgrade to this version (don't forget that your *Techniques* will not be upgraded automatically, follow the Technique Upgrade section to upgrade them manually)
  - On *Rudder* WebUI, at the top right of the screen, click on *Regenerate now*
  - You can use the API from the server with this command: *curl http://localhost/rudder/api/deploy/reload*
- Upgrade all agents connected to that server to 3.0.*.
- Upgrade the *Rudder* server to 3.0

4.1.3 Known issues

- After upgrade, if the web interface has display problems, empty your navigator cache and/or logout/login.

4.2 On Debian or Ubuntu

Following commands are executed as the `root` user.

Add the *Rudder* project repository:

```
echo "deb http://www.rudder-project.org/apt-3.0/ $(lsb_release -cs) main" > /etc/apt/ ↩
    sources.list.d/rudder.list
```

Update your local package database to retrieve the list of packages available on our repository:

```
apt-get update
```

For *Rudder Server*, upgrade all the packages associated to `rudder-server-root`:

- With apt-get:

```
apt-get install rudder-server-root ncf ncf-api-virtualenv
```

and after the upgrade of these packages, restart jetty to apply the changes on the Web application:

```
service rudder-jetty restart
```

For *Rudder* Agent, upgrade the `rudder-agent` package:

```
apt-get install rudder-agent
```

Warning
Rudder includes a script for upgrading all files, databases, etc… which need migrating. Therefore, you should not replace your old files by the new ones when apt-get/aptitude asks about this, unless you want to reset all your parameters.

You can now upgrade your local techniques.

4.3 On RHEL or CentOS

Following commands are executed as the `root` user.

Update your yum repository:

```
$ echo "[Rudder_3.0]
name=Rudder 3.0 Repository
baseurl=http://www.rudder-project.org/rpm-3.0/RHEL_7/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-3.0/RHEL_7/repodata/repomd.xml.key" > /etc/yum. ↩
    repos.d/rudder.repo
```

Tip
Replace *RHEL_7* with your *Enterprise* Linux version if necessary.

For *Rudder* server, upgrade the `rudder-*` and `ncf`-related packages:

```
yum update "rudder-*" ncf ncf-api-virtualenv
```

and after the upgrade of these packages, restart jetty to apply the changes on the Web application:

```
service rudder-jetty restart
```

For *Rudder* agent, upgrade the `rudder-agent` package:

```
yum update rudder-agent
```

You can now upgrade your local techniques.

4.4 On SLES

Following commands are executed as the `root` user.

Add the *Rudder* packages repository:

- On a SLES 11 system:

```
zypper ar -n "Rudder SLES repository" http://www.rudder-project.org/rpm-3.0/SLES_11_SP1/  ↩
    Rudder
```

- On a SLES 10 system:

```
zypper sa "http://www.rudder-project.org/rpm-3.0/SLES_10_SP3/" Rudder
```

Update your local package database to retrieve the list of packages available on our repository:

```
zypper ref
```

For *Rudder Server*, upgrade all the packages associated to `rudder-server-root`:

```
zypper update "rudder-*" "ncf*"
```

Warning
SLES 11 uses PostgreSQL 8.3 by default, which is not supported by *Rudder* and will cause various glitches in the web interface, as well as reporting failures.
It is really mandatory to migrate to PostgreSQL 8.4 at least (9.1+ is recommended).
Please look at Install Rudder Root server on SLES for details.

and after the upgrade of these packages, restart jetty to apply the changes on the Web application:

```
service rudder-jetty restart
```

For *Rudder* Agent, upgrade the `rudder-agent` package:

```
zypper update rudder-agent
```

You can now upgrade your local techniques.

4.5 Technique upgrade

At the first installation, *Rudder* will automatically deploy a *Technique* library in the `/var/rudder/configuration-repository/techniques` directory.

When upgrading *Rudder* to another version, a new (updated) *Technique* library will be deployed in `/opt/rudder/share/techniques`, and *Rudder* will automatically take care of updating the system *Techniques* in the configuration-repository directory.

However, the other *Techniques* will not be updated automatically (yet), so you will have to do it yourself.

Caution
Please keep in mind that if you did manual modifications on the *Techniques* in existing directories, or created new versions of them, you will have some merging work to do.

To upgrade you local techniques, run the following commands on the *Rudder Root Server*:

```
root@node:~# cd /var/rudder/configuration-repository
root@node:~# cp -a /opt/rudder/share/techniques/* techniques/
root@node:~# git status
#~Now, inspect the differences. If no conflict is noticeable, then go ahead.
root@node:~# git add techniques/
root@node:~# git commit -m "Technique upgrade" # Here, put a meaningful message about why ↩
    you are updating.
```

After the commit has been validated by git, please go to the *Rudder* web interface, to the Administration tab, Policy Server tab, and click on "Reload *Techniques*". It will reload the *Technique* library and trigger a full redeployment on nodes.

Please check that the deployment is successful before logging out.

Chapter 5

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.

5.1 Authentication

When accessing the *Rudder* web interface, a login / password is required. The default account is "admin" (Password: admin).

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

5.2 Presentation of Rudder Web Interface

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

5.2.1 Rudder Home

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

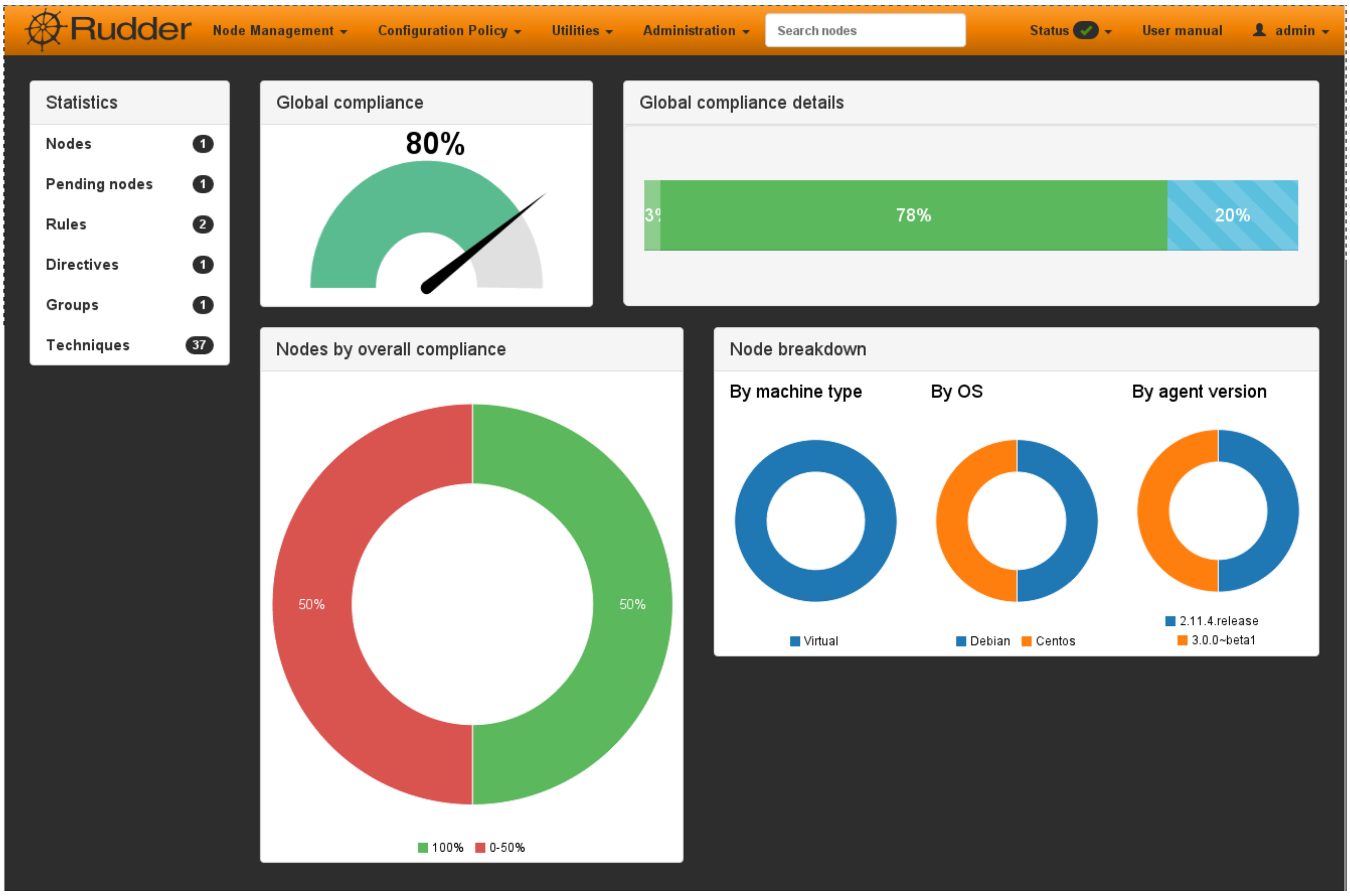
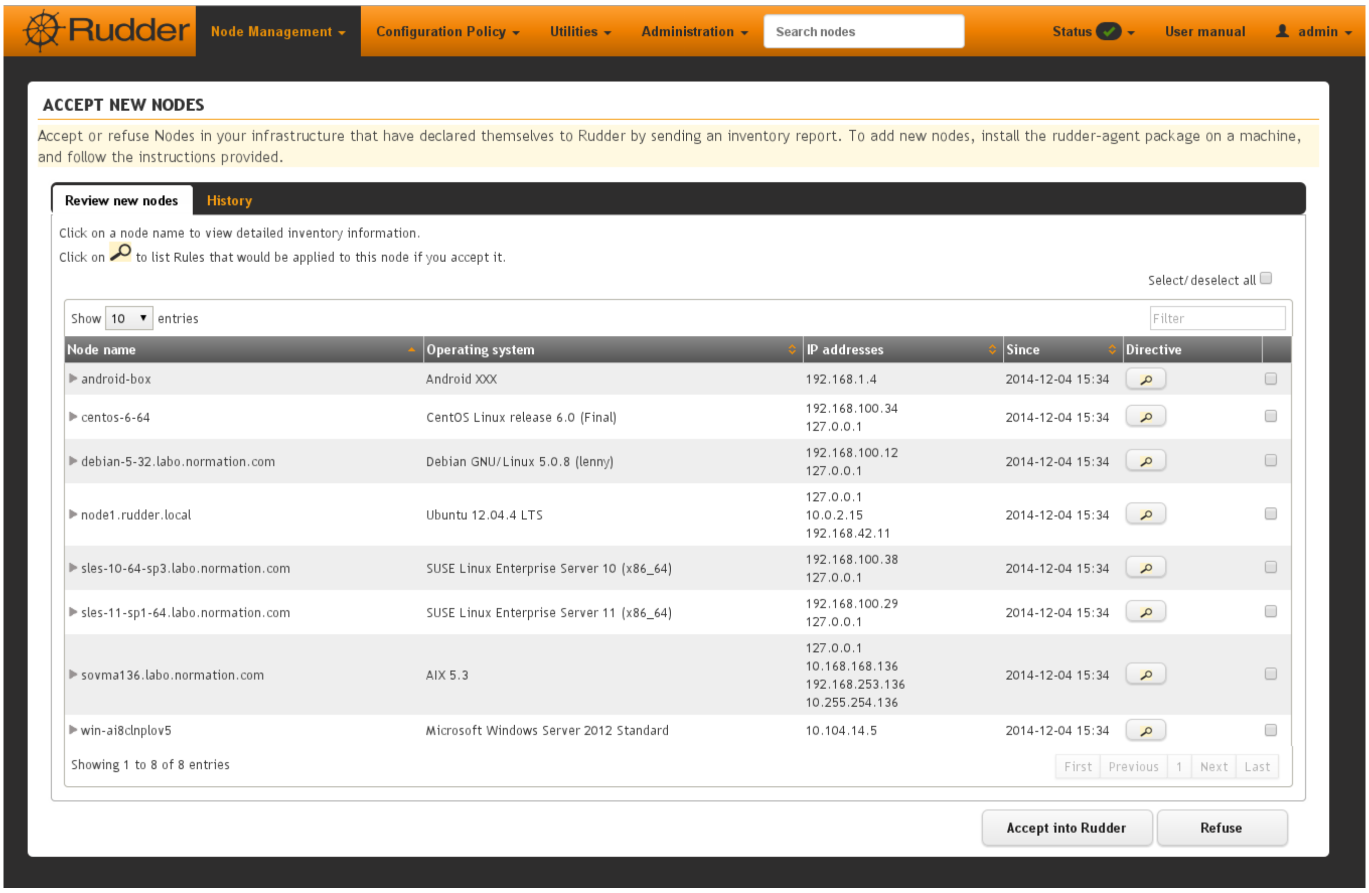


Figure 5.1: Rudder Homepage

5.2.2 Node Management

In the *Node* 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*.



ACCEPT NEW NODES

Accept or refuse Nodes in your infrastructure that have declared themselves to Rudder by sending an inventory report. To add new nodes, install the rudder-agent package on a machine, and follow the instructions provided.

Review new nodes | History

Click on a node name to view detailed inventory information.
Click on to list Rules that would be applied to this node if you accept it.

Select/deselect all

Show 10 entries

Node name	Operating system	IP addresses	Since	Directive	
android-box	Android XXX	192.168.1.4	2014-12-04 15:34		
centos-6-64	CentOS Linux release 6.0 (Final)	192.168.100.34 127.0.0.1	2014-12-04 15:34		
debian-5-32.labo.normation.com	Debian GNU/Linux 5.0.8 (lenny)	192.168.100.12 127.0.0.1	2014-12-04 15:34		
node1.rudder.local	Ubuntu 12.04.4 LTS	127.0.0.1 10.0.2.15 192.168.42.11	2014-12-04 15:34		
sles-10-64-sp3.labo.normation.com	SUSE Linux Enterprise Server 10 (x86_64)	192.168.100.38 127.0.0.1	2014-12-04 15:34		
sles-11-sp1-64.labo.normation.com	SUSE Linux Enterprise Server 11 (x86_64)	192.168.100.29 127.0.0.1	2014-12-04 15:34		
sovma136.labo.normation.com	AIX 5.3	127.0.0.1 10.168.168.136 192.168.253.136 10.255.254.136	2014-12-04 15:34		
win-ai8clnplov5	Microsoft Windows Server 2012 Standard	10.104.14.5	2014-12-04 15:34		

Showing 1 to 8 of 8 entries

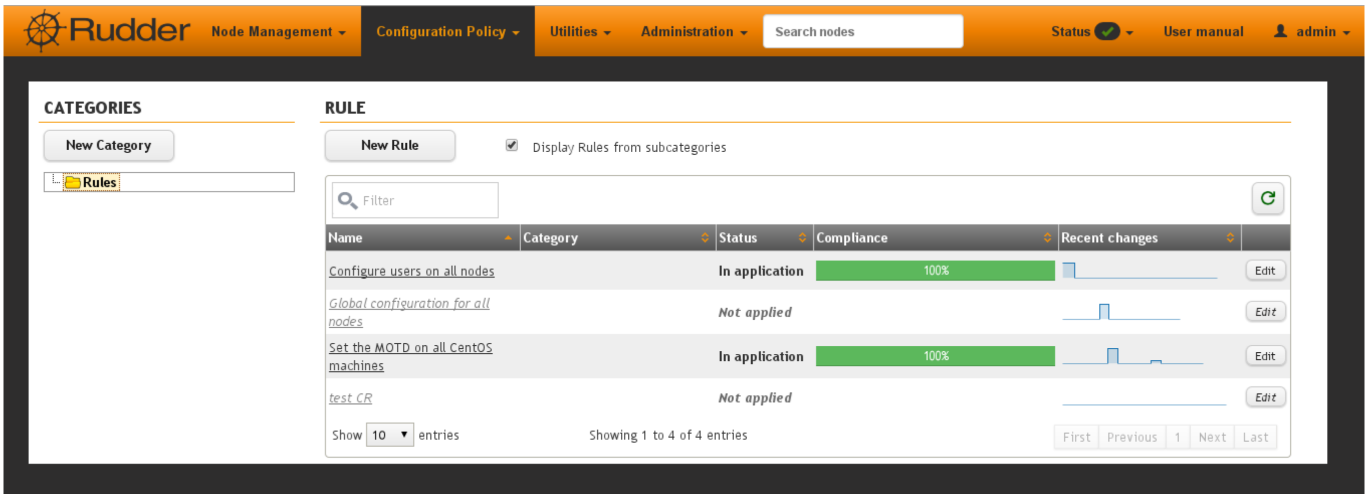
First Previous 1 Next Last

Accept into Rudder | Refuse

Figure 5.2: Accept new nodes screen

5.2.3 Configuration Management

In the Configuration Management section, you can select the *Techniques*, configure the *Directives* and manage the *Rules*.



CATEGORIES

New Category

Rules

RULE

New Rule

Display Rules from subcategories

Filter

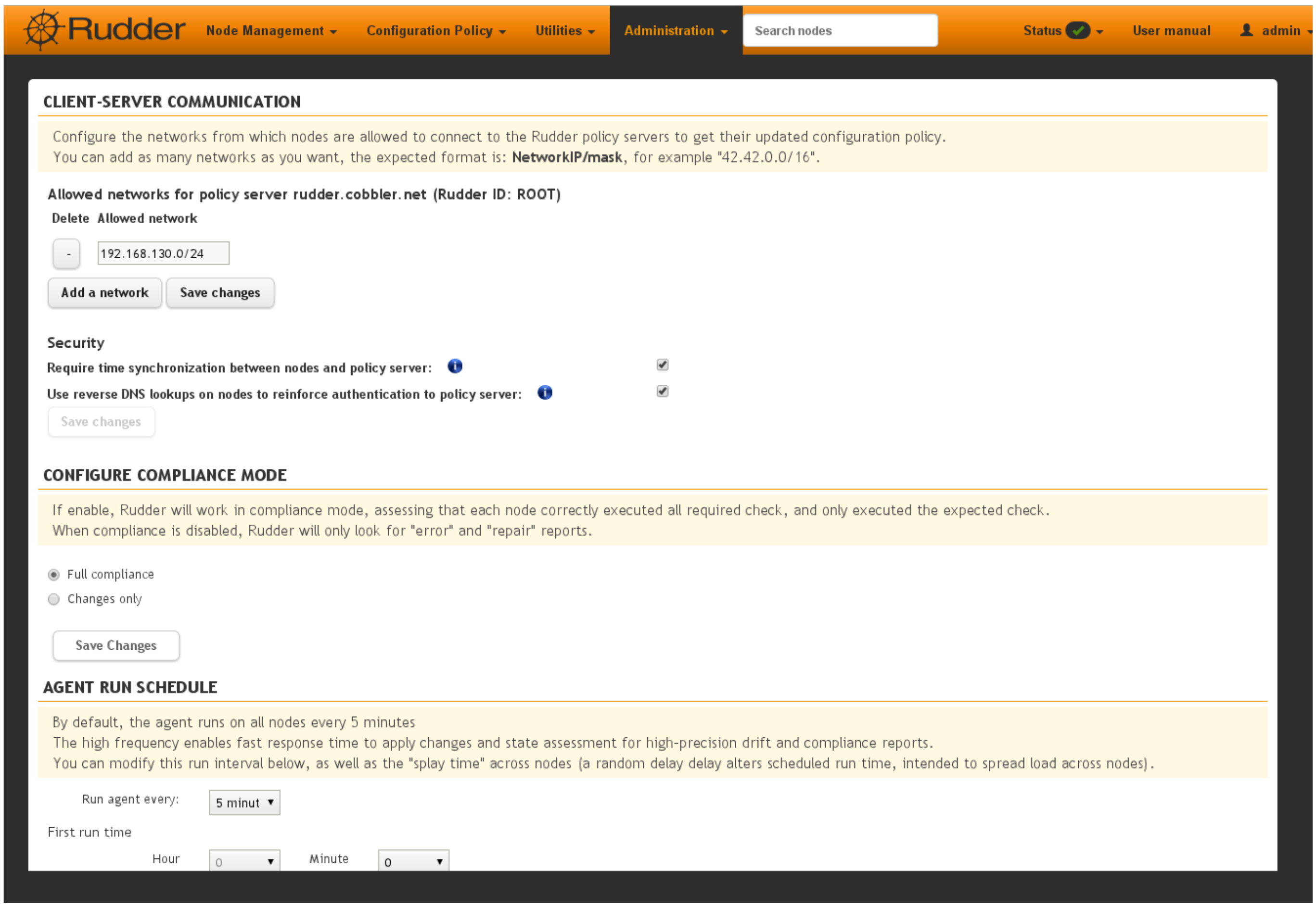
Name	Category	Status	Compliance	Recent changes	
Configure users on all nodes		In application	100%		Edit
Global configuration for all nodes		*Not applied*			*Edit*
Set the MOTD on all CentOS machines		In application	100%		Edit
test CR		*Not applied*			*Edit*

Show 10 entries Showing 1 to 4 of 4 entries First Previous 1 Next Last

Figure 5.3: Configuration Policy (Rules) screen

5.2.4 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.



CLIENT-SERVER COMMUNICATION

Configure the networks from which nodes are allowed to connect to the Rudder policy servers to get their updated configuration policy.
You can add as many networks as you want, the expected format is: **NetworkIP/mask**, for example "42.42.0.0/16".

Allowed networks for policy server rudder.cobbler.net (Rudder ID: ROOT)

Delete	Allowed network
-	192.168.130.0/24

Add a network | Save changes

Security

Require time synchronization between nodes and policy server: ✓

Use reverse DNS lookups on nodes to reinforce authentication to policy server: ✓

Save changes

CONFIGURE COMPLIANCE MODE

If enable, Rudder will work in compliance mode, assessing that each node correctly executed all required check, and only executed the expected check.
When compliance is disabled, Rudder will only look for "error" and "repair" reports.

[✓ Full compliance]
Changes only

Save Changes

AGENT RUN SCHEDULE

By default, the agent runs on all nodes every 5 minutes
The high frequency enables fast response time to apply changes and state assessment for high-precision drift and compliance reports.
You can modify this run interval below, as well as the "splay time" across nodes (a random delay delay alters scheduled run time, intended to spread load across nodes).

Run agent every: 5 minut

First run time

Hour 0 Minute 0

Figure 5.4: Settings screen

5.3 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.

5.3.1 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.

5.3.2 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
```

5.3.3 Supported units

Units used are non binary units, and a multiplication 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 5.1: Units supported by Rudder search engine

Chapter 6

Node Management

6.1 Node Inventory

Rudder integrates a node inventory tool which harvest useful information about the nodes. This information is used by *Rudder* to handle the nodes, and you can use the inventory information for Configuration Management purposes: 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 information about this *Node*. The inventory is organized as following: first tab is a *summary* of administrative information about the *Node*; other tabs are specialized for *hardware*, *network* interfaces, and *software* for every *Node*; tabs for *reports* and *logs* are added on *Rudder* managed *Nodes*.

The Node *Summary* presents administrative information 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 information is displayed like the *Node Name* and the *Date first accepted in* Rudder.

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

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

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

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

6.2 Accept new Nodes

At the starting point, the *Rudder Server* doesn't know anything about the *Nodes*. After the installation of the *Rudder* Agent, each *Node* registers 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 ***Node* 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 6.1 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 ***Node* 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.

6.3 Search Nodes

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

6.3.1 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* information. This search tool can be very useful to help you create a new search in the Advanced Search.

Example 6.2 Quick search the Node called `debian-node`

Assuming you have one managed *Node* called `debian-node.rudder-project.org`, which ID in *Rudder* is `d06b1c6c-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-4e5e-8049-d55f769ac33f [d06b1c6c-f59b-4e5e-8049-d55f769ac33f]`.

6.3.2 Advanced Search

In the Advanced Search tool, you can create complex searches based on *Node Inventory* information. 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*, *Environment Variable*, *Processes*, *Virtual Machines*; 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*, =, $\neq$ or *Regex* 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.

- a. Regex matching rule

You can use regular expressions to find whatever you want in *Node* inventories. A search request using a regexp will look for every node that match the pattern you entered.

Those regexps follow *Java* Pattern rules. See http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html for more details.

Example 6.3 Search node having an ip address matching `192.168.x.y`

Assuming you want to search every node using an ip address match 192.168.x.y, where x<10 and y could be everything. You will to add that line to your search request:

- Node *summary*, *Ip address*, `Regex`, *192\.168\.\d\..**

- b. Composite search

Some fields allow you to look for more than one piece of information at a time. That's the case for environment variable. For those fields you have to enter the first element then the separator then following elements. The name of the fields tells you about what is expected. It would look like `firstelement<sep>secondelement` assuming that <sep> is the separator.

Example 6.4 Search Environment Variable `LANG=C`.

Assuming you want to search every node having the environment variable LANG set to C. You will have to add that search line to your request:

- *Environment variable*, *key=value*, =, *LANG=C*.

- 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 6.5 Advanced search for Linux Nodes with `ssh`.

Assuming you want to search all 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`.

6.4 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 Node *Management* > *Search* Nodes page, or directly from the *Groups* list in Node *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.

Example 6.6 Create a dynamic group for Linux Nodes with `ssh` having an ip address in 192.18.42.x.

To create that dynamic group like described above, You first have to create a new group with group type set to `Dynamic`. Then you have to set its search request to:

1. Operator: `AND`.
2. First search line: Node, *Operating System*, =, *Linux*.
3. Second search line: *Software*, *Name*, =, `ssh`.
4. Third search line: Node *summary*, *Ip address*, `Regex`, *192\.168\.\d\..** .

Finally, you have to click on Search to populate the group and click on Save to actually save it.

Chapter 7

Configuration Management

7.1 Techniques

7.1.1 Concepts

A *Technique* 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 *Techniques* 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 *Techniques* 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.

7.1.2 Manage the Techniques

The *Techniques* shipped with *Rudder* are presented in a library that you can reorganize in **Configuration > *Techniques***. The library is organized in two parts: the available *Techniques*, and the selection made by the user.

***Technique* Library** This is an organized list of all available *Techniques*. This list can't be modified: every change made by a user will be applied to the Active *Techniques*.

***Active* Techniques** This is an organized list of the *Techniques* selected and modified by the user. By default this list is the same as the *Technique* Library. *Techniques* 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 *Technique* can appear only once in the *Active* Techniques list.

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

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

7.1.3 Available Techniques

7.1.3.1 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 *RHEL*.

7.1.3.2 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.

7.1.3.3 File state configuration

Set the permissions of files Set the permissions of files

7.1.3.4 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.

7.1.3.5 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

7.1.3.6 System settings: Process

Process Management Enforce defined parameters on system processes

7.1.3.7 System settings: Remote access

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

7.1.3.8 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.

7.2 Directives

Once you have selected and organized your *Techniques*, you can create your configurations in the **Configuration Management > *Directives*** section.

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

The screen is divided in three parts:

- on the left, your list of *Techniques* and *Directives*,
- on the right the description of the selected *Technique* or *Directive*.
- at the bottom, the configuration items of the selected *Directive*.

Click on the name of a *Technique* to show its description.

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

Example 7.1 Create a Directive for Name resolution

Use the *Technique Name resolution* to create a new *Directive* 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`.

7.3 Rules

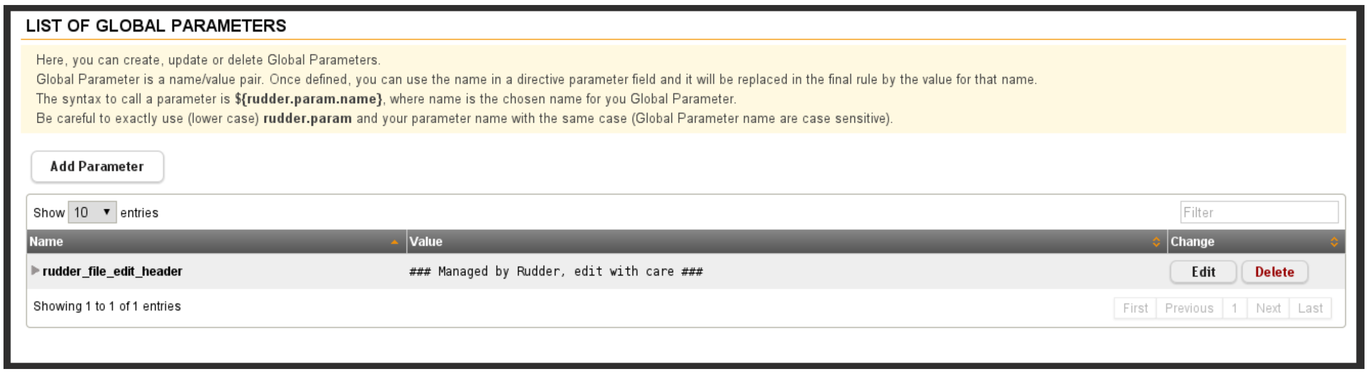
Rule It is the application of one or more directives to a group of nodes. It is the glue between both Asset Management and Configuration Management parts of the application.

When a *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 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*)

7.4 Variables

7.4.1 User defined parameters

Rudder provides a simple way to add common and reusable variables in either plain *Directives*, or techniques created using the *Technique* editor: the parameters.


LIST OF GLOBAL PARAMETERS

Here, you can create, update or delete Global Parameters.
Global Parameter is a name/value pair. Once defined, you can use the name in a directive parameter field and it will be replaced in the final rule by the value for that name.
The syntax to call a parameter is **${rudder.param.name}**, where name is the chosen name for you Global Parameter.
Be careful to exactly use (lower case) **rudder.param** and your parameter name with the same case (Global Parameter name are case sensitive).

Name	Value	Change
rudder_file_edit_header	### Managed by Rudder, edit with care ###	Edit Delete

Showing 1 to 1 of 1 entries

The parameters enable the user to specify a content that can be put anywhere, using the following syntax:

- In *Directives*: *${rudder.param.name}* will expand the content of the "name" parameter.
- In the *Technique* Editor: *${rudder_parameters.name}* will do the same.

Using this, you can specify common file headers (this is the default parameter, "rudder_file_edit_header"), common DNS or domain names, backup servers, site-specific elements. . .

7.4.2 System variables

Rudder also provides system variables that contain information about nodes and their policy server. You can use them like user defined parameters.

The information about a *Node*:

- *${rudder.node.id}* returns the *Rudder* generated id of the *Node*
- *${rudder.node.hostname}* returns the hostname of the *Node*
- *${rudder.node.admin}* returns the administrator login of the *Node*

The information about a *Node*'s policy server.

- *${rudder.node.policyserver.id}* returns the *Rudder* generated id of the Policy Server
- *${rudder.node.policyserver.hostname}* returns the hostname of the Policy Server
- *${rudder.node.policyserver.admin}* returns the administrator login of the Policy Server

7.5 Compliance

A *Directive* contains one or multiple components. Each component generates one or multiple reports, based on the number of keys in this component. For example, for a Sudoers *Directive*, 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.

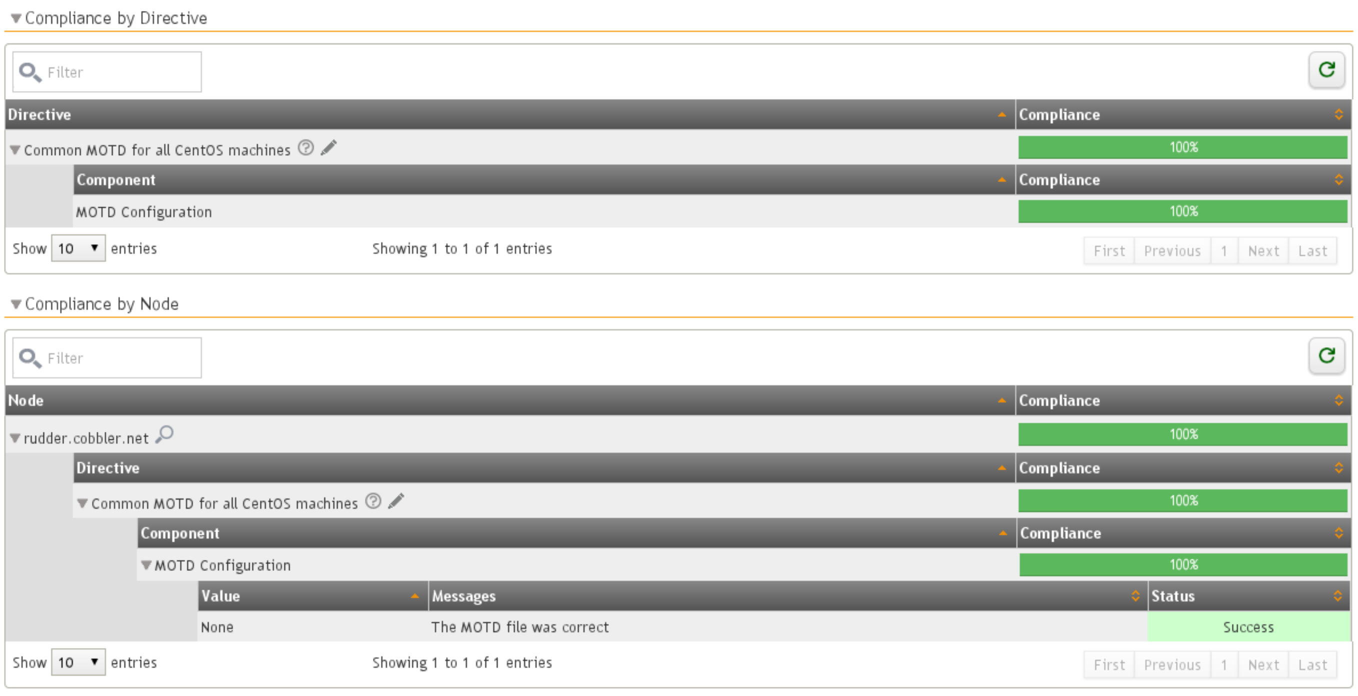
Applying When a *Directive* is applied, *Rudder* waits during 10 minutes for a report. During this period, the *Directive* is said *Applying*.

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

A *Directive* has gained conformity on a *Node* if every report for each component, for each key, is in *Success* state. This is the only condition.

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

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


▼Compliance by Directive

Directive	Compliance
▼Common MOTD for all CentOS machines	100%

Component	Compliance
MOTD Configuration	100%

Show 10 entries Showing 1 to 1 of 1 entries

▼Compliance by Node

Node	Compliance
▼rudder.cobbler.net	100%

Directive	Compliance
▼Common MOTD for all CentOS machines	100%

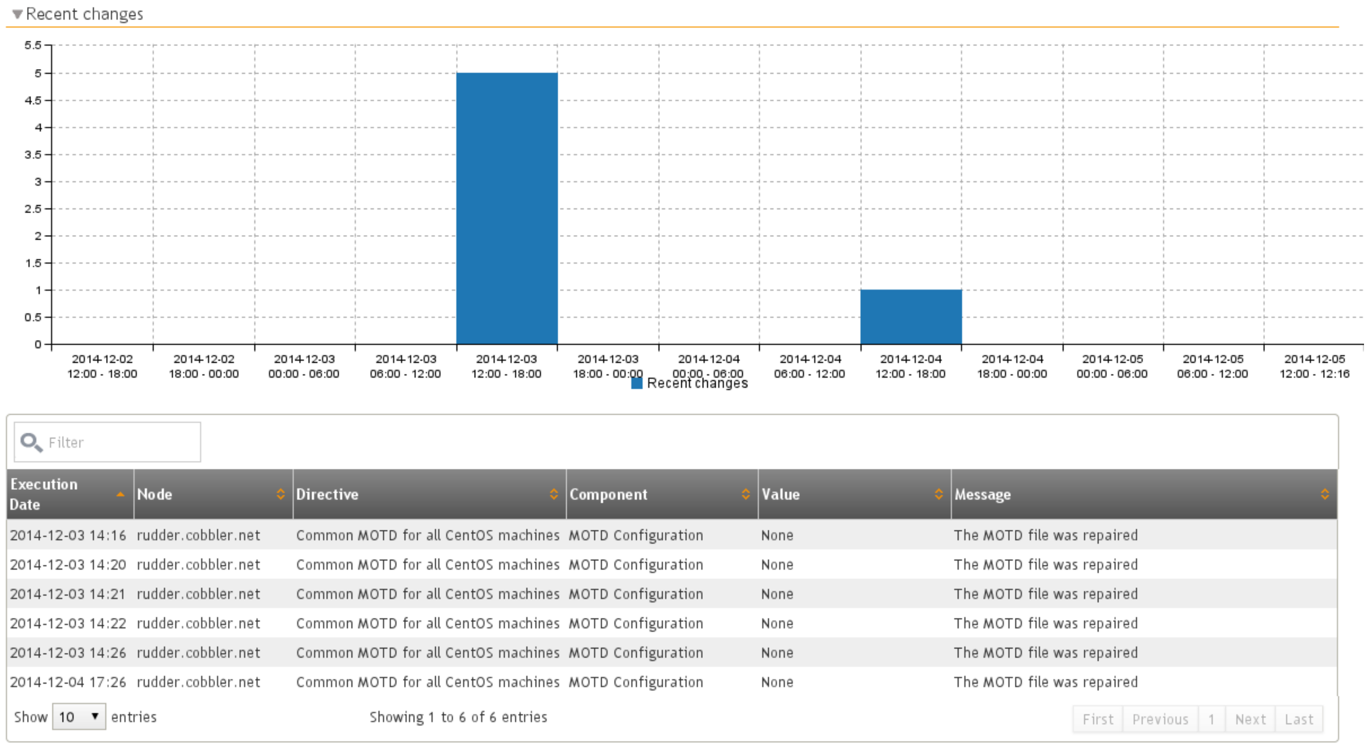
Component	Compliance
▼MOTD Configuration	100%

Value	Messages	Status
None	The MOTD file was correct	Success

Show 10 entries Showing 1 to 1 of 1 entries

Figure 7.1: Compliance on a Rule

The *Rule* detailed compliance screen will also graph compliance deviations on a recent period as well as display a deviation log history for this period.



Execution Date	Node	Directive	Component	Value	Message
2014-12-03 14:16	rudder.cobbler.net	Common MOTD for all CentOS machines	MOTD Configuration	None	The MOTD file was repaired
2014-12-03 14:20	rudder.cobbler.net	Common MOTD for all CentOS machines	MOTD Configuration	None	The MOTD file was repaired
2014-12-03 14:21	rudder.cobbler.net	Common MOTD for all CentOS machines	MOTD Configuration	None	The MOTD file was repaired
2014-12-03 14:22	rudder.cobbler.net	Common MOTD for all CentOS machines	MOTD Configuration	None	The MOTD file was repaired
2014-12-03 14:26	rudder.cobbler.net	Common MOTD for all CentOS machines	MOTD Configuration	None	The MOTD file was repaired
2014-12-04 17:26	rudder.cobbler.net	Common MOTD for all CentOS machines	MOTD Configuration	None	The MOTD file was repaired

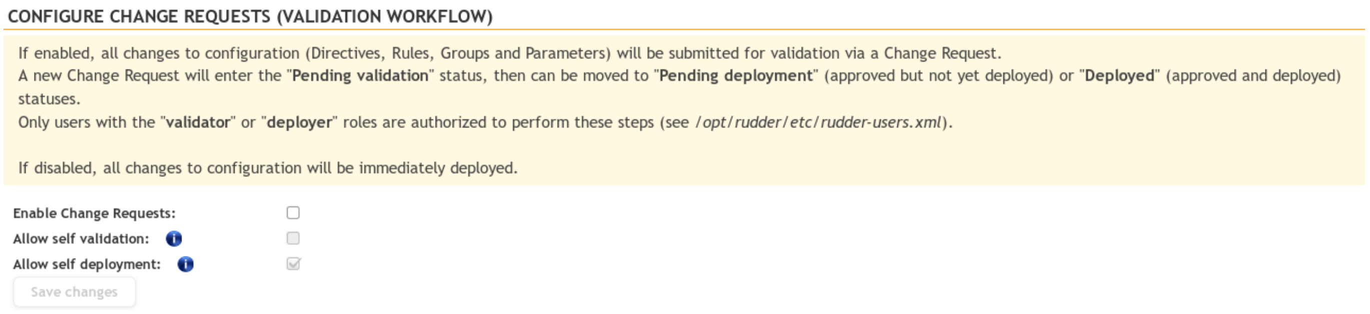
Figure 7.2: Compliance history on a Rule

7.6 Validation workflow in Rudder

The validation workflow is a feature whose purpose is to hold any change (*Rule*, *Directive*, *Group*) made by users in the web interface, to be reviewed first by other users with the adequate privileges before actual deployment.

The goal is to improve safety and knowledge sharing in the team that is using *Rudder*.

To enable it, you only have to tick "Enable Change Requests" in the Administration - Settings tab of the web interface. (This feature is optional and can be disabled at any time without any problem, besides risking the invalidation of yet-unapproved changes)


CONFIGURE CHANGE REQUESTS (VALIDATION WORKFLOW)

If enabled, all changes to configuration (Directives, Rules, Groups and Parameters) will be submitted for validation via a Change Request.
A new Change Request will enter the "**Pending validation**" status, then can be moved to "**Pending deployment**" (approved but not yet deployed) or "**Deployed**" (approved and deployed) statuses.
Only users with the "**validator**" or "**deployer**" roles are authorized to perform these steps (see */opt/rudder/etc/rudder-users.xml*).

If disabled, all changes to configuration will be immediately deployed.

Enable Change Requests:	☐
Allow self validation:	☐
Allow self deployment:	✓

7.6.1 What is a Change request ?

A Change request represents a modification of a *Rule*/*Directive*/*Group* from an old state to a new one. The Change is not saved and applied by the configuration, before that, it needs to be reviewed and approved by other members of the team.

A Change request has:

- An Id (an integer > 0)
- A title.
- A description.
- A creator.
- A status.
- Its own history.

This information can be updated on the change request detail page. For now, a Change request is linked to one change at a time.

7.6.1.1 Change request status

There is 4 Change request status:

Pending validation

- The change has to be reviewed and validated.
- Can be send to: Pending deployment, Deployed, Cancelled.

Pending deployment

- The change was validated, but now require to be deployed.
- Can be send to: Deployed, Cancelled.

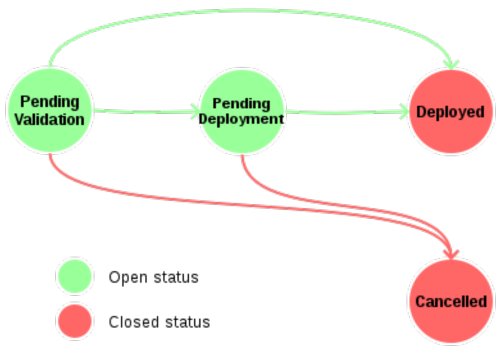
Deployed

- The change is deployed.
- This is a final state, it can't be moved anymore.

Cancelled

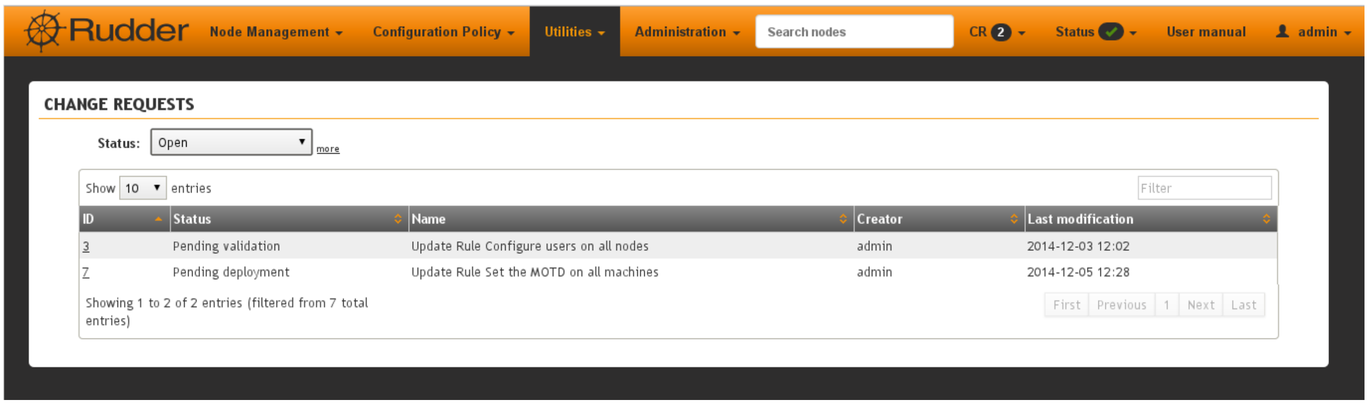
- The change was not approved.
- This is a final state, it can't be moved anymore.

Here is a diagram about all those states and transitions:



7.6.1.2 Change request management page

All Change requests can be seen on the /secure/utilities/changeRequests page. There is a table containing all requests, you can access to each of them by clicking on their id. You can filter change requests by status and only display what you need.



CHANGE REQUESTS

Status: Open

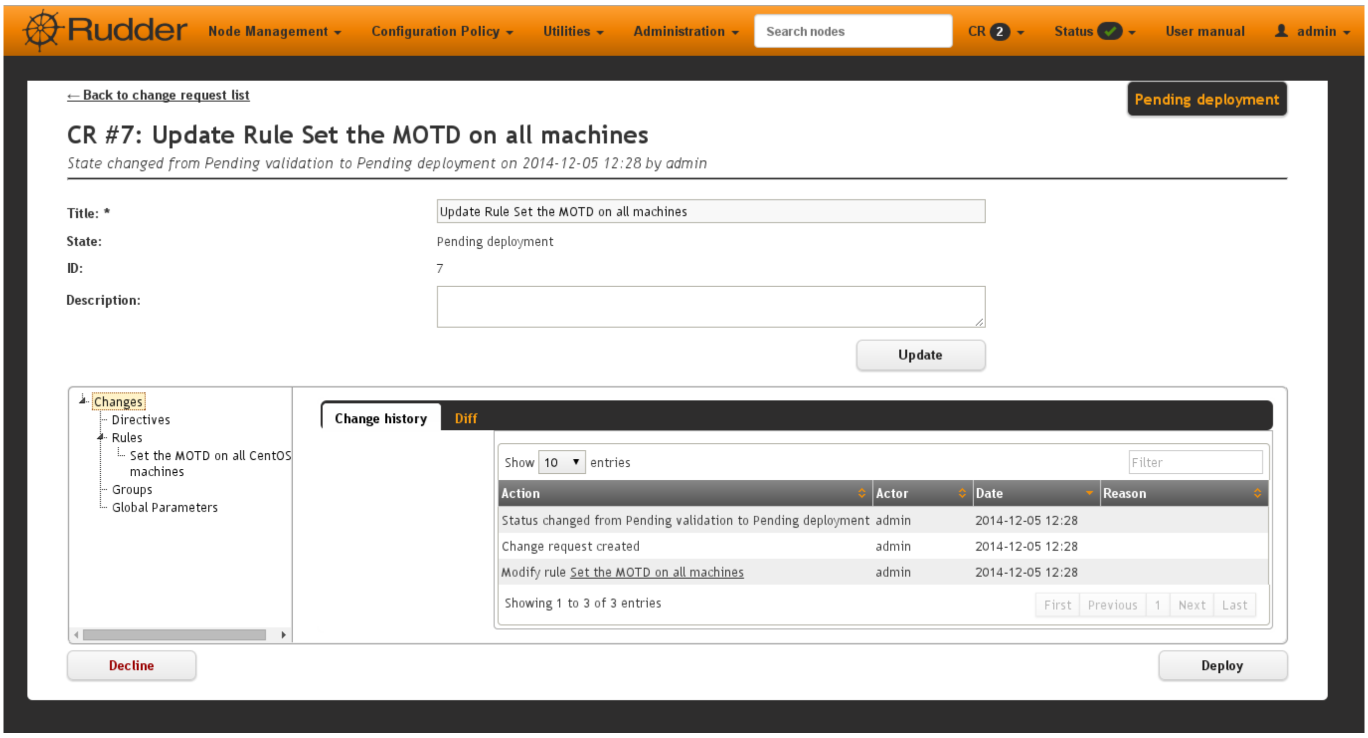
Show 10 entries

ID	Status	Name	Creator	Last modification
3	Pending validation	Update Rule Configure users on all nodes	admin	2014-12-03 12:02
7	Pending deployment	Update Rule Set the MOTD on all machines	admin	2014-12-05 12:28

Showing 1 to 2 of 2 entries (filtered from 7 total entries)

7.6.1.3 Change request detail page

Each Change request is reachable on the /secure/utilities/changeRequest/id.



CR #7: Update Rule Set the MOTD on all machines

State changed from Pending validation to Pending deployment on 2014-12-05 12:28 by admin

Title: *	Update Rule Set the MOTD on all machines
State:	Pending deployment
ID:	7
Description:	

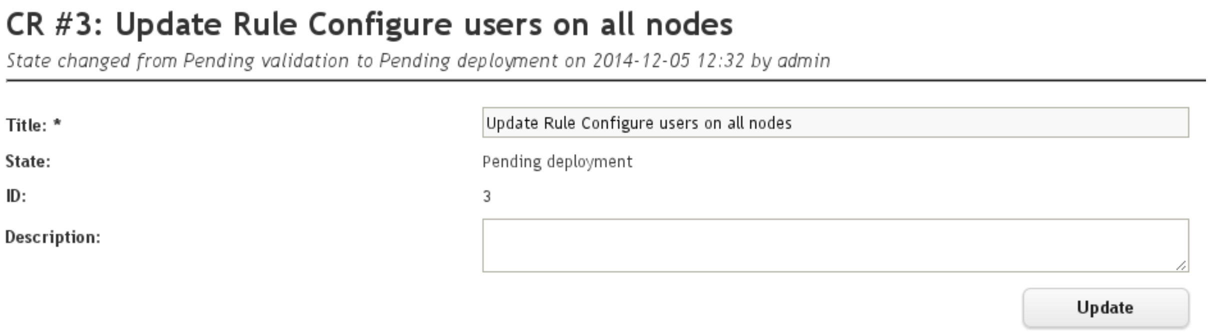
Show 10 entries

Action	Actor	Date	Reason
Status changed from Pending validation to Pending deployment	admin	2014-12-05 12:28	
Change request created	admin	2014-12-05 12:28	
Modify rule Set the MOTD on all machines	admin	2014-12-05 12:28	

Showing 1 to 3 of 3 entries

The page is divided into two sections:

Change request information display common information (title, description, status, id) and a form to edit them.



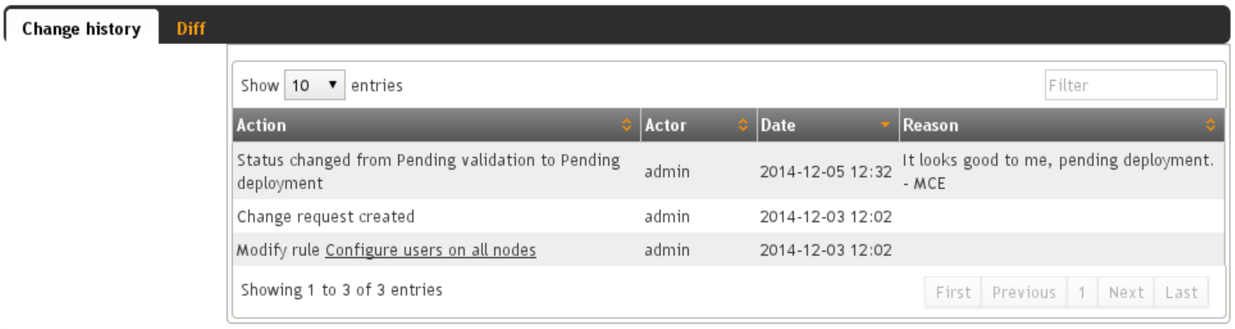
CR #3: Update Rule Configure users on all nodes

State changed from Pending validation to Pending deployment on 2014-12-05 12:32 by admin

Title: *	Update Rule Configure users on all nodes
State:	Pending deployment
ID:	3
Description:	

Change request content In this section, there is two tabs:

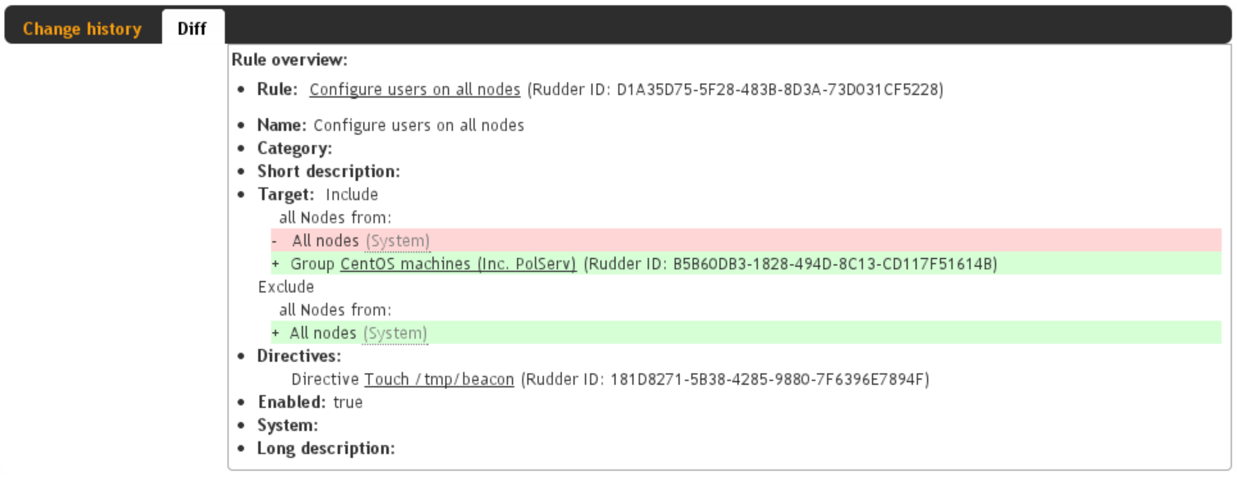
- History about that change request



Action	Actor	Date	Reason
Status changed from Pending validation to Pending deployment	admin	2014-12-05 12:32	It looks good to me, pending deployment. - MCE
Change request created	admin	2014-12-03 12:02	
Modify rule Configure users on all nodes	admin	2014-12-03 12:02	

Showing 1 to 3 of 3 entries

- Display the change proposed



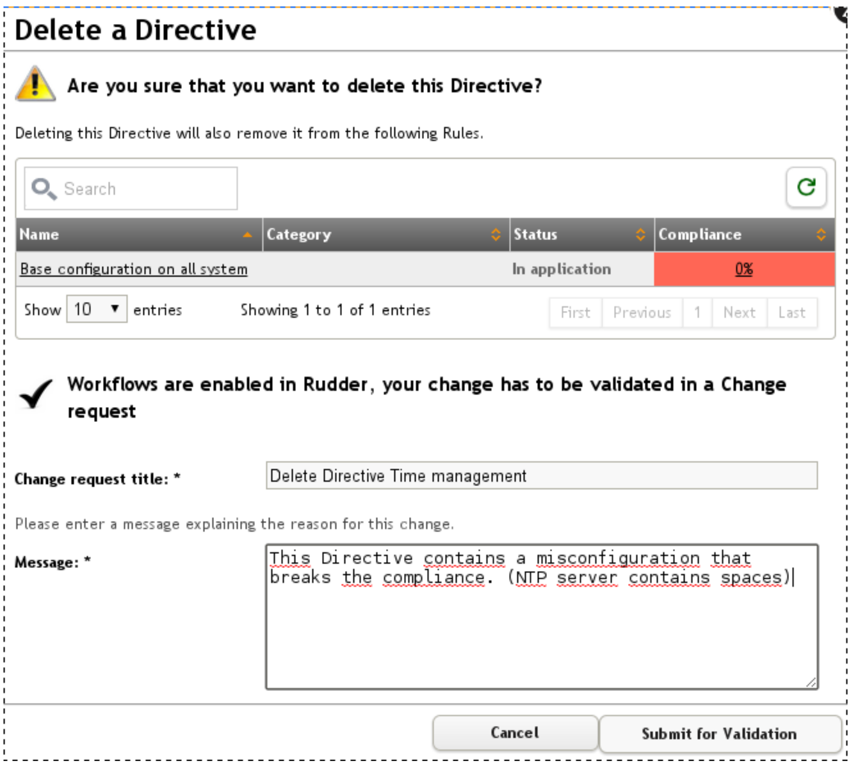
Rule overview:

- Rule: Configure users on all nodes (Rudder ID: D1A35D75-5F28-483B-8D3A-73D031CF5228)
- Name: Configure users on all nodes
- Category:
- Short description:
- Target: Include
  all Nodes from:
  - All nodes (System)
  + Group CentOS machines (Inc. PolServ) (Rudder ID: B5B60DB3-1828-494D-8C13-CD117F51614B)
  Exclude
  all Nodes from:
  + All nodes (System)
- Directives:
  Directive Touch /tmp/beacon (Rudder ID: 181D8271-5B38-4285-9880-7F6396E7894F)
- Enabled: true
- System:
- Long description:

7.6.2 How to create a Change request ?

If they are enabled in *Rudder*, every change in *Rudder* will make you create a Change request. You will have a popup to enter the name of your change request and a change message.

The change message will be used as description for you Change Request, so we advise to fill it anyway to keep an explanation ab out your change.



Delete a Directive

Are you sure that you want to delete this Directive?

Deleting this Directive will also remove it from the following Rules.

Name	Category	Status	Compliance
Base configuration on all system		In application	0%

Show 10 entries Showing 1 to 1 of 1 entries

Workflows are enabled in Rudder, your change has to be validated in a Change request

Change request title: * Delete Directive Time management

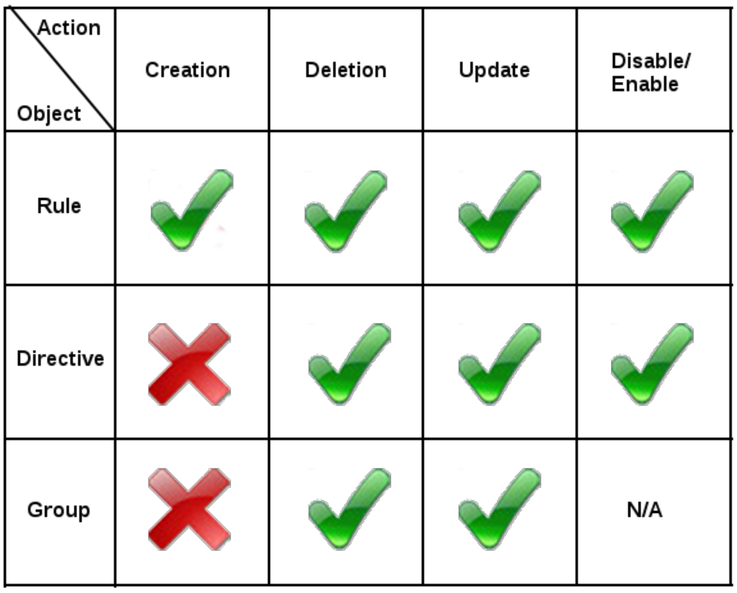
Please enter a message explaining the reason for this change.

Message: *

```
This Directive contains a misconfiguration that
breaks the compliance. (NTP server contains spaces)
```

Change request are not available for *Rule*/*Directive*/*Groups* creation, they are only active if the *Rule*/*Directive*/*Groups* existed before:

Here is a small table about all possibilities:



Action / Object	Creation	Deletion	Update	Disable/ Enable
Rule	✓	✓	✓	✓
Directive	✗	✓	✓	✓
Group	✗	✓	✓	N/A

7.6.3 How to validate a Change request ?

7.6.3.1 Roles

Not every user can validate or deploy change in *Rudder*. Only those with one of the following roles can act on Change request:

Validator	Can validate Change request

Deployer To deploy Change Request

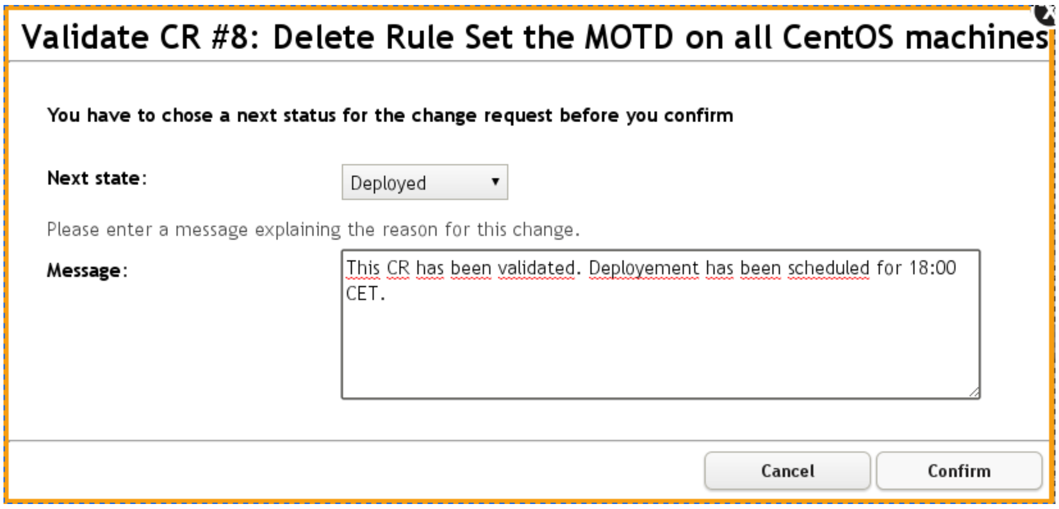
Both of those roles:

- Give you access to pending Change requests
- Allow you to perform actions on them (validate or cancel)

You have to change users in **/opt/rudder/etc/rudder-users.xml** and include those rights. Without one of those roles, you can only access Change Request in *Deployed* or *Cancelled* and those you opened before.

You can deploy directly if you have both the validator and deployer roles. The **administrator** Role gives you both the deployer and valdiator role.

There is also the possibility to access Change requests in Read only mode by using the role *validator_read* or *deployer_read*.


Validate CR #8: Delete Rule Set the MOTD on all CentOS machines

You have to chose a next status for the change request before you confirm

Next state: Deployed

Please enter a message explaining the reason for this change.

Message: This CR has been validated. Deployement has been scheduled for 18:00 CET.

7.6.3.2 Self Validations

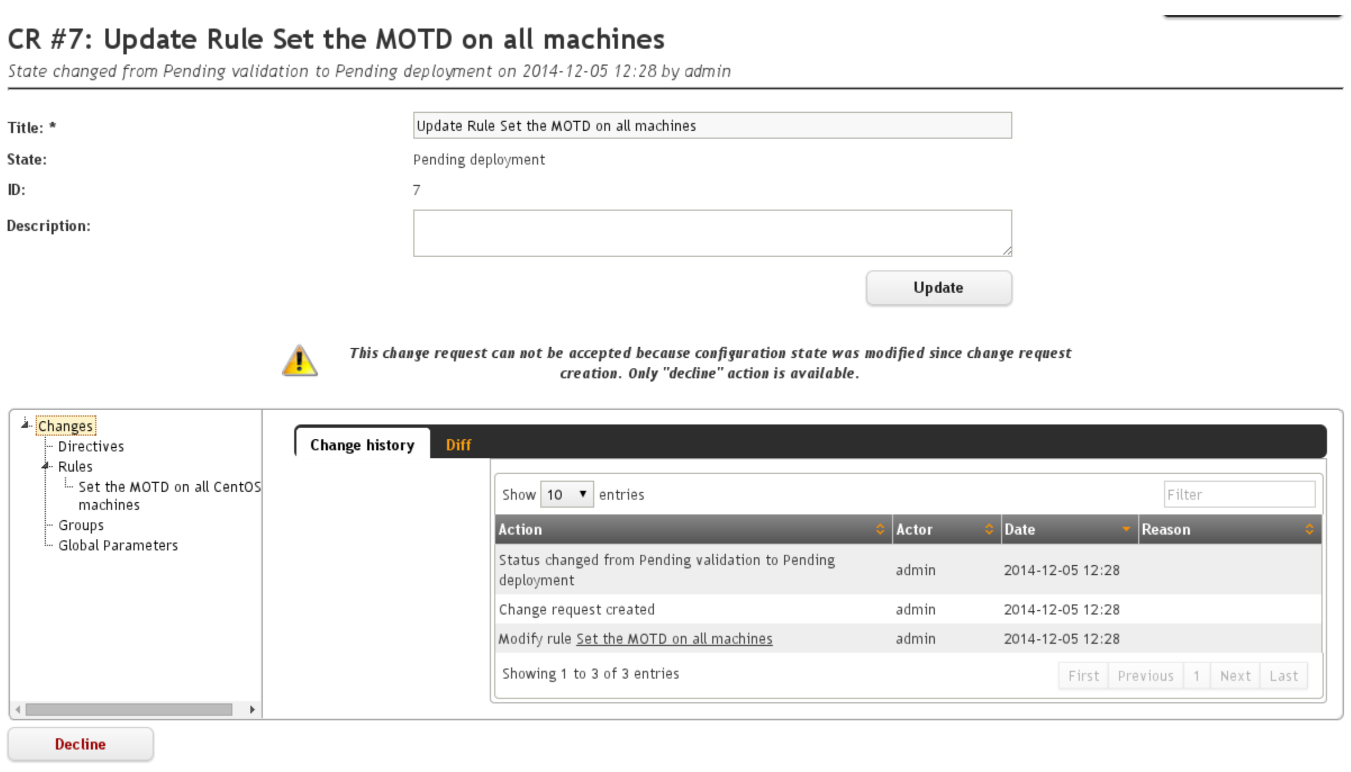
Using Change requests means that you want your team to share knowledge, and validate each other change. So by default:

- **Self validation** is disabled.
- **Self deployment** is enabled.

Those two behaviours can be changed in the property file **/opt/rudder/etc/rudder-web.properties**. *rudder.workflow.self.validation* and *rudder.workflow.self.deployment* are the properties that define this behaviour.

7.6.4 Change request and conflicts

When the initial state of a Change request has changed (i.e.: you want to modify a *Directive*, but someone else changes about that *Directive* has been accepted before yours), your change can't be validated anymore.


CR #7: Update Rule Set the MOTD on all machines

State changed from Pending validation to Pending deployment on 2014-12-05 12:28 by admin

Title: *	Update Rule Set the MOTD on all machines
State:	Pending deployment
ID:	7
Description:	

This change request can not be accepted because configuration state was modified since change request creation. Only "decline" action is available.

Action	Actor	Date	Reason
Status changed from Pending validation to Pending deployment	admin	2014-12-05 12:28	
Change request created	admin	2014-12-05 12:28	
Modify rule Set the MOTD on all machines	admin	2014-12-05 12:28	

Showing 1 to 3 of 3 entries

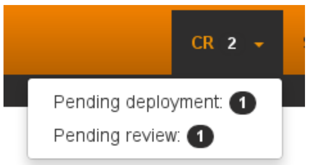
For now, we decided to reduce to the possibility of an error or inconsistency when there are concurrent changes. In a future version of *Rudder*, there will be a system to handle those conflicts, and make sure actual changes are not overwritten.

7.6.5 Notifications:

In several parts of *Rudder* webapp there are some Notifications about Change requests.

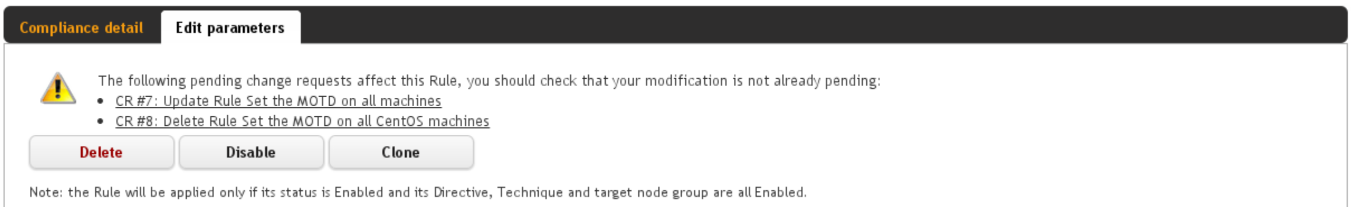
7.6.5.1 Pending change requests

This notification is displayed only if the validator/deployer role is active on your user account. It shows you how many Change requests are waiting to be reviewed/deployed. Clicking on it will lead you to the Change request management page, with a filter already applied.



7.6.5.2 Change already proposed on Rule/Directive/Group

When there is a change about the *Rule/Directive/Group* already proposed but not deployed/cancelled, you will be notified that there are some pending Change requests about that element. You will be provided a Link to those change request, So you can check if the change is already proposed.


The following pending change requests affect this Rule, you should check that your modification is not already pending:

- CR #7: Update Rule Set the MOTD on all machines
- CR #8: Delete Rule Set the MOTD on all CentOS machines

Note: the Rule will be applied only if its status is Enabled and its Directive, Technique and target node group are all Enabled.

Chapter 8

Manage your IT

8.1 How to

8.1.1 Enforce a line is present in a file only once

Enforcing that a line to be present in a single occurence in a file is not an easy process to automate. Providing templates is an easy way to achieve this but not always possible.

If you don't want to use a template, you can use *Technique* **Enforce a File content** to control the content of a file.

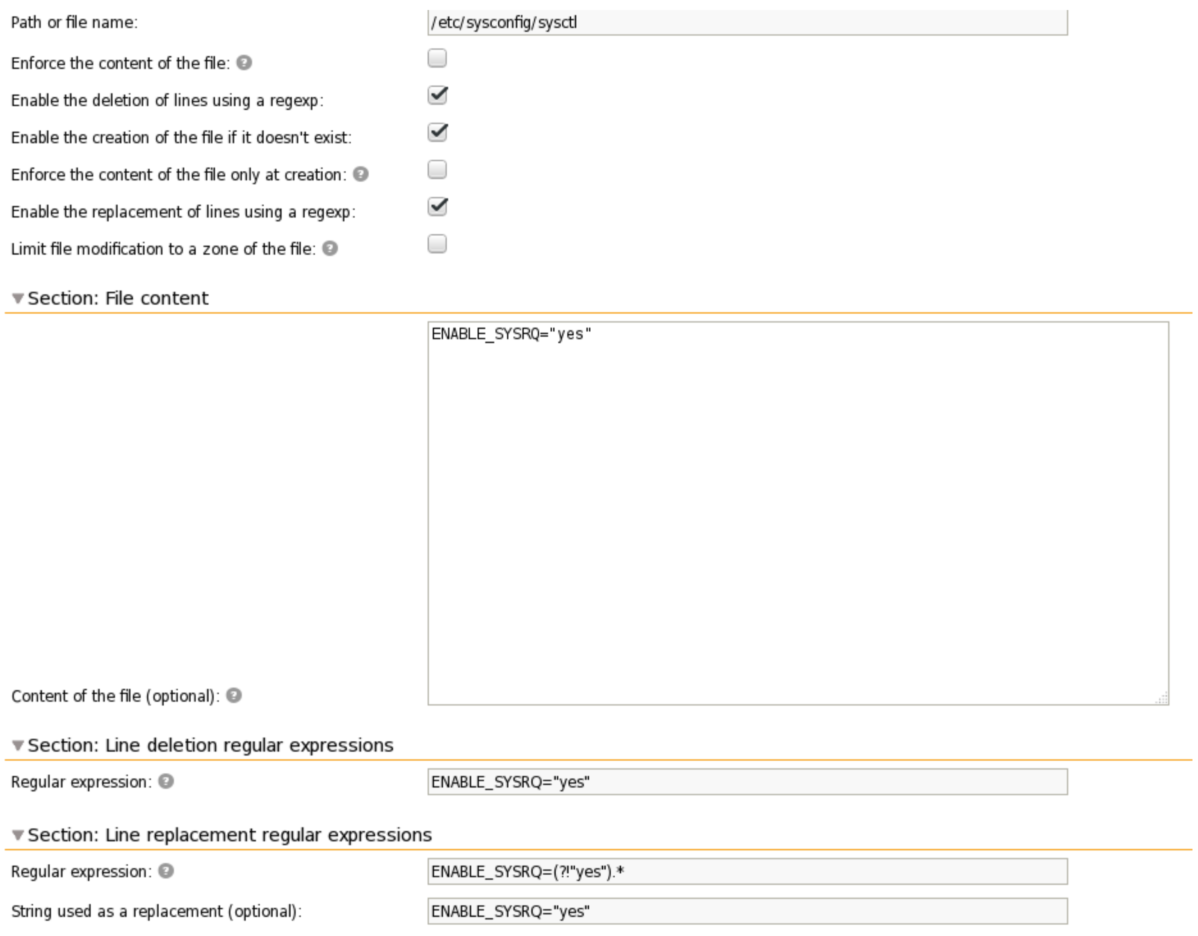
The whole logic to edit a file so it contain only one occurence of a line is:

- Add the line, so it will be added if missing)
- Replace line that looks almost like our line by the line
- Delete all duplicated lines

With these 3 steps, You will end with one line!

So, here is a small example: let's say you want /etc/sysconfig/sysctl to contain line *ENABLE_SYSRQ="yes"*

You will need to create a *Directive* based on Enforce a File content with the following content:



Path or file name:	/etc/sysconfig/sysctl
Enforce the content of the file:	☐
Enable the deletion of lines using a regexp:	✓
Enable the creation of the file if it doesn't exist:	✓
Enforce the content of the file only at creation:	☐
Enable the replacement of lines using a regexp:	✓
Limit file modification to a zone of the file:	☐

▼Section: File content

Content of the file (optional):

```
ENABLE_SYSRQ="yes"
```

▼Section: Line deletion regular expressions

Regular expression:	ENABLE_SYSRQ="yes"

▼Section: Line replacement regular expressions

Regular expression:	ENABLE_SYSRQ=(?!"yes").*
String used as a replacement (optional):	ENABLE_SYSRQ="yes"

Chapter 9

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.

9.1 Archives

9.1.1 Archive usecases

The archive feature of *Rudder* allows to:

- Exchange configuration between multiple *Rudder* instances, in particular when having distinct environments;
- Keep an history of major changes.

9.1.1.1 Changes testing

Export the current configuration of *Rudder* before you begin to make any change you have to test: if anything goes wrong, you can return to this archived state.

9.1.1.2 Changes qualification

Assuming you have multiple *Rudder* instances, each on dedicated for the development, qualification and production environment. You can prepare the changes on the development instance, export an archive, deploy this archive on the qualification environment, then on the production environment.

Versions of the Rudder servers
If you want to export and import configurations between environments, the version of the source and target *Rudder* server must be exactly the same. If the versions don't match (even if only the minor versions are different), there is a risk that the import will break the configuration on the target *Rudder* server.

9.1.2 Concepts

In the *Administration > Archives* section of the *Rudder Server* web interface, you can export and import the configuration of *Rudder Groups*, *Directives* and *Rules*. You can either archive the complete configuration, or only the subset dedicated to *Groups*, *Directives* or *Rules*.

When archiving configuration, a *git tag* is created into `/var/rudder/configuration-repository`. This tag is then referenced in the *Rudder* web interface, and available for download as a zip file. Please note that each change in the *Rudder* web interface is also committed in the repository.

The content of this repository can be imported into any *Rudder* server (with the same version).

9.1.3 Archiving

To archive *Rudder Rules*, *Groups*, *Directives*, or make a global archive, you need to go to the *Administration* > *Archives* section of the *Rudder Server* web interface.

To perform a global archive, the steps are:

1. Click on *Archive everything* - it will update the drop down list *Choose an archive* with the latest data
2. In the drop down list *Choose an archive*, select the newly created archive (archives are sorted by date), for example 2015-01-08 16:39
3. Click on *Download as zip* to download an archive that will contains all elements.

9.1.4 Importing configuration

On the target server, importing the configuration will "merge" them with the existing configuration: every groups, rules, directives or techniques with the same identifier will be replaced by the import, and all others will remain untouched.

To import the archive on the target *Rudder* server, you can follow the following steps:

1. Uncompress the zip archive in /var/rudder/configuration-repository
2. If necessary, correct all files permissions: `chown -R root:rudder directives groups parameters ruleCategories rules techniques`
3. Add all files in the git repository: `git add .&& git commit -am "Importing configuration"`
4. Finally, in the Web interface, go to the *Administration* > *Archives* section, and select *Latest Git commit* in the drop down list in the Global archive section, and click on *Restore everything* to restore the configuration.

Tip
You can also perform the synchronisation from on environment to another by using git, through a unique git repository referenced on both environment.
For instance, using one unique git repository you can follow this workflow:

1. On *Rudder* test:
    a. Use *Rudder* web interface to prepare your policy;
    b. Create an archive;
    c. `git push` to the central repository;
2. On *Rudder* production:
    a. `git pull` from the central repository;
    b. Use *Rudder* web interface to import the qualified archive.

9.1.5 Deploy a preconfigured instance

You can use the procedures of Archiving and Restoring configuration to deploy preconfigured instance. You would prepare first in your labs the configuration for *Groups*, *Directives* and *Rules*, create an Archive, and import the Archive on the new *Rudder* server installation

9.2 Event Logs

Every action happening in the *Rudder* web interface are logged in the PostgreSQL database. The last 1000 event log entries are displayed in the **Administration > View Event Logs** section of *Rudder* web application. Each log item is described by its *ID*, *Date*, *Actor*, and *Event Type*, *Category* and *Description*. For the most complex events, like changes in nodes, groups, techniques, directives, deployments, more details can be displayed by clicking on the event log line.

Event Categories

- User Authentication
- Application
- *Configuration* Rules
- Policy
- *Technique*
- Policy Deployment
- *Node* Group
- *Nodes*
- *Rudder* Agent*s*
- Policy *Node*
- Archives

9.3 Policy Server

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

9.3.1 Configure allowed networks

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

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

9.3.2 Clear caches

Clear cached data, like node configuration. That will trigger a full redeployment, with regeneration of all promises files.

9.3.3 Reload dynamic groups

Reload dynamic groups, so that new nodes and their inventories are taken into account. Normally, dynamic group are automatically reloaded unless that feature is explicitly disable in *Rudder* configuration file.

9.4 Plugins

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

A plugin is a JAR archive. The web application must be restarted after installation of a plugin.

9.4.1 Install a plugin

To install a plugin, just copy the JAR file and the configuration file in the according directories.

`/opt/rudder/jetty7/plugins/` This directory contains the JAR files of the plugins.

`/opt/rudder/etc/plugins/` This directory contains the configuration files of the plugins.

9.5 Basic administration of Rudder services

9.5.1 Restart the agent of the node

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

```
service 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*.

9.5.2 Restart the root rudder service

9.5.2.1 Restart everything

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

```
service rudder-server-root restart
```

9.5.2.2 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.

```
service rudder-agent restart
```

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

```
service rudder-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*.

```
service apache2 restart
```

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

```
service rudder-slapd restart
```

SQL server Store the received reports from the nodes.

```
service postgresql* restart
```

9.6 Password upgrade

This version of *Rudder* uses a central file to manage the passwords that will be used by the application: /opt/rudder/etc/rudder-passwords.conf

When first installing *Rudder*, this file is initialized with default values, and when you run rudder-init, it will be updated with randomly generated passwords.

On the majority of cases, this is fine, however you might want to adjust the passwords manually. This is possible, just be cautious when editing the file, as if you corrupt it *Rudder* will not be able to operate correctly anymore and will spit numerous errors in the program logs.

As of now, this file follows a simple syntax: ELEMENT:password

You are able to configure three passwords in it: The OpenLDAP one, the PostgreSQL one and the authenticated WebDAV one.

If you edit this file, *Rudder* will take care of applying the new passwords everywhere it is needed, however it will restart the application automatically when finished, so take care of notifying users of potential downtime before editing passwords.

Here is a sample command to regenerate the WebDAV password with a random password, that is portable on all supported systems. Just change the "RUDDER_WEBDAV_PASSWORD" to any password file statement corresponding to the password you want to change.

```
sed -i s/RUDDER_WEBDAV_PASSWORD.*/RUDDER_WEBDAV_PASSWORD:$(dd if=/dev/urandom count=128 bs ↩
    =1 2>&1 | md5sum | cut -b-12)/ /opt/rudder/etc/rudder-passwords.conf
```

9.7 User management

Change the users authorized to connect to the application. You can define authorization level for each user

9.7.1 Configuration of the users using a XML file

9.7.1.1 Generality

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

```
<authentication hash="sha512">
  <user name="alice"  password="xxxxxxx" role="administrator"/>
  <user name="bob"    password="xxxxxxx" role="administration_only, node_read"/>
  <user name="custom" password="xxxxxxx" role="node_read,node_write,configuration_read, ↩
      rule_read,rule_edit,directive_read,technique_read"/>
</authentication>
```

The name and password attributes are mandatory (non empty) for the user tags. The role attribute can be omitted but the user will have no permission, and only valid attributes are recognized.

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

```
service rudder-jetty restart
```

9.7.1.2 Passwords

The authentication tag should have a "hash" attribute, making "password" attributes on every user expect hashed passwords. Not specifying a hash attribute will fallback to plain text passwords, but it is strongly advised not to do so for security reasons.

The algorithm to be 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"	SHA1	`read mypass;echo -n $mypass | shasum`
"sha256" or "sha-256"	SHA256	`read mypass;echo -n $mypass | sha256sum`
"sha512" or "sha-512"	SHA512	`read mypass;echo -n $mypass | sha512sum`

Table 9.1: 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="sha256">

  <!-- In this example, the hashed password is: "secret", hashed as a sha256 value -->
  <user name="carol" password="2 ↩
      bb80d537b1da3e38bd30361aa855686bde0eacd7162fef6a25fe97bf527a25b" role="administrator ↩
      "/>

</authentication>
```

9.7.2 Configuring an LDAP authentication provider for Rudder

If you are operating on a corporate network or want to have your users in a centralized database, you can enable *LDAP* authentication for *Rudder* users.

9.7.2.1 LDAP is only for authentication

Take care of the following limitation of the current process: only **authentication** is delegated to *LDAP*, NOT **authorizations**. So you still have to declare user's authorizations in the *Rudder* user file (rudder-users.xml).

A user whose authentication is accepted by *LDAP* but not declared in the rudder-users.xml file is considered to have no rights at all (and so will only see a reduced version of *Rudder* homepage, with no action nor tabs available).

The credentials of a user are defined in the XML file `/opt/rudder/etc/rudder-users.xml`. It expects the same format as regular file-based user login, but in this case "name" will be the login used to connect to *LDAP* and the *password* field will be ignored and should be set to "*LDAP*" to make it clear that this *Rudder* installation uses *LDAP* to log users in.

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

```
service rudder-jetty restart
```

9.7.2.2 Enable LDAP authentication

LDAP authentication is enabled by setting the property `rudder.auth.ldap.enable` to `true` in file `/opt/rudder/etc/rudder-web.properties`

The *LDAP* authentication process is a bind/search/rebind in which an application connection (bind) is used to search (search) for a user entry given some base and filter parameters, and then, a bind (rebind) is tried on that entry with the credential provided by the user.

So next, you have to set-up the connection parameters to the *LDAP* directory to use. There are five properties to change:

- rudder.auth.ldap.connection.url
- rudder.auth.ldap.connection.bind.dn
- rudder.auth.ldap.connection.bind.password
- rudder.auth.ldap.searchbase
- rudder.auth.ldap.filter

The search base and filter are used to find the user. The search base may be left empty, and

Here are some usage examples,

on standard *LDAP*:

```
rudder.auth.ldap.searchbase=ou=People
rudder.auth.ldap.filter=(&(uid={0})(objectclass=person))
```

on *Active Directory*:

```
rudder.auth.ldap.searchbase=
rudder.auth.ldap.filter=(&(sAMAccountName={0})(objectclass=user))
```

9.7.3 Authorization management

For every user you can define an access level, allowing it to access different pages or to perform different actions depending on its level.

You can also build custom roles with whatever permission you want, using a type and a level as specified below.

In the xml file, the role attribute is a list of permissions/roles, separated by a comma. Each one adds permissions to the user. If one is wrong, or not correctly spelled, the user is set to the lowest rights (NoRights), having access only to the dashboard and nothing else.

9.7.3.1 Pre-defined roles

Name	Access level
administrator	All authorizations granted, can access and modify everything
administration_only	Only access to administration part of rudder, can do everything within it.
user	Can access and modify everything but the administration part
configuration	Can only access and act on configuration section
read_only	Can access to every read only part, can perform no action
inventory	Access to information about nodes, can see their inventory, but can't act on them
rule_only	Access to information about rules, but can't modify them

For each user you can define more than one role, each role adding its authorization to the user.

Example: "rule_only,administration_only" will only give access to the "Administration" tab as well as the *Rules*.

9.7.3.2 Custom roles

You can set a custom set of permissions instead of a pre-defined role.

A permission is composed of a type and a level:

- Type: Indicates what kind of data will be displayed and/or can be set/updated by the user
  - "configuration", "rule", "directive", "technique", "node", "group", "administration", "deployment".
- Level: Access level to be granted on the related type
  - "read", "write", "edit", "all" (Can read, write, and edit)

Depending on that value(s) you give, the user will have access to different pages and action in *Rudder*.

Usage example:

- configuration_read → Will give read access to the configuration (*Rule* management, *Directives* and Parameters)
- rule_write, node_read → Will give read and write access to the *Rules* and read access to the *Nodes*

9.7.4 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 or an enterprise 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.

9.8 Monitoring

This section will give recommendations for:

- Monitoring *Rudder* itself (besides standard monitoring)
- Monitoring the state of your configuration management

9.8.1 Monitoring Rudder itself

9.8.1.1 Monitoring a Node

The monitoring of a node mainly consists in checking that the *Node* can speak with its policy server, and that the agent is run regularly.

A good place to start is to check the content of the last run log file. It can be found in */var/rudder/cfengine-community/output/previous*.

You can search lines containing *FATAL:*, *Fatal :*, or *could not get an updated configuration* with a log monitoring tool to automatically detect communication issues with the policy server.

To get the last run time, you can lookup the modification date of */var/rudder/cfengine-community/last_successful_inputs_update*.

9.8.1.2 Monitoring a Server

You can use use regular API calls to check the server is running and has access to its data. For example, you can issue the following command to get the list of currently defined rules:

```
curl -X GET -H "X-API-Token: yourToken" http://your.rudder.server/rudder/api/latest/rules
```

You can then check the status code (which should be 200). See the API documentation for more information.

You can also check the webapp logs (in */var/log/rudder/webapp/year_month_day.stderrout.log*) for error messages.

9.8.2 Monitoring your configuration management

There are two interesting types of information:

- **Events**: all the changes made by the the agents on your *Nodes*
- **Compliance**: the current state of your *Nodes* compared with the expected configuration

The Web interface gives access to this, but we will here see how to process events automatically. They are available on the root server, in */var/log/rudder/compliance/non-compliant-reports.log*. This file contains two types of reports about all the nodes managed by this server:

- All the modifications made by the agent
- All the errors that prevented the application of a policy

The lines have the following format:

```
[%DATE%] N: %NODE_UUID% [%NODE_NAME%] S: [%RESULT%] R: %RULE_UUID% [%RULE_NAME%] D: % ↩
    DIRECTIVE_UUID% [%DIRECTIVE_NAME%] T: %TECHNIQUE_NAME%/%TECHNIQUE_VERSION% C: [% ↩
    COMPONENT_NAME%] V: [%KEY%] %MESSAGE%
```

In particular, the *RESULT* field contains the type of event (change or error, respectively *result_repaired* and *result_error*).

Below is a basic Logstash configuration file for parsing *Rudder* events. You can then use Kibana to explore the data, and create graphs and dashboards to visualize the changes in your infrastructure.

```
input {
   file {
      path => "/var/log/rudder/compliance/non-compliant-reports.log"
   }
}

filter {
   grok {
      match => { "message" => "^\[%{DATA:date}\] N: %{DATA:node_uuid} \[%{DATA:node}\] S:  ↩
          \[%{DATA:result}\] R: %{DATA:rule_uuid} \[%{DATA:rule}\] D: %{DATA:directive_uuid} ↩
           \[%{DATA:directive}\] T: %{DATA:technique}/%{DATA:technique_version} C: \[%{DATA: ↩
          component}\] V: \[%{DATA:key}\] %{DATA:message}$" }
   }
   # Replace the space in the date by a "T" to make it parseable by Logstash
   mutate {
      gsub => [ "date", " ", "T" ]
   }
   # Parse the event date
   date {
      match => [ "date" , "ISO8601" ]
   }
   # Remove the date field
   mutate { remove => "date" }
   # Remove the key field if it has the "None" value
   if [key] == "None" {
      mutate { remove => "key" }
   }
}

output {
    stdout { codec => rubydebug }
}
```

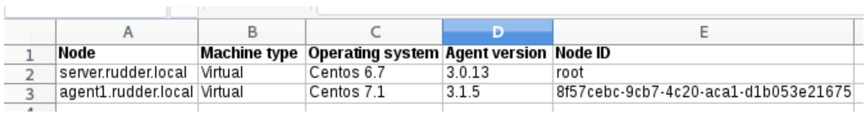
9.9 Use Rudder inventory in other tools

Rudder centralizes the information about your managed systems, and you can use this information in other tools, mainly through the API. We well here give a few examples.

9.9.1 Export to a spreadsheet

You can export the list of your nodes to a spreadsheet file (xls format) by using a tool available in the rudder-tools repository.

Simple follow the installation instructions, and run it against your *Rudder* server. You will get a file containing:



	A	B	C	D	E
1	**Node**	**Machine type**	**Operating system**	**Agent version**	**Node ID**
2	server.rudder.local	Virtual	Centos 6.7	3.0.13	root
3	agent1.rudder.local	Virtual	Centos 7.1	3.1.5	8f57cebc-9cb7-4c20-aca1-d1b053e21675

You can easily modify the script to add other information.

9.9.2 Use the inventory in Rundeck

Rundeck is a tool that helps automating infrastructures, by defining jobs that can be run manually or automatically. There is a plugin for Rundeck that allows using *Rudder* inventory data in Rundeck.

9.9.3 Use the inventory in Ansible

There is an inventory plugin for Ansible that makes possible to use *Rudder* inventory (including groups, nodes, group ids, node ids, and node properties) as inventory for Ansible, for example for orchestration tasks on your platform.

Chapter 10

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. . .

10.1 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.

10.2 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 *Techniques*, you could create a Name resolution *Directive* to use your own internal DNS servers for normal situations, and a second, alternative *Directive*, to use Google's public DNS servers, in case your internal DNS servers are no longer available.

10.3 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.

Chapter 11

Advanced usage

This chapter describe advanced usage of *Rudder*.

11.1 Node management

11.1.1 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.

```
rudder agent reset
```

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

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

11.1.2 Completely reinitialize a Node

You may want to completely reinitialize a *Node* to make it seen as a new node on the server, for example after cloning a VM.

Warning
This command will permanently delete your node uuid and keys, and no configuration will be applied before re-accepting and configuring the node on the server.

The command to reinitialize a *Node* is:

```
rudder agent reinit
```

This command will delete all local agent data, including its uuid and keys, and also reset the agent internal state. The only configuration kept is the server hostname or ip configured in `policy_server.dat`. It will also send an inventory to the server, which will treat it as a new node inventory.

11.1.3 Change the agent run schedule

By default, the agent runs on all nodes every 5 minutes. You can modify this value in **Administration** → **Settings** → ***Agent* Run Schedule**, as well as the "splay time" across nodes (a random delay that alters scheduled run time, intended to spread load across nodes).

> **Warning**
> When reducing notably the run interval length, reporting can be in *No report* state until the next run of the agent, which can take up to the previous (longer) interval.

11.1.4 Installation of the Rudder Agent

11.1.4.1 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*.

11.1.4.2 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 Policies, 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.

11.1.4.3 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 on *Rudder Root* and Relay servers, serving policies and files to *Rudder Nodes*.

11.1.4.4 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
```

11.1.5 Rudder Agent interactive

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

```
rudder agent run
```

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:

```
rudder agent run

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 address 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

```
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:

```
# rudder agent run
 !! 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:

```
service rudder-agent restart
```

11.1.6 Processing new inventories on the server

11.1.6.1 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/
```

11.1.6.2 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 *Inventory* Endpoint do its job and the new *Node* appears in the interface.

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

```
rudder agent run
```

11.1.6.3 Validate new Nodes

User interaction is required to validate new *Nodes*.

11.1.6.4 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. *Technique* has changed;
4. *Directive* has changed;
5. *Group* of *Node* has changed;
6. *Rule* has changed;
7. Regeneration was forced by the user.

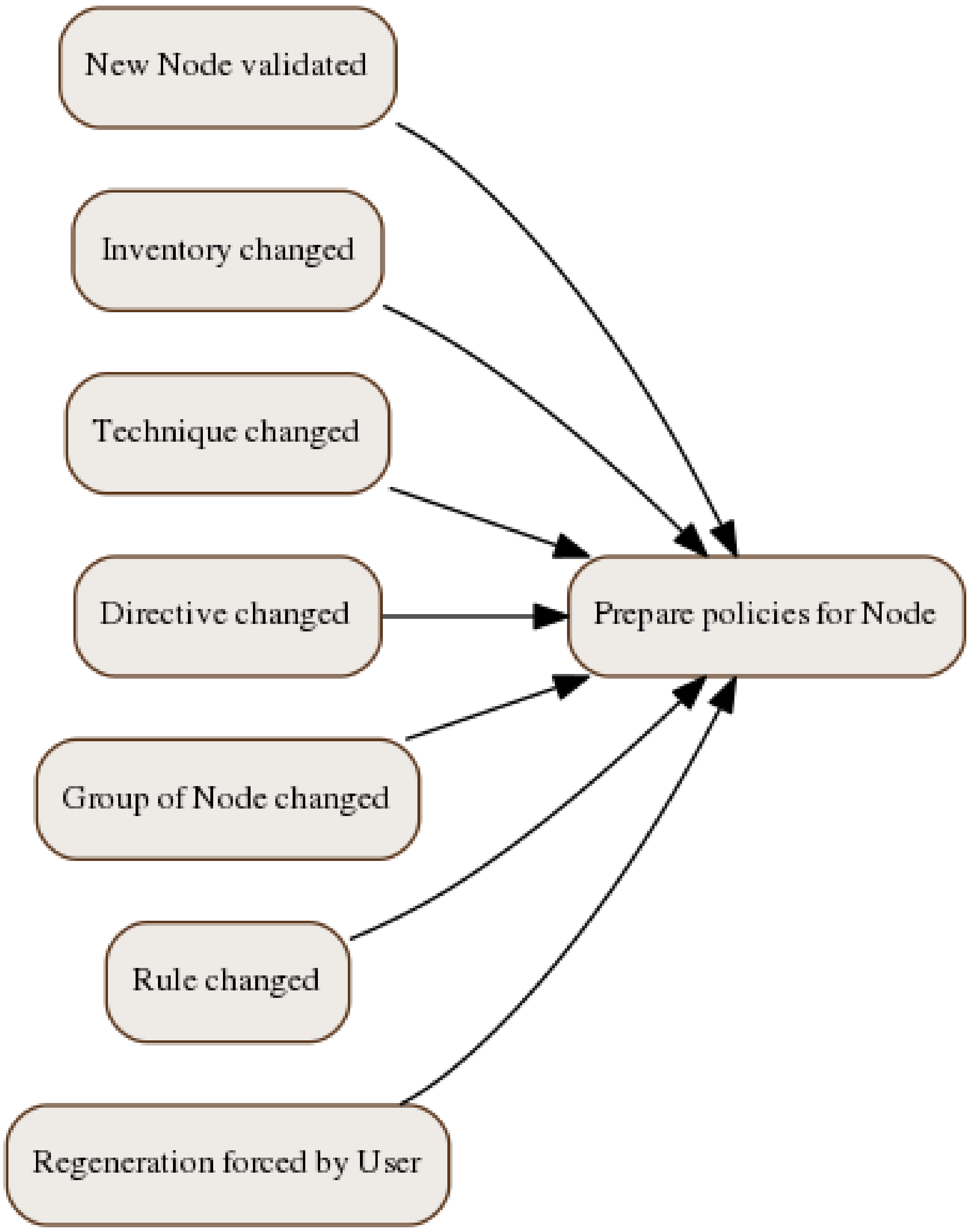


Figure 11.1: Generate policy workflow

11.1.7 Agent execution frequency on nodes

11.1.7.1 Checking configuration (CFEngine)

Rudder is configured to check and repair configurations using the *CFEngine* agent every 5 minutes, at 5 minutes past the hour, 10 minutes past the hour, etc.

The exact run time on each machine will be delayed by a random interval, in order to "smooth" the load across your infrastructure (also known as "splay time"). This reduces simultaneous connections on relay and root servers (both for the *CFEngine* server and for sending reports).

Up to and including *Rudder* 2.10.x, this random interval is between 0 and 1 minutes. As of *Rudder* 2.10.x and later, this random interval is between 0 and 5 minutes.

11.1.7.2 Inventory (FusionInventory)

The *FusionInventory* agent collects data about the node it's running on such as machine type, OS details, hardware, software, networks, running virtual machines, running processes, environment variables…

This inventory is scheduled once every 24 hours, and will happen in between 0:00 and 5:00 AM. The exact time is randomized across nodes to "smooth" the load across your infrastructure.

11.2 Password management

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

11.2.1 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
```

11.2.2 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
```

11.2.3 Configuration of the WebDAV access password

This time, the procedure is a bit more tricky, as you will have to update the *Technique* 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 *Techniques*

```
cd /var/rudder/configuration-repository/techniques/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 Directives* by either reloading them in the web interface (in the "Configuration Management/*Techniques*" tab) or restarting jetty (NOT recommended)

11.3 Policy generation

Each time a change occurs in the *Rudder* interface, having an impact on the *CFEngine* promises needed by a node, it is necessary to regenerate the modified promises for every impacted node. By default this process is launched after each change.

11.3.1 `Regenerate now` button

The button `Regenerate now` on the top right of the screen permit you to force the regeneration of the promises. As changes in the inventory of the nodes are not automatically taken into account by *Rudder*, this feature can be useful after some changes impacting the inventory information.

11.4 Technique creation

Rudder provides a set of pre-defined *Techniques* that cover some basic configuration and system administration needs. You can also create your own *Techniques*, to implement new functionalities or configure new services. This paragraph will walk you through this process.

There is two ways to configure new *Techniques*, either thanks to the web *Technique* Editor in *Rudder* or by coding them by hand.

The use of the *Technique* Editor (code name: ncf-builder [http://www.ncf.io/pages/ncf-builder.html]) is the easiest way to create new *Techniques* and is fully integrated with *Rudder*. On the other hand, it does not allow the same level of complexity and expressiveness than coding a *Technique* by hand. Of course, coding new *Techniques* by hand is a more involved process that needs to learn how the *Technique* description language and *Technique* reporting works.

We advice to always start to try to create new *Techniques* with the *Technique* Editor and switch to the hand-coding creation only if you discover specific needs not addressed that way.

11.4.1 Recommended solution: Technique Editor

The easiest way to create your own *Techniques* is to use the *Technique* editor, a web interface to create and manage *Techniques* based on the ncf framework.

Creating a technique in the *Technique* Editor will generate a *Technique* for *Rudder* automatically. You can then use that *Technique* to create a *Directive* that will be applied on your *Nodes* thanks to a *Rule*.

For more information about ncf and the *Technique* editor, you can visit: http://www.ncf.io/

11.4.1.1 Using the Technique Editor

The *Technique* Editor is available in the *Directive* screen or directly in the Utilities menu. Once on the *Technique* Editor, creating a *Technique* simply consist to add desired "Generic Methods" building block and configure them.

When the *Technique* match your expectations, hitting save will automatically add it to available *Technique* in the *Directive* screen of *Rudder* (in the "User *Technique*" category).

11.4.1.2 Logs

In case of any issue with the *Technique* Editor, the first step should always be to look for its log messages. These logs are sent to *Apache* system error logs:

- On *Debian*, by default: /var/log/apache2/error.log
- On *RHEL*, by default: /var/log/httpd/error_log

11.4.2 Understanding how Technique Editor works

In this chapter, we are giving an overview about how the *Technique* Editor works and how it is integrated with the main *Rudder* application.

11.4.2.1 Directory layout

As explained in http://www.ncf.io/, ncf uses a structured directory tree composed of several layers of logic, from internal libraries to *Techniques* and user services. All the files and logic in these folders will be named "library" for simplicity

ncf directory structure exists in two root folders:

- /usr/share/ncf/tree
  - This is the standard library installation folder. It is created and updated by the the ncf package. This folder will be completely overwritten when you update ncf package so you should never modify anything here: it will be lost at some point.
- /var/rudder/configuration-repository/ncf
  - This is were you add your own ncf Generic Methods and *Techniques*. *Techniques* created with the *Technique* Editor will be located here, and both Generic Methods and *Techniques* in that place will be accessible in the *Technique* Editor alongside what is provided by the standard library.

To share those folders to all nodes, *Rudder* makes a copy of these folders in /var/rudder/ncf:

- /var/rudder/ncf/local is a copy of /var/rudder/configuration-repository/ncf
- /var/rudder/ncf/common is a copy /usr/share/ncf/tree

They are synchronized automatically by the agent running on the server. So any modification done in files in these directories will be lost at the next synchronization.

A node updates its ncf local library by copying the content of these two folders during its promise update phase.

11.4.2.2 Technique Editor integration with Rudder

Here we will explain *Technique* Editor behavior and what workflow are initialized by different action on the *Technique* Editor.

Each action in the *Technique* Editor interface produces requests to an API defined over ncf.

All of the requests are authenticated thanks to a token passed in the JSESSIONID header. The token is generated when an authenticated user is connected to the *Rudder* interface (typically thanks to his browser).

That token is shared to the *Technique* Editor interface, which itself passes the JSESSIONID header to all requests.

If you have authentication issue, check that your *Rudder* session is not expired.

Get request Get request will get all *Techniques* and Generic Methods in a path passed as parameters of the request in the "path" javascript variable:

https://you-rudder/ncf-builder/#!?path=/var/rudder/configuration-repository/ncf

Get requests are triggered when accessing *Technique* editor.

The ncf API will parse all files in the parameter path by running "cf-promises -pjson" on all *Techniques*, checking that all *Techniques* are correctly formed.

The ncf API will also look to all Generic Methods description data to build the catalog of available Generic Methods.

The resulting information are sent back to the *Technique* Editor for displaying.

Post requests Post requests are issued when a *Technique* is created, modified or deleted. They will only work on *Techniques* available in the path given in parameter.

They are triggered when clicking on save/delete button.

The main difference with get requests is that hooks are launched before and after the action is made.

We will see all hooks behavior in the following dedicated hooks section.

11.4.2.3 Hooks

On each POST request, pre- and post- hooks are executed by the *Technique* Editor. These hooks are used for the *Rudder* integration to help transform pure ncf *Techniques* into *Rudder* one.

- pre-hooks are located in: /var/rudder/configuration-repository/ncf/pre-hooks.d
- post-hooks are located in: /var/rudder/configuration-repository/ncf/post-hooks.d

As of March 2015, we have two post-hooks defined and no pre-hooks:

- post.write_technique.commit.sh
  - It commits the *Technique* newly created into *Rudder* Git configuration repository located in /var/rudder/configuration-repository.
- post.write_technique.rudderify.sh
  - It generates a valid *Rudder Technique* from a the newly created *Technique* and reloads *Rudder Technique* Library so that updates are taken into account.

If you want to run post hooks by hand, you can use the following command:

```
/var/rudder/configuration-repository/ncf/post-hooks.d/post.write_technique.commit. ↩
    sh /var/rudder/configuration-repository bundle_name
```

11.4.3 Create Technique manually

11.4.3.1 Prerequisite

To create a *Technique*, you'll need a few things:

***CFEngine* knowledge** *Rudder*'s *Techniques* are implemented using *CFEngine*. *Rudder* takes care of a lot of the work of using *CFEngine*, but you'll need to have a reasonable understanding of the *CFEngine* syntax.

***Rudder* installation for testing** To be able to test your new *Technique*, you'll need a working *Rudder* installation (at least a server and a node).

Text editor The only other tool you need is your favorite text editor!

11.4.3.2 Define your objective

Before starting to create a new *Technique*, have you checked that it doesn't already exist in *Rudder*? The full list of current *Techniques* is available from GitHub, at GitHub rudder-techniques repository.

OK, now we've got that over with, let's go on.

A *Technique* should be an abstract configuration. This means that your *Technique* shouldn't just configure something one way, but instead it should implement **how** to configure something, and offer options for users to choose what way they want it configured. Before starting, make sure you've thought through what you want to create.

Here's a quick checklist to help:

- Do you need to install packages?
- Do you need to create or edit configuration files?
- Do you need to copy files from a central location?
- Do you need to launch processes or check that they're running?
- Do you need to run commands to get things working?

Once you've made a list of what needs doing, consider what options could be presented in the user interface, when you create a *Directive* from your new *Technique*. Intuitively, the more variables there are, the more flexible your *Technique* will be. However, experience shows that making the *Technique* **too** configurable will actually make it harder to use, so a subtle balance comes in to play here.

At this stage, make a list of all the variables that should be presented to users configuring a *Directive* from your *Technique*.

11.4.3.3 Initialize your new Technique

The simplest way to create a new *Technique* and be able to test it as you work is to start on a *Rudder* server. Open a terminal and connect to your *Rudder* server by ssh, and cd into the directory where *Techniques* are stored:

```
$ cd /var/rudder/configuration-repository/techniques
```

Under this directory, you'll find a set of categories, and sub-categories. Before creating your *Technique*, choose a category to put it in, and change to that directory. For example:

```
$ cd applications
```

You can consult the description of each category by looking at the `category.xml` file in each directory. For this example:

```
$ cat category.xml
<xml>
    <name>Application management</name>
    <description>This category contains Techniques designed to install,
        configure and manage applications</description>
</xml>
```

Once you've decided on a category, it's time to create the basic skeleton of your *Technique*. The technical name for your *Technique* is it's directory name, so choose wisely:

```
mkdir sampleTechnique
```

All directories under this one are version numbers. Let's start with a simple 1.0 version. From now on, we'll work in this directory.

```
mkdir sampleTechnique/1.0
cd sampleTechnique/1.0
```

Now, you need a minimum of two files to get your *Technique* working:

metadata.xml This file describes the *Technique*, and configures how it will be displayed in the web interface.

st files These files are templates for *CFEngine* configuration files. You need at least one, but can have as many as you like. *Rudder* processes them to generate .cf files ready to be used by *CFEngine*.

To get started, copy and paste these sample files, or download them from GitHub:

`metadata.xml` (original file: `technique-metadata-sample.xml`)

```
include::technique-metadata-sample.xml
```

`sample_technique.st` (original file: `technique-st-sample.xml`)

```
include::technique-st-sample.xml
```

11.4.3.4 Define variables

WORK IN PROGRESS Define metadata. Enter the variables in sections in the metadata.xml file. Cf http://www.rudder-project.org/foswiki/Development/PolicyTemplateXML

11.4.3.5 First test in the Rudder interface

Load the new *Technique* into *Rudder* and check that the variables and sections are displayed as you expect.

11.4.3.6 Implement the behavior

WORK IN PROGRESS Write *CFEngine* promises to implement the behavior that your Template should have.

11.4.3.7 Read in the variables from Rudder

WORK IN PROGRESS Using StringTemplate notation… Cf http://www.rudder-project.org/foswiki/Development/*Technique*

11.4.3.8 Add reporting

WORK IN PROGRESS The reports format Cf http://www.rudder-project.org/foswiki/Development/ReportsIn*Techniques*

11.5 REST API

Rudder can be used as a web service using a *REST API*.

This documentation covers the version 1 of *Rudder*'s API, that has been present since *Rudder* 2.4.

The version 2 has now been implemented, which is much more complete, in *Rudder* 2.7, and has a dedicated documentation available here: http://www.rudder-project.org/rudder-api-doc/

Warning
The version 1 is to be considered legacy and should not be used anymore. Please migrate to version 2 to benefit from the new authentication features and more complete existing methods.

11.5.1 Default setup

Access to *REST API* can be either using *Rudder* authentication, either unauthenticated, using authentication mechanisms set elsewhere, for instance at *Apache* level.

11.5.1.1 Rudder Authentication

By default, the access to the *REST API* is open to users not authenticated in *Rudder*.

The method of authentication can be configured in `/opt/rudder/etc/rudder-web.properties`

```
rudder.rest.allowNonAuthenticatedUser=true
```

11.5.1.2 Apache access rules

By default, the *REST API* is exposed for localhost only, at `http://localhost/rudder/api`.

Example 11.1 Example usage of non authenticated REST API

Unrestricted access can be granted to local scripts accessing to `localhost`, whereas remote access to the *REST API* will be either denied, or restricted through authentication at apache level.

11.5.1.3 User for REST actions

Actions done using the *REST API* are logged by default as run by the user `UnknownRestUser`.

To change the name of this user, add following header to the HTTP request:

```
X-REST-USERNAME: MyConfiguredRestUser
```

If the *REST API* is authenticated, the authenticated user name will be used in the logs.

11.5.2 Status

`http://localhost/rudder/api/status` Check if *Rudder* server is up and return `OK`. If *Rudder* server is not responding, an error is displayed.

11.5.3 Promises regeneration

`http://localhost/rudder/api/deploy/reload` Regenerate promises (same action as the `Regenerate now` button).

11.5.4 Dynamic groups regeneration

`http://localhost/rudder/api/dyngroup/reload` Check all dynamic groups for changes. If changes have occurred, regenerate the groups in the *LDAP* and the *CFEngine* promises.

11.5.5 Technique library reload

`http://localhost/rudder/api/techniqueLibrary/reload` Check the technique library for changes. If changes have occurred, reload the technique library in memory and regenerate the *CFEngine* promises.

11.5.6 Archives manipulation

Various methods are available to import and export items:

11.5.6.1 Archiving:

`http://localhost/rudder/api/archives/archive/groups` Export node groups and node groups categories.

`http://localhost/rudder/api/archives/archive/directives` Export policy library (categories, active techniques, directives).

`http://localhost/rudder/api/archives/archive/rules` Export rules

`http://localhost/rudder/api/archives/archive/full` Export everything

11.5.6.2 Listing:

`http://localhost/rudder/api/archives/list/groups` List available archives datetime for groups (the datetime is in the format awaited for restoration).

`http://localhost/rudder/api/archives/list/directives` List available archives datetime for policy library (the datetime is in the format awaited for restoration).

`http://localhost/rudder/api/archives/list/rules` List available archives datetime for configuration rules (the datetime is in the format awaited for restoration).

`http://localhost/rudder/api/archives/list/full` List available archives datetime for full archives (the datetime is in the format awaited for restoration).

11.5.6.3 Restoring a given archive:

`http://localhost/rudder/api/archives/restore/groups/datetime/[archiveId]` Restore given groups archive.

`http://localhost/rudder/api/archives/restore/directives/datetime/[archiveId]` Restore given directives archive.

`http://localhost/rudder/api/archives/restore/rules/datetime/[archiveId]` Restore given rules archive.

`http://localhost/rudder/api/archives/restore/full/datetime/[archiveId]` Restore everything.

11.5.6.4 Restoring the latest available archive (from a previously archived action, and so from a Git tag):

```
http://localhost/rudder/api/archives/restore/groups/latestArchive
http://localhost/rudder/api/archives/restore/directives/latestArchive
http://localhost/rudder/api/archives/restore/rules/latestArchive
http://localhost/rudder/api/archives/restore/full/latestArchive
```

11.5.6.5 Restoring the latest available commit (use Git HEAD):

```
http://localhost/rudder/api/archives/restore/groups/latestCommit
http://localhost/rudder/api/archives/restore/directives/latestCommit
http://localhost/rudder/api/archives/restore/rules/latestCommit
http://localhost/rudder/api/archives/restore/full/latestCommit
```

11.5.6.6 Downloading a ZIP archive

The *REST API* allows to download a ZIP archive of groups, directives and rules (as XML files) for a given Git commit ID (the commit HASH).

It is not designed to query for available Git commit ID, so you will need to get it directly from a Git tool (for example with Git log) or from the list API.

Note that that API allows to download ANY Git commit ID as a ZIP archive, not only the one corresponding to *Rudder* archives.

Note 2: you should rename the resulting file with a ".zip" extension as most zip utilities won't work correctly on a file not having it.

`http://localhost/rudder/api/archives/zip/groups/[GitCommitId]` Download groups for the given Commit ID as a ZIP archive.

`http://localhost/rudder/api/archives/zip/directives/[GitCommitId]` Download directives for the given Commit ID as a ZIP archive.

`http://localhost/rudder/api/archives/zip/rules/[archiveId]` Download rules for the given Commit ID as a ZIP archive.

`http://localhost/rudder/api/archives/zip/full/[archiveId]` Download groups, directives and rules for the given Commit ID as a ZIP archive.

11.6 Multiserver Rudder

From version 3.0 *Rudder* can be divided into 4 different components:

- rudder-web: an instance with the webapp and the central policy server
- rudder-ldap: the inventory endpoint and its ldap backend
- rudder-db: the postgresql storage
- rudder-relay-top: the contact point for nodes

11.6.1 Preliminary steps

You need the setup scripts provided at https://github.com/normation/rudder-tools/tree/master/scripts/rudder-multiserver-setup. You can download them with this command:

```
mkdir rudder-multiserver-setup
cd rudder-multiserver-setup
for i in add_repo detect_os.sh rudder-db.sh rudder-ldap.sh rudder-relay-top.sh rudder-web. ↩
    sh
do
  wget --no-check-certificate https://raw.githubusercontent.com/Normation/rudder-tools/ ↩
      master/scripts/rudder-multiserver-setup/$i
done
chmod 755 *
cd ..
```

You need 4 instances of supported OS, one for each component. Only the rudder-web instance need at least 2GB of RAM.

Register the 4 names in the DNS or add them in `/etc/hosts` on each instance.

Add firewall rules:

- from rudder-web to rudder-db port pgsql TCP
- from rudder-* to rudder-web port rsyslog 514 TCP
- from rudder-relay-top to rudder-ldap port 8080 TCP
- from rudder-web to rudder-ldap port 8080 TCP
- from rudder-web to rudder-ldap port 389 TCP
- from rudder-web to rudder-relay-top port 5309

11.6.2 Install rudder-relay-top

Copy the rudder-multiserver-setup directory to you instance.

Run rudder-relay-top.sh as root, replace <rudder-web> with the hostname of the rudder-web instance:

```
cd rudder-multiserver-setup
./rudder-relay-top.sh <rudder-web>
```

Take note of the UUID. If you need it later read, it is in the file /opt/rudder/etc/uuid.hive

11.6.3 Install rudder-db

Copy the rudder-multiserver-setup directory to you instance.

Run rudder-db.sh as root, replace <rudder-web> with the hostname of the rudder-web instance, replace <allowed-network> with the network containing the rudder-web instances:

```
cd rudder-multiserver-setup
./rudder-db.sh <rudder-web> <allowed-network>
```

11.6.4 Install rudder-ldap

Copy the rudder-multiserver-setup directory to you instance.

Run rudder-ldap.sh as root, replace <rudder-web> with the hostname of the rudder-web instance:

```
cd rudder-multiserver-setup
./rudder-ldap.sh <rudder-web>
```

11.6.5 Install rudder-web

Copy the rudder-multiserver-setup directory to you instance.

Run rudder-relay-top.sh as root, replace <rudder-*> with the hostname of the corresponding instance:

```
cd rudder-multiserver-setup
./rudder-web.sh <rudder-web> <rudder-ldap> <rudder-db> <rudder-relay-top>
```

Connect rudder web interface and accept all nodes. Then run the following command where <relay-uuid> is the uuid from rudder-relay-top setup.

```
/opt/rudder/bin/rudder-node-to-relay <relay-uuid>
```

11.7 Server migration

11.7.1 What files you need

To copy a server on a new location, you need at least to keep the configuration applied by your server.

You need to keep:

- ***Rules***
- ***Directives***
- ***Groups***
- ***Techniques***

If you keep your actual nodes, you also have to handle with *CFEngine* keys. New nodes won't have problems with the new server.

If your new server has a different IP, you will have to change it on your nodes.

You will have to accept nodes

There are multiple ways to migrate your server, here are the best we propose you.

11.7.2 Handle configuration files

11.7.2.1 Copy /var/rudder/configuration-repository

The simplest way to migrate your server to a new one is to copy /var/rudder/configuration-repository from your former server to the new one. In this folder you will find all your *Rules*/*Groups*/*Directives*/*Techniques* are stored. By copying that folder you will keep the git tree used by your server and keep your comments.

- Copy `/var/rudder/configuration-repository` to your new server
- In *Rudder* UI Go to **Administration > Policy Server**
- Reload the *Technique* Library
- Go to **Administration > Archives**
- In Global Archive, "Choose an archive" select *Latest git commit*
- Click on *Restore everything*
- After deployment, your configuration should be restored

11.7.2.2 Use Archive feature of Rudder

Alternatively, you can follow the Archive/Import procedures described in Archives

11.7.3 Handle CFEngine keys

11.7.3.1 Keep your CFEngine keys

Copy `/var/rudder/cfengine-community/ppkeys` to your new server

11.7.3.2 Change CFEngine keys

On every node that were using your old rudder server, you will have to erase the server public key (root-MD5=*.pub file)

Run `rm /var/rudder/cfengine-community/ppkeys/root-MD5=*.pub`

On the next run of rudder-agent, nodes will get the new public key of the server

11.7.4 On your nodes

If your server has changed of IP address you have to modify `/var/rudder/cfengine-community/policy_server.dat` with the new address

Then you force your nodes to send their inventory while running `rudder agent inventory`

In your *Rudder* UI, you should now be able to accept the nodes.

Your configuration is now totally migrated.

11.8 Mirroring Rudder repositories

You can also use your own packages repositories server instead of *www.rudder-project.org* if you want. This is possible with a synchronization from our repositories with rsync.

We've got public read only rsync modules *rudder-apt* and *rudder-rpm*.

To synchronize with the APT repository just type:

```
rsync -av www.rudder-project.org::rudder-apt /your/local/mirror
```

To synchronize with the RPM repository just type:

```
rsync -av www.rudder-project.org::rudder-rpm /your/local/mirror
```

Finally, you have to set up these directories (/your/local/mirror) to be shared by HTTP by a web server (i.e., *Apache*, nginx, lighttpd, etc…).

Chapter 12

Reference

This chapter contains the reference *Rudder* configuration files

12.1 Rudder Server data workflow

To have a better understanding of the Archive feature of *Rudder*, a description of the data workflow can be useful.

All the logic of *Rudder Techniques* is stored on the filesystem in `/var/rudder/configuration-repository/techniques`. The files are under version control, using git. The tree is organized as following:

1. At the first level, techniques are classified in categories: applications, fileConfiguration, fileDistribution, jobScheduling, system, systemSettings. The description of the category is included in `category.xml`.
2. At the second and third level, *Technique* identifier and version.
3. At the last level, each technique is described with a `metadata.xml` file and one or several *CFEngine* template files (name ending with `.st`).

An extract of Rudder Techniques filesystem tree

```
+-- techniques
|   +-- applications
|   |   +-- apacheServer
|   |   |   +-- 1.0
|   |   |       +-- apacheServerConfiguration.st
|   |   |       +-- apacheServerInstall.st
|   |   |       +-- metadata.xml
|   |   +-- aptPackageInstallation
|   |   |   +-- 1.0
|   |   |       +-- aptPackageInstallation.st
|   |   |       +-- metadata.xml
|   |   +-- aptPackageManagerSettings
|   |   |   +-- 1.0
|   |   |       +-- aptPackageManagerSettings.st
|   |   |       +-- metadata.xml
|   |   +-- category.xml
|   |   +-- openvpnClient
|   |   |   +-- 1.0
|   |   |       +-- metadata.xml
|   |   |       +-- openvpnClientConfiguration.st
|   |   |       +-- openvpnInstall.st
```

At *Rudder Server* startup, or after the user has requested a reload of the *Rudder Techniques*, each `metadata.xml` is mapped in memory, and used to create the *LDAP* subtree of *Active* Techniques. The *LDAP* tree contains also a set of subtrees for *Node* Groups, *Rules* and *Node Configurations*.

At each change of the *Node Configurations*, *Rudder Server* creates *CFEngine* draft policies (`Cf3PolicyDraft`) that are stored in memory, and then invokes `cf-clerk`. `cf-clerk` finally generates the *CFEngine* promises for the *Nodes*.

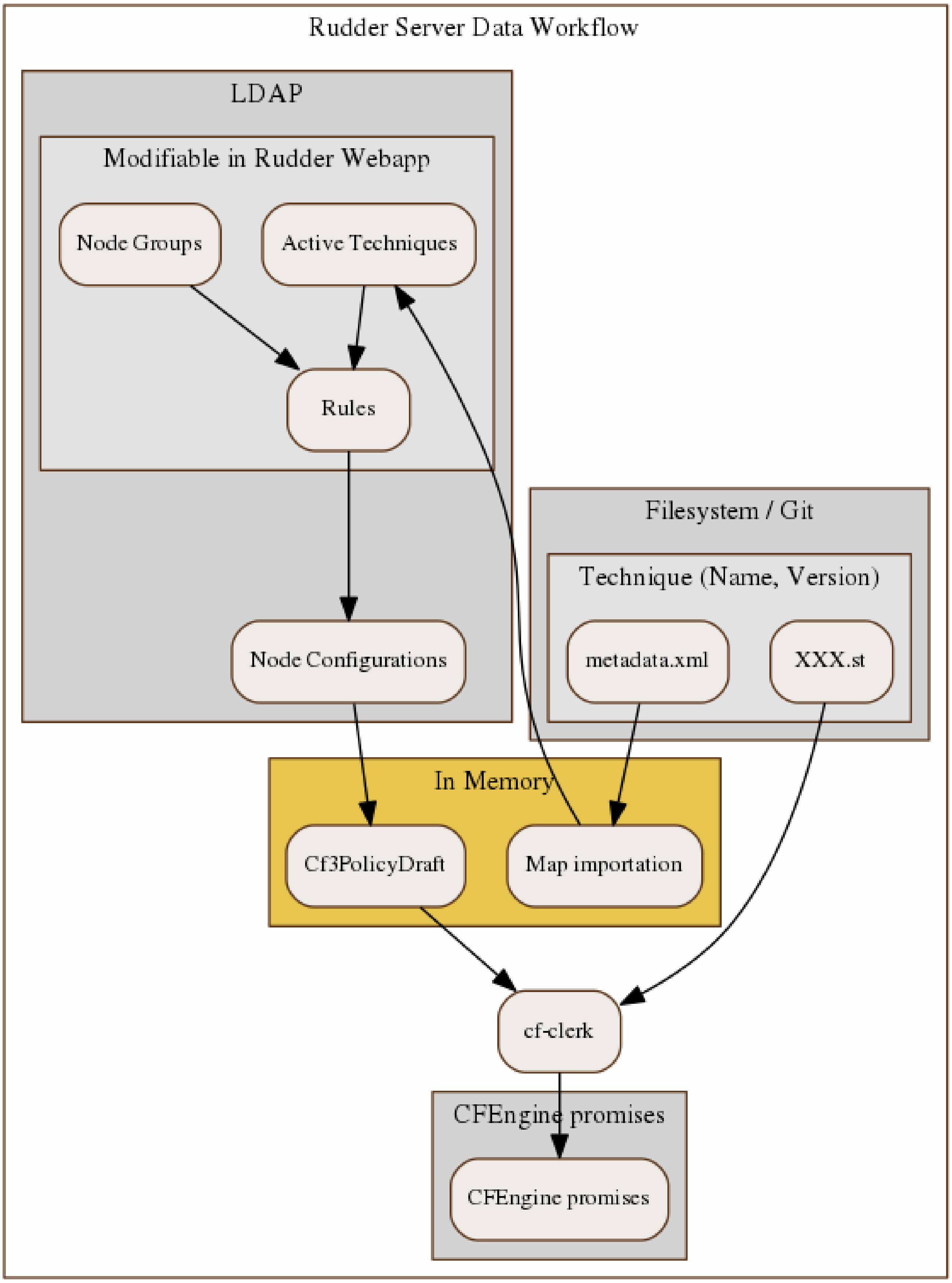


Figure 12.1: Rudder data workflow

12.2 Configuration files for Rudder Server

- /opt/rudder/etc/htpasswd-webdav
- /opt/rudder/etc/inventory-web.properties
- /opt/rudder/etc/logback.xml
- /opt/rudder/etc/openldap/slapd.conf
- /opt/rudder/etc/reportsInfo.xml
- /opt/rudder/etc/rudder-users.xml
- /opt/rudder/etc/rudder-web.properties

12.3 Rudder Agent workflow

In this chapter, we will have a more detailed 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.

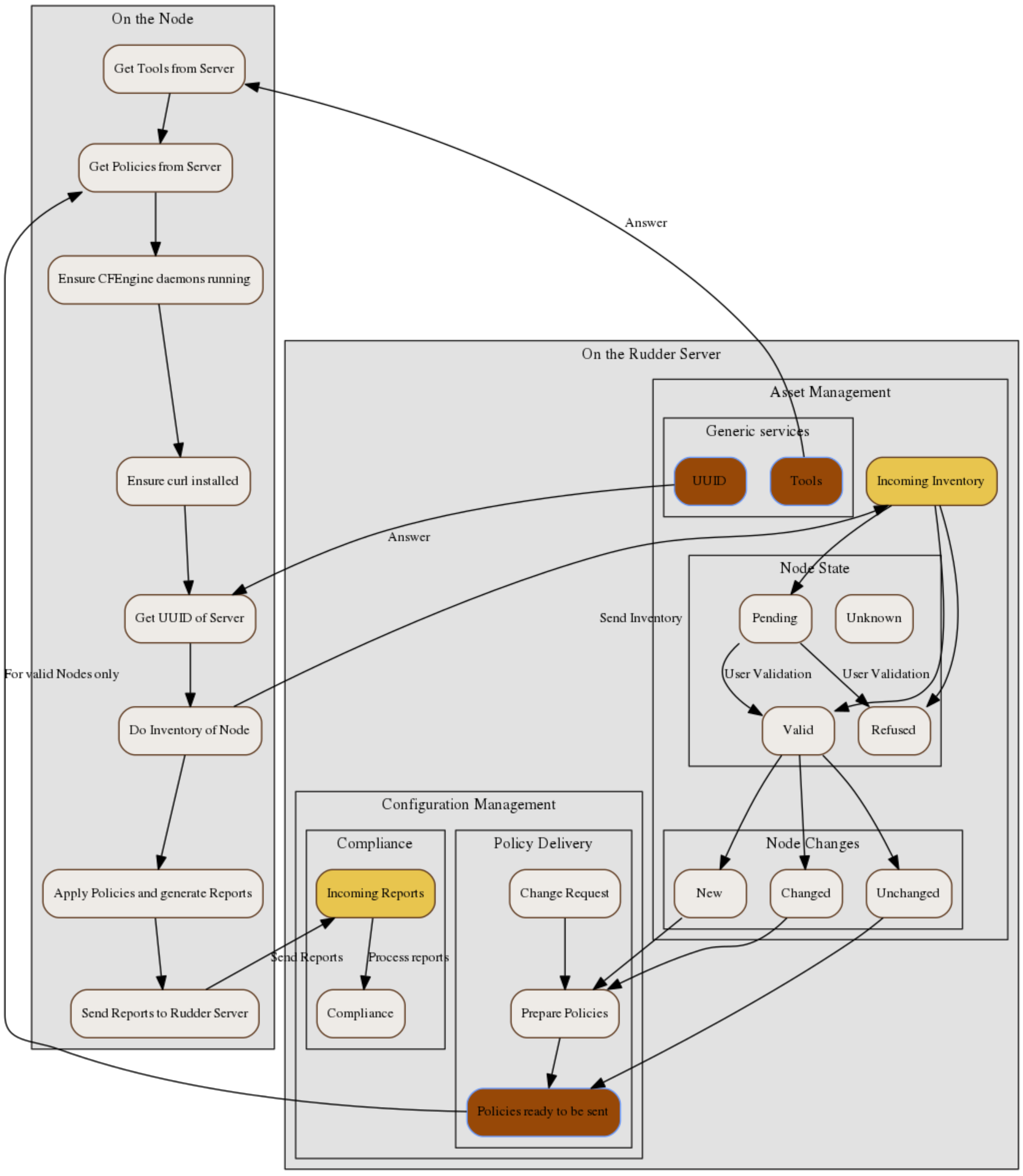


Figure 12.2: Rudder Agent workflow

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

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

12.3.1 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 Policies 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.

12.3.2 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.

12.3.3 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

12.3.4 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.

```
rudder agent inventory
```

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

12.3.5 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`.

12.3.6 Apply Directives

Apply other policies and write reports locally.

12.4 Configuration files for a Node

- /etc/default/rudder-agent

12.5 Packages organization

12.5.1 Packages

Rudder components are distributed as a set of packages.

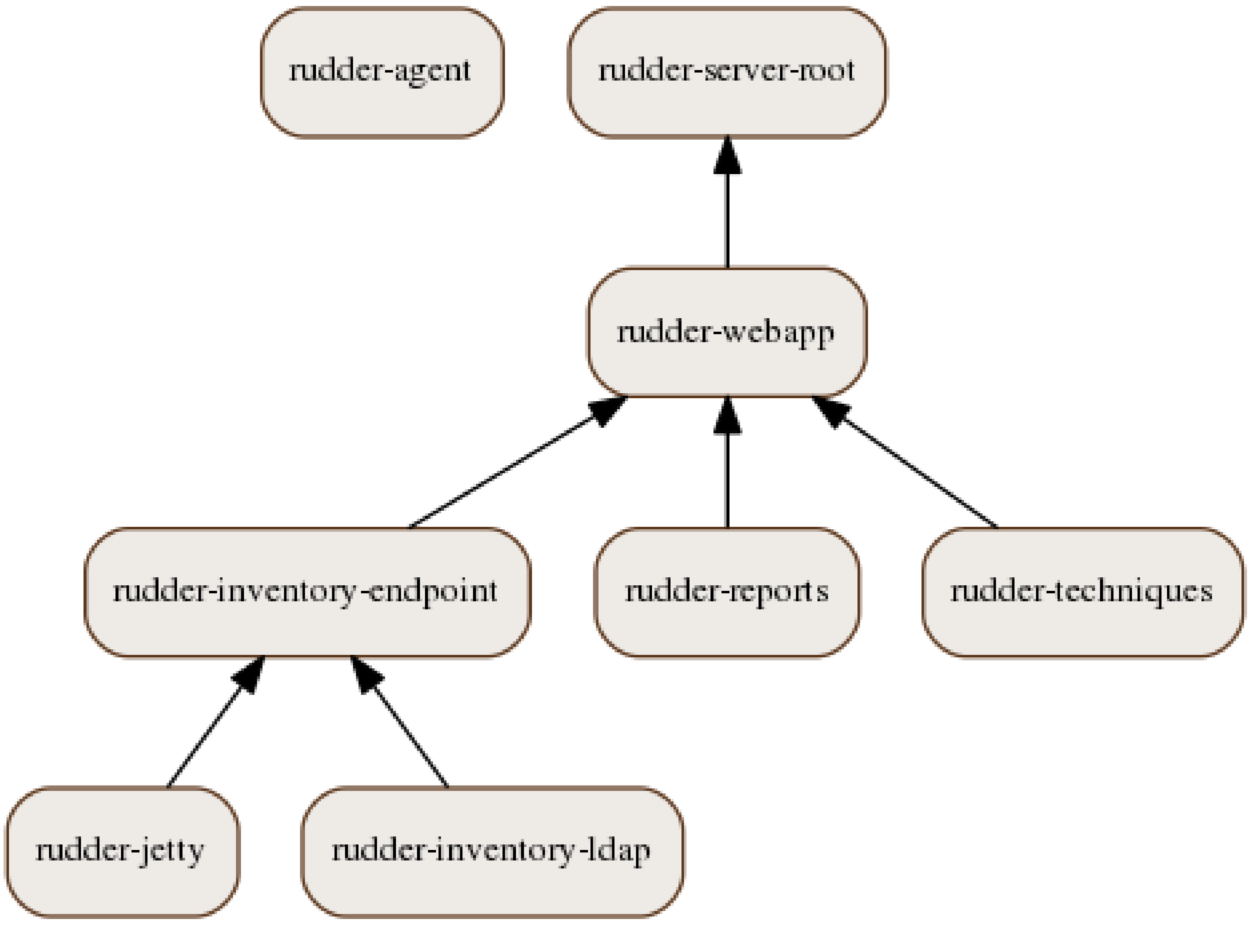


Figure 12.3: Rudder packages and their dependencies

`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 a compatible *Java* 7 Runtime Environment.

`rudder-techniques` Package for the *Techniques*. They are installed in `/opt/rudder/share/techniques`. At run-time, the *Techniques* are copied into a *git* repository in `/var/rudder/configuration-repository`. Therefore, the package depends on the `git` package.

`rudder-inventory-ldap` Package for the database containing the inventory and configuration information for each pending and validated *Node*. 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 dependency on the `postgresl` package, creates the database named `rudder` and installs the inialisation scripts for that database in `/opt/rudder/etc/postgresql/*.sql`.

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

- installs the *Rudder* configuration script:

```
/opt/rudder/bin/rudder-init
```

- 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 libraries and utilities:

- `OpenSSL`
- `libpcre`
- `liblmdb` (On platforms where it is available as a package - on others the rudder-agent package bundles it)
- `uuidgen`

12.5.2 Software dependencies and third party components

The *Rudder* Web application requires the installation of Apache *2 httpd*, Oracle Java *6 JRE* or *OpenJDK 7 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 (version >= 8.4) is used to store logs sent by the *Nodes* and reports generated by *Rudder*. PosgreSQL 9.1+ is recommended for optimal performances.

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.

***Java* 7 JRE** The *Java* runtime is needed by the Jetty application server. Where possible, the package from the distribution is used, else a *Java* RE must be downloaded from *Oracle*'s website (http://www.java.com).

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

git The running *Techniques* Library is maintained as a git repository in `/var/rudder/configuration-repository/techniques`.

12.6 Generic methods

This section documents all the generic methods available in the Technique Editor.

12.6.1 Command

12.6.1.1 command_execution

Execute a command

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **command_name**: Command name

Classes defined

```
command_execution_${command_name}_{kept, repaired, not_ok, reached}
```

12.6.1.2 command_execution_result

Execute a command and create outcome classes depending on its exit code

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Execute a command and create outcome classes depending on the exit codes given in parameters. If an exit code is not in the list it will lead to an error status. If you want 0 to be a success you have to list it in the kept_codes list

Parameters

- **command**: The command to run
- **kept_codes**: List of codes that produce a kept status separated with commas (ex: 1,2,5)
- **repaired_codes**: List of codes that produce a repaired status separated with commas (ex: 3,4,6)

Classes defined

```
command_execution_result_${command}_{kept, repaired, not_ok, reached}
```

12.6.2 Directory

12.6.2.1 directory_check_exists

Checks if a directory exists

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `directory_check_exists_${directory_name}_{ok, reached, kept}` if the directory exists, or `directory_check_exists_${directory_name}_{not_ok, reached, not_kept, failed}` if the directory doesn't exists

Parameters

- **directory_name**: Full path of the directory to check

Classes defined

```
directory_check_exists_${directory_name}_{kept, repaired, not_ok, reached}
```

12.6.2.2 directory_create

Create a directory if it doesn't exist

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **target**: Full path of directory to create (trailing / is optional)

Classes defined

```
directory_create_${target}_{kept, repaired, not_ok, reached}
```

12.6.3 File

12.6.3.1 file_check_FIFO_pipe

Checks if a file exists and is a FIFO/Pipe

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_FIFO_pipe_${file_name}_{ok, reached, kept}` if the file is a FIFO, or `file_check_FIFO_pipe_${file_name}_{not_ok, reached, not_kept, failed}` if the file is not a fifo or does not exist

Parameters

- **file_name**: File name

Classes defined

```
file_check_FIFO_pipe_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.2 file_check_block_device

Checks if a file exists and is a block device

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_block_device_${file_name}_{ok, reached, kept}` if the file is a block_device, or `file_check_block_device_${file_name}_{not_ok, reached, not_kept, failed}` if the file is not a block device or does not exist

Parameters

- **file_name**: File name

Classes defined

```
file_check_block_device_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.3 file_check_character_device

Checks if a file exists and is a character device

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_character_device_${file_name}_{ok, reached, kept}` if the file is a character device, or `file_check_character_device_${file_name}_{not_ok, reached, not_kept, failed}` if the file is not a character device or does not exist

Parameters

- **file_name**: File name

Classes defined

```
file_check_character_device_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.4 file_check_exists

Checks if a file exists

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_exists_${file_name}_{ok, reached, kept}` if the file exists, or `file_check_exists_${file_name}_{not_ok, reached, not_kept, failed}` if the file doesn't exists

Parameters

- **file_name**: File name

Classes defined

```
file_check_exists_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.5 file_check_hardlink

Checks if two files are the same (hard links)

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_hardlink_${file_name_1}_{ok, reached, kept}` if the two files `${file_name_1}` and `${file_name_2}` are hard links of each other, or `file_check_hardlink_${file_name_1}_{not_ok, reached, not_kept, failed}` if if the files are not hard links.

Parameters

- **file_name_1**: File name #1
- **file_name_2**: File name #2

Classes defined

```
file_check_hardlink_${file_name_1}_{kept, repaired, not_ok, reached}
```

12.6.3.6 file_check_regular

Checks if a file exists and is a regular file

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_regular_${file_name}_{ok, reached, kept}` if the file is a regular_file, or `file_check_regular_${file_name}_{not_ok, reached, not_kept, failed}` if the file is not a regular file or does not exist

Parameters

- **file_name**: File name

Classes defined

```
file_check_regular_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.7 file_check_socket

Checks if a file exists and is a socket

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_socket_${file_name}_{ok, reached, kept}` if the file is a socket, or `file_check_socket_${file_name}_{not_ok, reached, not_kept, failed}` if the file is not a socket or does not exist

Parameters

- **file_name**: File name

Classes defined

```
file_check_socket_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.8 file_check_symlink

Checks if a file exists and is a symlink

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_symlink_${file_name}_{ok, reached, kept}` if the file is a symlink, or `file_check_symlink_${file_name}_{not_ok, reached, not_kept, failed}` if the file is not a symlink or does not exist

Parameters

- **file_name**: File name

Classes defined

```
file_check_symlink_${file_name}_{kept, repaired, not_ok, reached}
```

12.6.3.9 file_check_symlinkto

Checks if first file is symlink to second file

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `file_check_symlinkto_${target}_{ok, reached, kept}` if the file `${symlink}` is a symbolic link to `${target}`, or `file_check_symlinkto_${target}_{not_ok, reached, not_kept, failed}` if if it is not a symbolic link, or any of the files does not exist. The symlink's path is resolved to the absolute path and checked against the target file's path, which must also be an absolute path.

Parameters

- **symlink**: Symbolic link (absolute path)
- **target**: Target file (absolute path)

Classes defined

```
file_check_symlinkto_${symlink}_{kept, repaired, not_ok, reached}
```

12.6.3.10 file_copy_from_local_source

This is a bundle to ensure that a file or directory is copied from a local source

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source**: Source file
- **destination**: Destination file

Classes defined

```
file_copy_from_local_source_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.11 file_copy_from_local_source_recursion

This is a bundle to ensure that a file or directory is copied from a local source

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source**: Source file
- **destination**: Destination file
- **recursion**: Recursion depth to enforce for this path (0, 1, 2, . . . , inf)

Classes defined

```
file_copy_from_local_source_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.12 file_copy_from_remote_source

This is a bundle to ensure that a file or directory is copied from a remote source

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Note: This method uses *CFEngine* file copy protocol, and can only download files from the policy server. To download a file from an external source, you can use HTTP with the file_download method.

This method requires that the policy server is configured to accept copy of the source file from the agents it will be applied to.

You have to write the full path of the file on the policy server, for example:

```
/home/myuser/myfile
```

If you are using *Rudder*, you can download a file from the shared files with:

```
/var/rudder/configuration-repository/shared-files/PATH_TO_YOUR_FILE
```

Parameters

- **source**: Source file
- **destination**: Destination file

Classes defined

```
file_copy_from_remote_source_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.13 file_copy_from_remote_source_recursion

This is a bundle to ensure that a file or directory is copied from a remote source

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This method requires that the policy server is configured to accept copy of the source file or directory from the agents it will be applied to.

You have to write the full path of the file or directory on the policy server, for example:

```
/home/myuser/mydirectory
```

If you are using *Rudder*, you can download a file from the shared files with:

```
/var/rudder/configuration-repository/shared-files/PATH_TO_YOUR_DIRECTORY_OR_FILE
```

Parameters

- **source**: Source file
- **destination**: Destination file
- **recursion**: Recursion depth to enforce for this path (0, 1, 2, . . . , inf)

Classes defined

```
file_copy_from_remote_source_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.14 file_create

Create a file if it doesn’t exist

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **target**: File to create

Classes defined

```
file_create_${target}_{kept, repaired, not_ok, reached}
```

12.6.3.15 file_create_symlink

This is a bundle to create a symlink at a destination path and pointing to a source target except if a file or directory already exists.

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source**: Source file
- **destination**: Destination file

Classes defined

```
file_create_symlink_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.16 file_create_symlink_enforce

This is a bundle to create a symlink at a destination path and pointing to a source target. This is also possible to enforce its creation

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source**: Source file
- **destination**: Destination file
- **enforce**: Force symlink if file already exist (true or false)

Classes defined

```
file_create_symlink_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.17 file_create_symlink_force

This is a bundle to create a symlink at a destination path and pointing to a source target even if a file or directory already exists.

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source**: Source file
- **destination**: Destination file

Classes defined

```
file_create_symlink_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.18 file_download

Download a file if it does not exit, using curl with a fallback on wget

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This method finds a HTTP command-line tool and downloads the given source into the destination.

It tries `curl` first, and `wget` as fallback.

Parameters

- **source**: URL to download from
- **destination**: File destination

Classes defined

```
file_download_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.19 file_enforce_content

This is a bundle to enfore the content of a file

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **lines**: Line(s) to add in the file
- **enforce**: Enforce the file to contain only line(s) defined (true or false)

Classes defined

```
file_ensure_lines_present_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.20 file_ensure_block_in_section

This is a bundle to ensure that a section contains exactly a text block

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **section_start**: Start of the section
- **section_end**: End of the section
- **block**: Block representing the content of the section

Classes defined

```
file_ensure_block_in_section_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.21 file_ensure_block_present

This is a bundle to ensure that a text block is present in a specific location

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **block**: Block(s) to add in the file

Classes defined

```
file_ensure_block_present_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.22 file_ensure_key_value

Ensure that the file contains a pair of "key separator value"

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Edit (or create) the file, and ensure it contains an entry key → value with arbitrary separator between the key and its value. If the key is already present, the method will change the value associated with this key.

Parameters

- **file**: File name to edit
- **key**: Key to define
- **value**: Value to define
- **separator**: Separator between key and value (for example "=" or " ")

Classes defined

```
file_ensure_key_value_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.23 file_ensure_key_value_present_in_ini_section

This is a bundle to ensure that a key-value pair is present in a section in a specific location. The objective of this method is to handle INI-style files.

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **section**: Name of the INI-style section under which the line should be added or modified (not including the [] brackets)
- **name**: Name of the key to add or edit
- **value**: Value of the key to add or edit

Classes defined

```
file_ensure_key_value_present_in_ini_section_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.24 file_ensure_keys_values

Ensure that the file contains all pairs of "key separator value", with arbitrary separator between each key and its value

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

You can create a dict variable by using the variable_dict* methods.

Parameters

- **file**: File name to edit
- **keys**: Dict structure containing the keys (keys of the dict), and values to define (values of the dict)
- **separator**: Separator between key and value (for example "=" or " ")

Classes defined

```
file_ensure_keys_values_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.25 file_ensure_line_present_in_ini_section

This is a bundle to ensure that a line is present in a section in a specific location. The objective of this method is to handle INI-style files.

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **section**: Name of the INI-style section under which lines should be added (not including the [] brackets)
- **line**: Line to ensure is present inside the section

Classes defined

```
file_ensure_line_present_in_ini_section_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.26 file_ensure_line_present_in_xml_tag

This is a bundle to ensure that a line is present in a tag in a specific location. The objective of this method is to handle XML-style files. Note that if the tag is not present in the file, it won't be added, and the edition will fail.

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **tag**: Name of the XML tag under which lines should be added (not including the <> brackets)
- **line**: Line to ensure is present inside the section

Classes defined

```
file_ensure_line_present_in_xml_tag_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.27 file_ensure_lines_absent

This is a bundle to ensure that a line is absent in a specific location

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **lines**: Line(s) to remove in the file

Classes defined

```
file_ensure_lines_absent_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.28 file_ensure_lines_present

This is a bundle to ensure that one or more lines are present in a file

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **lines**: Line(s) to add in the file

Classes defined

```
file_ensure_lines_present_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.29 file_from_template

This is a bundle to build a file from a template

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source_template**: Source file containing a template to be expanded
- **destination**: Destination file

Classes defined

```
file_from_template_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.30 file_from_template_mustache

This is a bundle to build a file from a mustache template

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source_template**: Source file containing a template to be expanded
- **destination**: Destination file

Classes defined

```
file_from_template_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.31 file_from_template_type

This is a bundle to build a file from a template

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **source_template**: Source file containing a template to be expanded
- **destination**: Destination file
- **template_type**: Template type (cfengine or mustache)

Classes defined

```
file_from_template_${destination}_{kept, repaired, not_ok, reached}
```

12.6.3.32 file_remove

Remove a file if it exists

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **target**: File to remove

Classes defined

```
file_remove_${target}_{kept, repaired, not_ok, reached}
```

12.6.3.33 file_replace_lines

This is a bundle to ensure that a line in a file is replaced by another one

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **file**: File name to edit
- **line**: Line to match in the file
- **replacement**: Line to add in the file as a replacement

Classes defined

```
file_replace_lines_${file}_{kept, repaired, not_ok, reached}
```

12.6.3.34 file_template_expand

This is a bundle to expand a template in a specific location

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **tml_file**: File name (with full path within the framework) of the template file
- **target_file**: File name (with full path) where to expand the template
- **mode**: Mode of destination file
- **owner**: Owner of destination file
- **group**: Froup of destination file

Classes defined

```
file_template_expand_${target_file}_{kept, repaired, not_ok, reached}
```

12.6.4 Group

12.6.4.1 group_absent

Make sure a group is absent

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **group**: *Group* name

Classes defined

```
group_absent_${group}_{kept, repaired, not_ok, reached}
```

12.6.4.2 group_present

Create a group

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **group**: *Group* name

Classes defined

```
group_present_${group}_{kept, repaired, not_ok, reached}
```

12.6.5 Http

12.6.5.1 http_request_check_status_headers

Checks status of an HTTP URL

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Perform a HTTP request on the URL, method and headers provided and check that the response has the expected status code (ie 200, 404, 503, etc)

Parameters

- **method**: Method to call the URL (GET, POST, PUT, DELETE)
- **url**: URL to query
- **expected_status**: Expected status code of the HTTP response
- **headers**: Headers to include in the HTTP request (as a string, without ’)

Classes defined

```
http_request_check_status_headers_${url}_{kept, repaired, not_ok, reached}
```

12.6.5.2 http_request_content_headers

Make an HTTP request with a specific header

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Perform a HTTP request on the URL, method and headers provided and send the content provided. Will return an error if the request failed.

Parameters

- **method**: Method to call the URL (POST, PUT)
- **url**: URL to send content to
- **content**: Content to send
- **headers**: Headers to include in the HTTP request

Classes defined

```
http_content_headers_${url}_{kept, repaired, not_ok, reached}
```

12.6.6 Logger

12.6.6.1 logger_rudder

Logging output for *Rudder* reports

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **message**: The common part of the message to display
- **class_prefix**: The prefix of the class for different states

Classes defined

```
logger_rudder_${class_prefix}_{kept, repaired, not_ok, reached}
```

12.6.7 Package

12.6.7.1 package_check_installed

Verify if a package is installed in any version

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `package_check_installed_${file_name}_{ok, reached, kept}` if the package is installed, or `package_check_installed_${file_name}_{not_ok, reached, not_kept, failed}` if the package is not installed

Parameters

- **package_name**: Name of the package to check

Classes defined

```
package_check_installed_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.2 package_install

Install or update a package in its latest version available

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **package_name**: Name of the package to install

Classes defined

```
package_install_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.3 package_install_version

Install or update a package in a specific version

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **package_name**: Name of the package to install
- **package_version**: Version of the package to install (can be "latest" to install it in its latest version)

Classes defined

```
package_install_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.4 package_install_version_cmp

Install a package or verify if it is installed in a specific version, or higher or lower version than a version specified

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Example:

```
methods:
    "any" usebundle => package_install_version_cmp("postgresql", ">=", "9.1", "verify");
```

Parameters

- **package_name**: Name of the package to install or verify
- **version_comparator**: Comparator between installed version and defined version, can be ==,⇐,>=,<,>,!=
- **package_version**: The version of the package to verify (can be "latest" for latest version)
- **action**: Action to perform, can be add, verify (defaults to verify)

Classes defined

```
package_install_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.5 package_install_version_cmp_update

Install a package or verify if it is installed in a specific version, or higher or lower version than a version specified, optionally test update or not (*Debian*-, Red Hat- or *SuSE*-like systems only)

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Example:

```
methods:
    "any" usebundle => package_install_version_cmp_update("postgresql", ">=", "9.1", " ↩
        verify", "false");
```

Parameters

- **package_name**: Name of the package to install or verify
- **version_comparator**: Comparator between installed version and defined version, can be ==,⇐,>=,<,>,!=
- **package_version**: The version of the package to verify (can be "latest" for latest version)
- **action**: Action to perform, can be add, verify (defaults to verify)
- **update_policy**: While verifying packages, check against latest version ("true") or just installed ("false")

Classes defined

```
package_install_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.6 package_remove

Remove a package

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

Example:

```
methods:
    "any" usebundle => package_remove("htop");
```

Parameters

- **package_name**: Name of the package to remove

Classes defined

```
package_remove_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.7 package_verify

Verify if a package is installed in its latest version available

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **package_name**: Name of the package to verify

Classes defined

```
package_install_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.7.8 package_verify_version

Verify if a package is installed in a specific version

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **package_name**: Name of the package to verify
- **package_version**: Version of the package to verify (can be "latest" for latest version)

Classes defined

```
package_install_${package_name}_{kept, repaired, not_ok, reached}
```

12.6.8 Permissions

12.6.8.1 permissions

Set permissions on a file or directory (non recursively)

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **path**: Path to the file/directory
- **mode**: Mode to enforce (like "640")
- **owner**: Owner to enforce (like "root")
- **group**: *Group* to enforce (like "wheel")

Classes defined

```
permissions_${path}_{kept, repaired, not_ok, reached}
```

12.6.8.2 permissions_dirs

Verify if a directory has the right permissions non recursively

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **path**: Path of the directory
- **mode**: Mode to enfore
- **owner**: Owner to enforce
- **group**: *Group* to enforce

Classes defined

```
permissions_${path}_{kept, repaired, not_ok, reached}
```

12.6.8.3 permissions_dirs_recurse

Verify if a directory has the right permissions recursively

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **path**: Path to the directory
- **mode**: Mode to enforce
- **owner**: Owner to enforce
- **group**: *Group* to enforce

Classes defined

```
permissions_${path}_{kept, repaired, not_ok, reached}
```

12.6.8.4 permissions_recurse

Verify if a file or directory has the right permissions recursively

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **path**: Path to the file / directory
- **mode**: Mode to enforce
- **owner**: Owner to enforce
- **group**: *Group* to enforce

Classes defined

```
permissions_${path}_{kept, repaired, not_ok, reached}
```

12.6.8.5 permissions_type_recursion

This is a bundle to ensure that a file or directory is present and has the right mode/owner/group

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **path**: Path to edit
- **mode**: Mode of the path to edit
- **owner**: Owner of the path to edit
- **group**: *Group* of the path to edit
- **type**: Type of the path to edit (all/files/directories)
- **recursion**: Recursion depth to enforce for this path (0, 1, 2, . . . , inf)

Classes defined

```
permissions_${path}_{kept, repaired, not_ok, reached}
```

12.6.9 Schedule

12.6.9.1 schedule_simple

Trigger a repaired outcome when a job should be run

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run
- **mode**: "nodups": avoid duplicate runs in the same period / "catchup": avoid duplicates and one or more run have been missed, run once before next period / "stateless": no check is done on past runs

Classes defined

```
schedule_simple_${job_id}_{kept, repaired, not_ok, reached}
```

12.6.9.2 schedule_simple_catchup

Trigger a repaired outcome when a job should be run

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run

- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run
- **mode**: "nodups": avoid duplicate runs in the same period / "catchup": avoid duplicates and one or more run have been missed, run once before next period / "stateless": no check is done on past runs

Classes defined

```
schedule_simple_${job_id}_{kept, repaired, not_ok, reached}
```

12.6.9.3 schedule_simple_nodups

Trigger a repaired outcome when a job should be run

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run
- **mode**: "nodups": avoid duplicate runs in the same period / "catchup": avoid duplicates and one or more run have been missed, run once before next period / "stateless": no check is done on past runs

Classes defined

```
schedule_simple_${job_id}_{kept, repaired, not_ok, reached}
```

12.6.9.4 schedule_simple_stateless

Trigger a repaired outcome when a job should be run

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run
- **mode**: "nodups": avoid duplicate runs in the same period / "catchup": avoid duplicates and one or more run have been missed, run once before next period / "stateless": no check is done on past runs

Classes defined

```
schedule_simple_${job_id}_{kept, repaired, not_ok, reached}
```

12.6.10 Service

12.6.10.1 service_action

Trigger an action on a service using tools like systemctl, service, init.d, *Windows*…

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service
- **action**: Action to trigger on the service (start, stop, restart, reload, …)

Classes defined

```
service_action_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.2 service_check_running

Check if a service is running using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Process name

Classes defined

```
service_check_running_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.3 service_check_running_ps

Check if a service is running using ps

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_regex**: Regular expression used to select a process in ps output

Classes defined

```
service_check_running_${service_regex}_{kept, repaired, not_ok, reached}
```

12.6.10.4 service_check_started_at_boot

Check if a service is set to start at boot using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service name (as recognized by systemd, init.d, etc…)

Classes defined

```
service_check_started_at_boot_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.5 service_ensure_running

Ensure that a service is running using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service name (as recognized by systemd, init.d, etc…)

Classes defined

```
service_ensure_running_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.6 service_ensure_running_path

Ensure that a service is running using the appropriate method, specifying the path of the service in the ps output, or using *Windows* task manager

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service name (as recognized by systemd, init.d, *Windows*, etc...)
- **service_path**: Service with its path, as in the output from *ps*

Classes defined

```
service_ensure_running_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.7 service_ensure_started_at_boot

Force a service to be started at boot

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service name (as recognized by systemd, init.d, *Windows*, SRC, SMF, etc...)

Classes defined

```
service_ensure_started_at_boot_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.8 service_ensure_stopped

Ensure that a service is stopped using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service

Classes defined

```
service_ensure_stopped_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.9 service_reload

Reload a service using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service

Classes defined

```
service_reload_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.10 service_restart

Restart a service using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Name of the service to restart in systemd, init.d, . . .

Classes defined

```
service_restart_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.11 service_restart_if

Restart a service using the appropriate method if the specified class is true, otherwise it is considered as not required and success classes are returned.

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service
- **trigger_class**: class(es) which will trigger the restart of Service "(package_service_installed|service_conf_changed)" by example

Classes defined

```
service_restart_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.12 service_start

Start a service using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service

Classes defined

```
service_start_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.10.13 service_stop

Stop a service using the appropriate method

Compatible with nodes running *Rudder* 2.11 or higher.

Parameters

- **service_name**: Service

Classes defined

```
service_stop_${service_name}_{kept, repaired, not_ok, reached}
```

12.6.11 User

12.6.11.1 user_absent

Remove a user

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This method ensures that a user does not exist on the system.

Parameters

- **login**: User login

Classes defined

```
user_absent_${login}_{kept, repaired, not_ok, reached}
```

12.6.11.2 user_create

Create a user

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

This method does not create the user's home directory.

Parameters

- **login**: User login
- **description**: User description
- **home**: User's home directory
- **group**: User's primary group
- **shell**: User's shell
- **locked**: Is the user locked ? true or false

Classes defined

```
user_create_${login}_{kept, repaired, not_ok, reached}
```

12.6.12 Variable

12.6.12.1 variable_dict

Define a variable that contains key,value pairs (a dictionnary)

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

To use the generated variable, you must use the form `${variable_prefix.variable_name[key]}` with each name replaced with the parameters of this method.

Be careful that using a global variable can lead to unpredictable content in case of multiple definition, which is implicitly the case when a technique has more than one instance (directive). Please note that only global variables are available within templates.

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **value**: The variable content in JSON format

Classes defined

```
variable_dict_${variable_name}_{kept, repaired, not_ok, reached}
```

12.6.12.2 variable_dict_from_file

Define a variable that contains key,value pairs (a dictionnary) from a JSON file

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

To use the generated variable, you must use the form `${variable_prefix.variable_name[key]}` with each name replaced with the parameters of this method.

Be careful that using a global variable can lead to unpredictable content in case of multiple definition, which is implicitly the case when a technique has more than one instance (directive). Please note that only global variables are available within templates.

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **file_name**: The file name with JSON content

Classes defined

```
variable_dict_from_file_${variable_name}_{kept, repaired, not_ok, reached}
```

12.6.12.3 variable_iterator

Define a variable that will be automatically iterated over

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

The generated variable is a special variable (slist in cfengine speaking) that is automatically iterated over. When you call a generic method with this variable as a parameter, n calls will be made, one for each items of the variable. Note: there is a limit of 10000 items

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **value**: The variable content
- **separator**: Regular expression that is used to split the value into items (usually: ,)

Classes defined

```
variable_iterator_${variable_name}_{kept, repaired, not_ok, reached}
```

12.6.12.4 variable_iterator_from_file

Define a variable that will be automatically iterated over

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

The generated variable is a special variable (slist in cfengine speaking) that is automatically iterated over. When you call a generic method with this variable as a parameter, n calls will be made, one for each items of the variable. Note: there is a limit of 10000 items Note: empty items are ignored

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **file_name**: The path to the file
- **separator_regex**: Regular expression that is used to split the value into items (usually:)
- **comments_regex**: Regular expression that is used to remove comments (usually: #.*?(?=))

Classes defined

```
variable_iterator_from_file_${variable_name}_{kept, repaired, not_ok, reached}
```

12.6.12.5 variable_string

Define a variable from a string parameter

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

To use the generated variable, you must use the form `${variable_prefix.variable_name}` with each name replaced with the parameters of this method.

Be careful that using a global variable can lead to unpredictable content in case of multiple definition, which is implicitly the case when a technique has more than one instance (directive). Please note that only global variables are available within templates.

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **value**: The variable content

Classes defined

```
variable_string_${variable_name}_{kept, repaired, not_ok, reached}
```

12.6.12.6 variable_string_from_file

Define a variable from a file content

Compatible with nodes running *Rudder* 2.11 or higher.

Usage

To use the generated variable, you must use the form `${variable_prefix.variable_name}` with each name replaced with the parameters of this method.

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **file_name**: The path of the file

Classes defined

```
variable_string_from_file_${variable_name}_{kept, repaired, not_ok, reached}
```

12.7 Man pages

12.7.1 rudder(8)

12.7.1.1 NAME

rudder - execute commands to control the *Rudder* configuration management tool.

12.7.1.2 SYNOPSIS

rudder *component* [*option*] *command*

rudder *component* help

12.7.1.3 DESCRIPTION

A tool to trigger actions or get information about a running rudder-agent, whether on agent or server. It only targets administration actions, for all node configuration tasks you can use the rudder-cli tool.

12.7.1.4 OPTIONS

-h Print command-line syntax and command options.

-v Print detailed information.

-d Print all available information.

12.7.1.5 COMMANDS

====== agent

commands for rudder agent, run with **rudder agent** *command*

disable forbid rudder-agent to be run by cron or service. This is useful when you want to temporarily prevent your *Rudder* agent from doing any modification to your system.

enable re-enable a disabled rudder-agent.

Options:

-s: start rudder-agent in addition to enabling it

inventory force the agent to create and send a new inventory. This will trigger a new inventory creation and send it to the policy server. Even if the agent will do it regularly, it can be used to force the update after a modification on the node. This won't affect the node state, but only update server-side information.

Options:

-q: run the agent in quiet mode (display only error messages)

reinit re-initialise the agent to make it be seen as a new node on the server. This command will delete all local agent data, including its uuid and keys, and also reset the agent internal state. The only configuration kept is the server hostname or ip configured in *policy_server.dat*. It will also send an inventory to the server, which will treat it as a new node inventory.

WARNING: This command will permanently delete your node uuid and keys, and no configuration will be applied before re-accepting and configuring the node on the server.

reset reset agent status and cache. Remove all locks and state cache of the agent, and restore initial promises. This won't affect the desired state of the node, but will only reset the internal state of the agent. It is useful to test a rule without caching interference or when you have trouble with the promises updates.

run force run agent promises. This command will force the agent to enforce current policies. You can run **rudder agent update** before to update the promises.

Options:

-v: run the agent in verbose mode

-q: run the agent in quiet mode (display only error messages)

update update promises on agent. The agent will fetch the last version of its promises from its configured policy server.

Options:

-f: force full update

version get the agent version. Displays the version of the *Rudder* agent and of the underlying *CFEngine* agent.

====== server

commands for rudder server, run with **rudder server** *command*

debug run a debug `cf-serverd` intended for a specific node. This command targets a specific node and does not affect the running infrastructure. It uses *iptables* to redirect the specific node communications to the port the debug server is listening on (5310 by default).

Use Ctrl+C to stop the debug server.

Arguments:

node: IP or hostname of the host you want to debug

12.7.1.6 AUTHOR

Normation SAS (contact@normation.com)

12.7.1.7 RESOURCES

Main web site: https://rudder-project.org/

Sources of the rudder command-line: https://github.com/*Normation*/rudder-agent/

12.7.1.8 COPYING

Copyright (C) 2014-2015 *Normation* SAS.

Chapter 13

Handbook

This chapter contains some tips and tricks you might want to know about using *Rudder* in a production environment, with some useful optimization and procedures.

13.1 Database maintenance

Rudder uses two backends to store information as of now: *LDAP* and SQL

To achieve this, OpenLDAP and PostgreSQL are installed with *Rudder*.

However, like every database, they require a small amount of maintenance to keep operating well. Thus, this chapter will introduce you to the basic maintenance procedure you might want to know about these particular database implementations.

13.1.1 Automatic PostgreSQL table maintenance

Rudder uses an automatic mechanism to automate the archival and pruning of the reports database.

By default, this system will:

- Archive reports older that 3 days (30 in *Rudder* 2.6)
- Remove reports older than 90 days

It thus reduces the work overhead by only making *Rudder* handle relevant reports (fresh enough) and putting aside old ones.

This is obviously configurable in /opt/rudder/etc/rudder-web.properties, by altering the following configuration elements:

- rudder.batch.reportscleaner.archive.TTL: Set the maximum report age before archival
- rudder.batch.reportscleaner.delete.TTL: Set the maximum report age before deletion

The default values are OK for systems under moderate load, and should be adjusted in case of excessive database bloating.

The estimated disk space consumption, with a 5 minute agent run frequency, is 150 to 400 kB per *Directive*, per day and per node, which is roughly 5 to 10 MB per *Directive* per month and per node.

Thus, 25 directives on 100 nodes, with a 7 day log retention policy, would take 2.5 to 10 GB, and 25 directives on 1000 nodes with a 1 hour agent execution period and a 30 day log retention policy would take 9 to 35 GB.

13.1.2 PostgreSQL database vacuum

In some cases, like a large report archiving or deletion, the *Rudder* interface will still display the old database size. This is because even if the database has been cleaned as requested, the physical storage backend did not reclaim space on the hard drive, resulting in a "fragmented" database. This is not an issue, as PostgreSQL handles this automatically, and new reports sent by the nodes to *Rudder* will fill the blanks in the database, resulting in a steady growth of the database. This task is handled by the autovacuum process, which periodically cleans the storage regularly to prevent database bloating.

However, to force this operation to free storage immediately, you can trigger a "vacuum full" operation by yourself, however keep in mind that this operation is very disk and memory intensive, and will lock both the *Rudder* interface and the reporting system for quite a long time with a big database.

Manual vacuuming using the psql binary

```
#~You can either use sudo to change owner to the postgres user, or use the rudder ↩
    connection credentials.

#~With sudo:
sudo -u postgres psql -d rudder

#~With rudder credentials, it will ask the password in this case:
psql -u rudder -d rudder -W

# And then, when you are connected to the rudder database in the psql shell, trigger a ↩
    vacuum:
rudder=# VACUUM FULL;

# And take a coffee.
```

13.1.3 LDAP database reindexing

In some very rare case, you will encounter some *LDAP* database entries that are not indexed and used during searches. In that case, OpenLDAP will output warnings to notify you that they should be.

LDAP database reindexing

```
# Stop OpenLDAP
service rudder-slapd stop

# Reindex the databases
service rudder-slapd reindex

# Restart OpenLDAP
service rudder-slapd restart
```

13.2 Migration, backups and restores

It is advised to backup frequently your *Rudder* installation in case of a major outage.

These procedures will explain how to backup your *Rudder* installation.

13.2.1 Backup

This backup procedure will operate on the three principal *Rudder* data sources:

- The *LDAP* database

- The PostgreSQL database
- The configuration-repository folder

It will also backup the application logs.

How to backup a Rudder installation

```
#~First, backup the LDAP database:
/opt/rudder/sbin/slapcat -l /tmp/rudder-backup-$(date +%Y%m%d).ldif

# Second, the PostgreSQL database:
sudo -u postgres pg_dump rudder > /tmp/rudder-backup-$(date +%Y%m%d).sql

#~Or without sudo, use the rudder application password:
pg_dump -U rudder rudder > /tmp/rudder-backup-$(date +%Y%m%d).sql

#~Third, backup the configuration repository:
tar -C /var/rudder -zvcf /tmp/rudder-backup-$(date +%Y%m%d).tar.gz configuration-repository ↩
    / cfengine-community/ppkeys/

# Finally, backup the logs:
tar -C /var/log -zvcf /tmp/rudder-log-backup-$(date +%Y%m%d).tar.gz rudder/

#~And put the backups wherever you want, here /root:
cp /tmp/rudder-backup* /root
cp /tmp/rudder-log-backup* /root
```

13.2.2 Restore

Of course, after a total machine crash, you will have your backups at hand, but what should you do with it ?

Here is the restoration procedure:

How to restore a Rudder backup

```
# First, follow the standard installation procedure, this one assumes you have a working " ↩
    blank"
Rudder on the machine

# Stop Rudder
service rudder-server-root stop

# Drop the OpenLDAP database
rm -rf /var/rudder/ldap/openldap-data/alock /var/rudder/ldap/openldap-data/*.bdb /var/ ↩
    rudder/ldap/openldap-data/__db* /var/rudder/ldap/openldap-data/log*

# Import your backups

#~Configuration repository
tar -C /var/rudder -zvxf /root/rudder-backup-XXXXXXXX.tar.gz

#~LDAP backup
/opt/rudder/sbin/slapadd -l /root/rudder-backup-XXXXXXXX.ldif

#~PostgreSQL backup
sudo -u postgres psql -d rudder < /root/rudder-backup-XXXXXXXX.sql
#~or
psql -u rudder -d rudder -W < /root/rudder-backup-XXXXXXXX.sql

#~And restart the machine or just Rudder:
service rudder-server-root restart
```

13.2.3 Migration

To migrate a *Rudder* installation, just backup and restore your *Rudder* installation from one machine to another.

Please remember that The *CFEngine* key restoration is mandatory for the clients to update properly, but if the *Rudder* server address changes, the agents will block. You have to delete every root-*.pub key in /var/rudder/cfengine-community/ppkeys/ for things to work again.

13.3 Performance tuning

Rudder and some applications used by *Rudder* (like the *Apache* web server, or Jetty) can be tuned to your needs.

13.3.1 Reports retention

To lower *Rudder* server's disk usage, you can configure the retention duration for node's execution reports in `/opt/rudder/etc/rudder-web.properties` file with the options:

`rudder.batch.reportscleaner.archive.TTL=30`

`rudder.batch.reportscleaner.delete.TTL=90`

13.3.2 Apache HTTPd

The apache HTTPd is used by *Rudder* as a proxying server, to connect to the Jetty application server.

But it is also widely used as a regular HTTP serving application. You are heavily advised if interested to read the tons of documentation about it in your Linux distribution website, to learn about what it can do.

13.3.3 Jetty

The Jetty 7 (Hightide) application server is the main application that runs the *Rudder* code. It is based on the *Java* programming language.

About the latter, there is some configuration switches that you might want to tune to obtain better performance with *Rudder*, in /opt/rudder/etc/rudder-jetty.conf, whereas the default ones fit the basic recommendations for the minimal *Rudder* hardware requirements.

- JAVA_XMX : That parameter tune the total amount of RAM usable / dedicated to the java process. It is what you want to tune at first to give *Rudder* some more RAM.
- JAVA_MAXPERMSIZE: That parameter is acceptable for most installations, but you might want to decrease them a bit if using a machine that is not very powerful / RAM abundant. Increasing them is not really useful.

13.3.4 Java "Out Of Memory Error"

It may happen that you get java.lang.OutOfMemoryError. They can be of several types, but the most common is: "java.lang.OutOfMemo
Java heap space".

This error means that the web application needs more RAM than what was given. It may be linked to a bug where some process consumed much more memory than needed, but most of the time, it simply means that your system has grown and needs more memory.

You can follow the configuration steps described in the following paragraph.

13.3.5 Configure RAM allocated to Jetty

To change the RAM given to Jetty, you have to:

```
# edit /opt/rudder/etc/rudder-jetty.conf with your preferred text editor, for example vim:
vim /opt/rudder/etc/rudder-jetty.conf

# modify JAVA_XMX to set the value to your need.
# The value is given in MB by default, but you can also use the "G" unit to specify a size ↩
    in GB.

JAVA_XMX=2G

# save your changes, and restart Jetty:
service restart rudder-jetty
```

The amount of memory should be the half of the RAM of the server, rounded down to the nearest GB. For example, if the server has 5GB of RAM, 2GB should be used.

13.3.6 Optimize PostgreSQL server

The default out-of-the-box configuration of PostgreSQL server is really not compliant for high end (or normal) servers. It uses a really small amount of memory.

The location of the PostgreSQL server configuration file is usually:

```
/etc/postgresql/9.x/main/postgresql.conf
```

On a *SuSE* system:

```
/var/lib/pgsql/data/postgresql.conf
```

13.3.6.1 Suggested values on an high end server

```
#
# Amount of System V shared memory
# --------------------------------
#
# A reasonable starting value for shared_buffers is 1/4 of the memory in your
# system:

shared_buffers = 1GB

# You may need to set the proper amount of shared memory on the system.
#
#   $ sysctl -w kernel.shmmax=1073741824
#
# Reference:
# http://www.postgresql.org/docs/8.4/interactive/kernel-resources.html#SYSVIPC
#
# Memory for complex operations
# -----------------------------
#
# Complex query:

work_mem = 24MB
max_stack_depth = 4MB

# Complex maintenance: index, vacuum:
```

```
maintenance_work_mem = 240MB

# Write ahead log
# ---------------
#
# Size of the write ahead log:

wal_buffers = 4MB

# 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:

effective_cache_size = 1024MB
```

13.3.6.2 Suggested values on a low end server

```
shared_buffers = 128MB
work_mem = 8MB
max_stack_depth = 3MB
maintenance_work_mem = 64MB
wal_buffers = 1MB
effective_cache_size = 128MB
```

13.3.7 CFEngine

If you are using *Rudder* on a highly stressed machine, which has especially slow or busy I/O's, you might experience a sluggish *CFEngine* agent run everytime the machine tries to comply with your *Rules*.

This is because the *CFEngine* agent tries to update its class database everytime the agent executes a promise (the cf-lock.db file in the /var/rudder/cfengine-community/state directory), which even if the database is very light, takes some time if the machine has a very high iowait value.

In this case, here is a workaround you can use to restore *CFEngine*'s full speed: you can use a RAMdisk to store *CFEngine* states.

You might use this solution either temporarily, to examine a slowness problem, or permanently, to mitigate a known I/O problem on a specific machine. We do not recommend as of now to use this on a whole IT infrastructure.

Be warned, this solution has only one drawback: you should backup and restore the content of this directory manually in case of a machine reboot because all the persistent states are stored here, so in case you are using, for example the jobScheduler *Technique*, you might encounter an unwanted job execution because *CFEngine* will have "forgotten" the job state.

Also, note that the mode=0700 is important as *CFEngine* will refuse to run correctly if the state directory is world readable, with an error like:

```
error: UNTRUSTED: State directory /var/rudder/cfengine-community (mode 770) was not private ↩
    !
```

Here is the command line to use:

How to mount a RAMdisk on CFEngine state directory

```
# How to mount the RAMdisk manually, for a "one shot" test:
mount -t tmpfs -o size=128M,nr_inodes=2k,mode=0700,noexec,nosuid,noatime,nodiratime tmpfs / ↩
    var/rudder/cfengine-community/state

# How to put this entry in the fstab, to make the modification permanent
echo "tmpfs /var/rudder/cfengine-community/state tmpfs defaults,size=128M,nr_inodes=2k,mode ↩
    =0700,noexec,nosuid,noatime,nodiratime 0 0" >> /etc/fstab
mount /var/rudder/cfengine-community/state
```

13.3.8 Rsyslog

On *Rudder* policy servers (root or relay), when managing a large number of nodes, you can experience issues with rsyslog. This happens because *Rudder* uses TCP by default for sending reports to rsyslog, which implies the system has to keep track of the connections. It can lead to reach some limits, especially:

- max number of open files for the user running rsyslog
- size of network backlogs
- size of the conntrack table

All settings needing to modify */etc/sysctl.conf* require to run *sysctl -p* to be applied.

13.3.8.1 Maximum number of file descriptors

If you plan to manage hundreds of *Nodes* behind a relay or a root server, you should increase the open file limit (10k is a good starting point, you might have to get to 100k with thousands of *Nodes*).

You can change the system-wide maximum number of file descriptors in */etc/sysctl.conf* if necessary:

```
fs.file-max = 100000
```

Then you have to get the user running rsyslog enough file descriptors. To do so, you have to:

- Have a high enough hard limit for rsyslog
- Set the limit used by rsyslog

The first one can be set in */etc/security/limits.conf*:

```
username hard nofile 8192
```

For the second one, you have two options:

- Set the soft limit (which will be used by default) in */etc/security/limits.conf* (with *username soft nofile 8192*)
- If you want to avoid changing soft limit (particularly if rsyslog is running as root), you can configure rsyslog to change its limit to a higher value (but not higher than the hard limit) with the *$MaxOpenFiles* configuration directive in */etc/rsyslog.conf*

You have to restart rsyslog for these settings to take effect.

You can check current soft and hard limits by running the following commands as the user you want to check:

```
$ ulimit -Sn
$ ulimit -Hn
```

13.3.8.2 Network backlog

You can also have issues with the network queues (which may for example lead to sending SYN cookies):

- You can increase the maximum number of connection requests awaiting acknowledgment by changing *net.ipv4.tcp_max_syn_backlog = 4096* (for example, the default is 1024) in */etc/sysctl.conf*.
- You may also have to increase the socket listen() backlog in case of bursts, by changing *net.core.somaxconn = 1024* (for example, default is 128) in */etc/sysctl.conf*.

13.3.8.3 Conntrack table

You may reach the size of the conntrack table, especially if you have other applications running on the same server. You can increase its size in */etc/sysctl.conf*, see the Netfilter FAQ for details.

Chapter 14

Troubleshooting and common issues

This chapter covers common issues and the available solutions.

14.1 Some reports are in "No report"

14.1.1 If you get no reports at all for the Node

First thing to check is to see if reports were received by *Rudder* server.

Check the last report time (called **Last seen**) in **List *Nodes*** page. If you see:

- **Never**: your *Node* is misconfigured or has a communication issue with the server
- A date far (more than 15 minutes) from current time: Synchronize server and node time
- A recent date: check if the node has correctly updated

Now we will check if promises were updated on the *Node*. Maybe the node could not update its promises anymore, even if the reporting looks ok and *Rules* seems to be applied but report keeps in 'No report'.

To check if a node can update its promises, run (one the node) */var/rudder/cfengine-community/bin/cf-agent -KI* | *grep Update@@None* You'll get the result of the 'Update' component in the execution. The message at the end of the line will tell you if the update succeed of fail: *R: @@Common@@result_error@@hasPolicyServer-root@@common-root@@00@@Update@@None@@*
01-07 07:40:45-02:00##dda3cba8-f6ca-4766-a3cb-09d61f53f6c5@#Cannot update node's policy or dependencies

To update its promises, a *Node* needs to get a directory on *Rudder* server (/var/rudder/share/node_uuid), and *Rudder* checks if the node is authorized to access that directory. This check is based on the capability to resolve the ip as an accepted node. So if your node can't update its promises, it's probably because of a DNS issue!

14.1.2 If you get incomplete reporting for the Node

Some executions may be truncated, leading to incomplete reports. *Rudder* executions start with a *StartRun* report and ends with an *EndRun*. If the execution is incomplete and the *EndRun* never sent some reports will be missing. reasons could be:

- the execution of the agent encountered an error during its execution and could not complete, please report a bug
- the agent is executed to launch specific bundles
- reporting is missing on a *Technique*, in this case report a bug

14.2 Communication issues between agent and server

14.2.1 DNS issues

If one of the following problems happen:

- the agent does not manage to get its configuration back from the server with weird errors
- the server complains about being unable to resolve the node hostname
- when starting or restarting *Rudder* (or rudder-agent) service, `cf-serverd` start hangs

You probably have a name resolution problem. Please keep in mind that *Rudder* needs a working name resolution environment to operate properly, and therefore every machine should be at least able to resolve the name of their peer.

You have two options:

- Fix your DNS server or the `/etc/hosts` on both the server and the node, so they can resolve each other (you can check using nslookup). You need to restart rudder-agent on the server to apply it
- Disable hostname checking on the server in **Administration** → **Settings** → **Use reverse DNS lookups on nodes to reinforce authentication to policy server**. This is the preferred solution if you have nodes behind a NAT.

14.2.2 Inventory issues

If you cannot send inventories to the server, it may be because of a proxy configured in */etc/profile* or shell configuration. *Rudder* agents use cURL to send inventories to their server, and the server actually uses it too to send received inventories to the inventory web application. There are two solutions usable to prevent this problem:

- Disable the proxy temporarily in your shell session, so *Rudder* can operate freely:

```
unset http_proxy; unset https_proxy; unset ftp_proxy; unset ftps_proxy; unset HTTP_PROXY;  ↩
    unset HTTPS_PROXY; unset FTP_PROXY; unset FTPS_PROXY
```

- If you are using the Squid proxy, you are in luck, as the workaround might simply be to add this entry to your */etc/squid/squid.conf*: *ignore_expect_100 on*, it will make Squid more tolerant to programs like cURL than send some terse http requests. (Thanks to Albaro A. for this tip!)

14.3 Technique editing

If you have committed an invalid technique, then fixed it and commited it again, but the webapp still doesn't start, you have to force *Technique* library reloading.

To do this, deleting the attribute techniqueLibraryVersion from entry *techniqueCategoryId*=Active *Techniques,ou*=Rudder*,cn=rudder-configuration* in your *Rudder LDAP* backend. When re-starting, the webapp should now reload latest techniques.

14.4 Database is using too much space

Rudder stores a lot of data in the Postgresql database, and most historical data is removed from it. You can configure how many days of logs you want to keep in the database. However, due to the nature of Postgresql, when data are removed, space is not reclaimed on the storage system, it is simply marked as "free" for the database to write again in the removed rows. This space can be reclaimed by a VACUUM FULL, but it needs at least as much free space on the drive as the database size. Moreover, if you are using Postgresql 8.3 (or in a lesser extend 8.4), you’ll be likely to experience indexes bloating, where the physical size of the indexes grows without real reason, and need to be regularly purged.

There are two ways to reclaim space, the fast one (which doesn’t reclaim completely all wasted space), and the complete one (which is unfortunately very slow)

Fast solution (especially for 8.x version of postgresql): Simply reindexing the database will save some space; depending on the size of your database, it may take several minutes to a couple of hours

```
# First, stop the Rudder server
touch /opt/rudder/etc/disable-agent
/etc/init.d/rudder-jetty stop

# Then log into postgresql
psql -U rudder -h localhost
REINDEX TABLE ruddersysevents;

#Exit postgresql
\q

# Restart the Rudder server
rm /opt/rudder/etc/disable-agent
/etc/init.d/rudder-jetty start
```

Complete solution: this solution will reclaim all that can be reclaimed, but is really really slow (can last several hours)

```
# First, stop the Rudder server
touch /opt/rudder/etc/disable-agent
/etc/init.d/rudder-jetty stop

# Then log into postgresql
psql -U rudder -h localhost
drop index component_idx;
drop index composite_node_execution_idx;
drop index keyvalue_idx;
drop index nodeid_idx;
drop index ruleid_idx;
drop index executiontimestamp_idx;
vacuum full ruddersysevents; -- this will take several hours

create index nodeid_idx on RudderSysEvents (nodeId);
CREATE INDEX executionTimeStamp_idx on RudderSysEvents (executionTimeStamp);
CREATE INDEX composite_node_execution_idx on RudderSysEvents (nodeId, executionTimeStamp);
CREATE INDEX component_idx on RudderSysEvents (component);
CREATE INDEX keyValue_idx on RudderSysEvents (keyValue);
CREATE INDEX ruleId_idx on RudderSysEvents (ruleId);

# Exit postgresql
\q

# Restart the Rudder server
rm /opt/rudder/etc/disable-agent
/etc/init.d/rudder-jetty start
```

Chapter 15

Appendix: Glossary

***Active* Techniques** This is an organized list of the *Techniques* selected and modified by the user. By default this list is the same as the *Technique* Library. *Techniques* 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 *Technique* can appear only once in the *Active* Techniques list.

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

`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 on *Rudder Root* and Relay servers, serving policies and files to *Rudder Nodes*.

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

This version used to be called Nova before.

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

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

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

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.

Port 443, TCP, for users HTTP/S communication port, used to access the *Rudder* web interface.

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

Port 5308, TCP *CFEngine Enterprise* communication port, which is required to manage *Windows* nodes.

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

Port 5310, TCP *CFEngine* communication port, used to communicate the policies to the *Rudder* nodes when debugging communication between a *Node* and a policy server with the `rudder server debug` command.

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

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 an optional component in a *Rudder* architecture. They can act as a proxy for all network communications between *Rudder* agents and a *Rudder* server. This enables them to be installed in a remote datacenter, or inside a restricted network zone, to limit the network flows required to use *Rudder*.

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 Assessment software. *Rudder* associates Asset Management and Configuration Management. *Rudder* is a Free Software developed by *Normation*.

Rule It is the application of one or more directives to a group of nodes. It is the glue between both Asset Management and Configuration Management parts of the application.

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.

***Technique* Library** This is an organized list of all available *Techniques*. This list can’t be modified: every change made by a user will be applied to the Active *Techniques*.

Technique This is a configuration skeleton, adapted to a function or a particular service (e.g. 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 addresses of DNS servers, the default search box, …)

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