Rudder User Documentation

COLLABORATORS			
	TITLE : Rudder User Documentation		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jonathan Clarke, Nicolas Charles, and Fabrice Flore-Thebault	February 2012	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
2.3.0	October 2011	First release of the Rudder User Documentation.	NC, JC, FFT
2.4.0	February 2012	Rudder User Documentation for 2.4 release of Rudder.	NC, JC, FFT

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 informations produced by the asset management function to identify these *Nodes* and evaluate some specific informations 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 (eg DNS resolver configuration). This skeleton includes the configuration logic for this function or service, and can be set according to a list of variables (in the same example: IP 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 an 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.

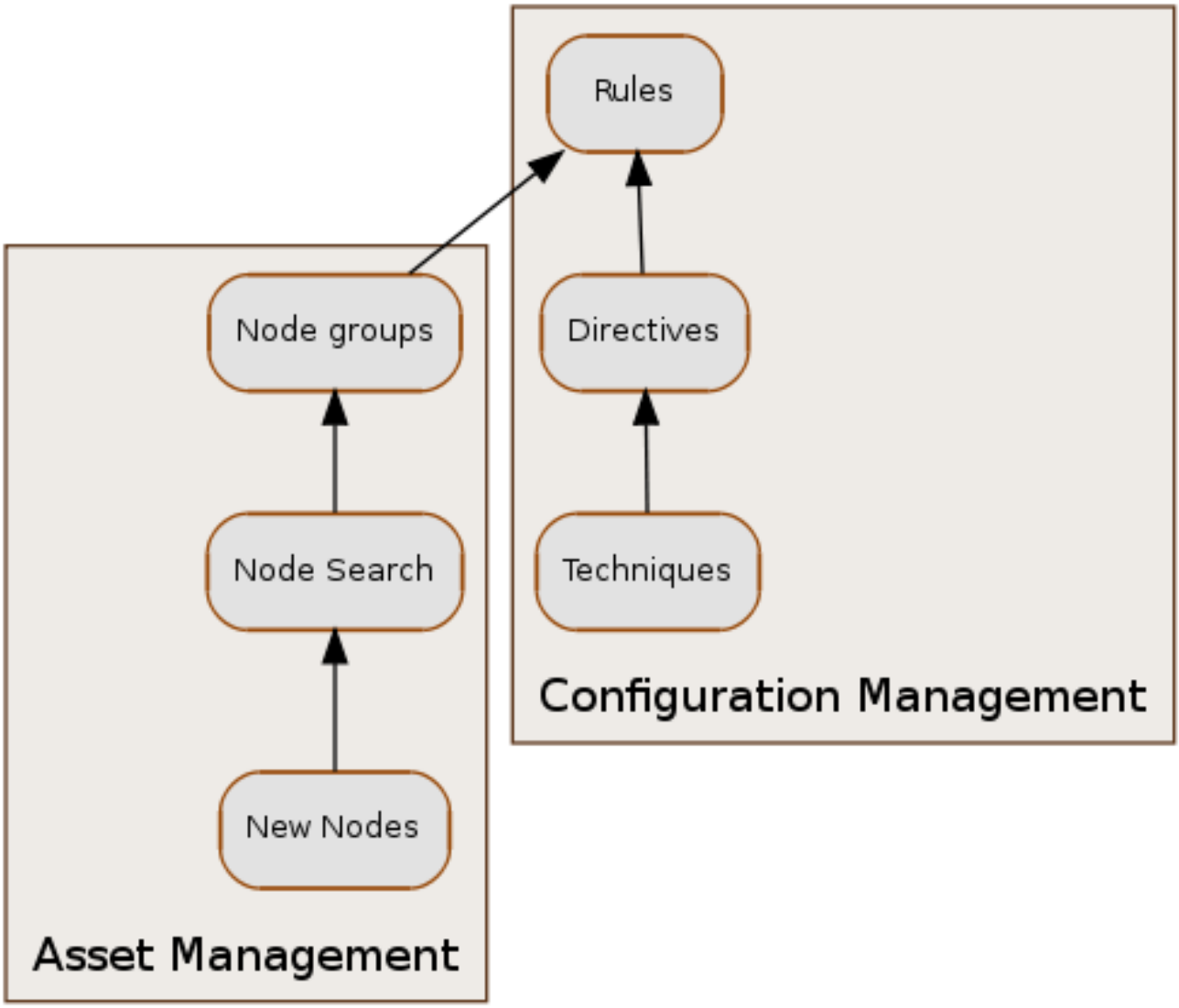

Figure 2.1: Concepts diagram

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

2.3 Specifications for Rudder Nodes

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

- *Debian* GNU/Linux 5 (*Lenny*)
- *Debian* GNU/Linux 6 (*Squeeze*)
- *Debian* GNU/Linux 7 (Wheezy)
- Microsoft Windows Server 2000
- Microsoft Windows Server 2003
- Microsoft Windows Server 2008
- Red Hat Enterprise Linux (RHEL) / *CentOS* 5
- Red Hat Enterprise Linux (RHEL) / *CentOS* 6
- *SuSE* Linux Enterprise Server (SLES) 10 SP3
- *SuSE* Linux Enterprise Server (SLES) 11 SP1
- *Ubuntu* 10.04 LTS (lucid)
- *Ubuntu* 10.10 (maverick)
- *Ubuntu* 11.10 (oneiric)
- *Ubuntu* 12.04 LTS (precise)

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

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

2.4 Specifications for Rudder Root Server

2.4.1 Hardware specifications

A dedicated server is strongly recommended.

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

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

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

2.4.2 Supported Operating Systems

The following operating systems are supported as a Root server:

- *Debian* GNU/Linux 5 (*Lenny*)
- *Debian* GNU/Linux 6 (*Squeeze*)
- *Debian* GNU/Linux 7 (Wheezy)
- Red Hat Enterprise Linux (RHEL) / *CentOS* 6
- *SuSE* Linux Enterprise Server (SLES) 11 SP1
- *Ubuntu* server 11.10 (Oneiric)
- *Ubuntu* server 12.04 LTS (Precise)

2.4.3 Packages

Rudder components are distributed as a set of packages.

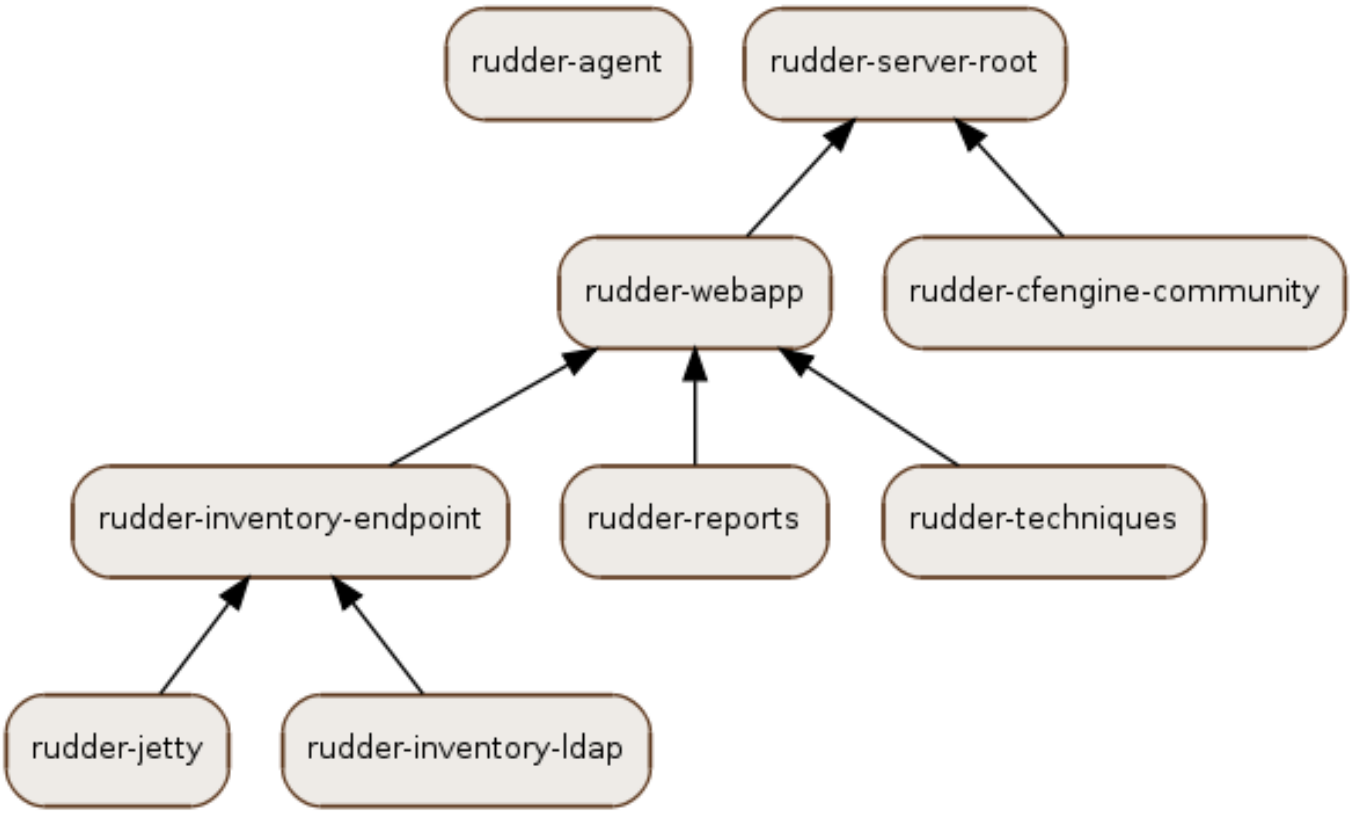

Figure 2.2: Rudder packages and their dependancies

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

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

rudder-jetty Application server for `rudder-webapp` and `rudder-inventory-endpoint`. Both packages are written in *Scala*. At compilation time, they are converted into `.war` files. They need to be run in an application server. *Jetty* is this application server. It depends on a compatible *Java* Runtime Environment. It can be either Oracle Java *JRE* or *OpenJDK 7 JRE*.

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

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

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

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

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

- installs the *Rudder* configuration script:

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

- installs the initial promises for the Root Server in:

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

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

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

- installs the logrotate configuration:

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

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

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

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

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

2.4.4 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 is used to store logs sent by the *Nodes* and reports generated by *Rudder*.

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

***Oracle Java* JRE or OpenJDK 7 JRE** The *Java* runtime is needed by the Jetty application server. On *Debian*, the package from the distribution is used. On SLES, the package must be dowloaded from *Oracle* website.

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

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

2.5 Configure the network

2.5.1 Mandatory flows

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

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

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

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

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

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

2.5.2 Optional flows

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

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

2.5.3 DNS - Name resolution

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

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

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

Chapter 3

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

/var/lib/pgsql Or wherever is located your postgresql database. Can grow by several GB per day.

/var/rudder Contains most of your server information, *LDAP* database, etc.. Slower growth over time.

/var/log/rudder Reports logs can easily grow to 1.5GB per day.

3.1 Install Rudder Root server on Debian or Ubuntu

3.1.1 Update the system

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

Specifically for *Debian* 5 (*Lenny*), since the release of *Debian* 6 (*Squeeze*), the signing key of packages repositories has changed. If you haven't already done it, you should also force the upgrade of the `debian-archive-keyring` package to fetch the new key:

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

3.1.2 Add the Rudder packages repository

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

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

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

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

Then add the *Rudder* repository, by typing:

```
root@rudder-server:~# echo "deb http://www.rudder-project.org/apt-2.4/ $(lsb_release -cs)  ↩
    main" > /etc/apt/sources.list.d/rudder.list
```

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

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

3.1.3 Java on Debian/Ubuntu

The *Rudder Root* server needs a compatible Java *Runtime Environment* to run. In most cases, this will be installed automatically thanks to packaging dependencies, however in some cases manual installation is required.

On Debian Squeeze *(6)* and Debian Lenny *(5)*, the available package is Oracle Java *6 JRE*, namely `sun-java-6-jre`, which is in the *non-free* component. You must make sure this is enabled in your apt sources. Check that `/etc/apt/sources.list` contains the following lines:

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

Tip
Your mirror may differ, `ftp.fr.debian.org` is only an example. Also, please adapt the distribution name if needed (`squeeze` could be replaced by `lenny`).

On Ubuntu *Natty (11.04)* and previous *Ubuntu* versions, you will have to install *Java* yourself as the packaging of the *Oracle* JVM is now restricted by *Oracle*™ and *Rudder* is not compatible with OpenJDK 6, which is the only available JDK from *Ubuntu*. See http://www.java.com/fr/download/ to get *Oracle*'s JVM.

On Debian *Wheezy (7)* and above and Ubuntu *Oneiric (11.10)* and above, the available package is *OpenJDK 7 JRE*, namely `openjdk-7-jre`. It will be installed automatically as a dependency of the *Rudder* packages, and does not require the non-free component.

3.1.4 Install your Rudder Root Server

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

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

Note
If Oracle Java *6 JRE* is installed (usually on *Debian Lenny* (5) or *Squeeze* (6) only), you will be asked to accept the license of the product during installation.

3.1.5 Compatibility with RHEL/CentOS 5 and syslogd

Warning

For users running the *Rudder* **server** on *Ubuntu* Server 12.04 or later, any nodes running syslogd (not syslog-ng or rsyslog) will fail to send any reports about the configuration rules they have applied. This is the case by default on RHEL/*CentOS* 5, but not on any other supported platforms.

Rudder will apply rules on nodes but will never get reports from those using syslogd. Therefore *Rudder* will not be able to calculate compliance.

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.

3.2 Install Rudder Root server on SLES

3.2.1 Configure the package manager

Ensure that the `zypper` package manager is configured, and install the required packages: `rsyslog`, `rsyslog-pgsql` and Oracle Java *6 JRE* or *OpenJDK 7 JRE*. `rsyslog` and `rsyslog-pgsql` are downloadable along *Rudder* and *Java* is available through *Oracle*'s website: http://www.java.com.

3.2.2 Update the system

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

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

3.2.3 Add the Rudder packages repository

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

```
root@rudder-server:~# zypper ar -n "Normation RPM Repositories" \
http://www.rudder-project.org/rpm-2.4/SLES_11_SP1/ Normation
```

Or this one on a SLES 10:

```
root@rudder-server:~# zypper sa "http://www.rudder-project.org/rpm-2.4/SLES_10_SP3/"  ↩
    Normation
```

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

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

3.2.4 Install your Rudder Root Server

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

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

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

3.3 Install Rudder Root server on RedHat or CentOS

3.3.1 Java on RHEL/CentOS

The *Rudder Root* server needs a compatible Java *Runtime Environment* to run.

On RHEL/*CentOS* 6, the available package compatible with *Rudder* server is `java-1.7.0-openjdk` but *Rudder* is also compatible with *Oracle* JRE 1.6 or later.

Oracle JRE 1.6, *Oracle* JRE 1.7 and OpenJDK 1.6 aren't provided by the same virtual package on RHEL/*CentOS* 6 than OpenJDK 1.7. Besides, only OpenJDK 1.7 is provided by default on RHEL/*CentOS* contrary to *Oracle* JRE.

This is why even if *Rudder Server* would work with *Oracle* JRE 1.6 or 1.7, the dependencies will not be resolved with them.

3.3.2 Add the Rudder packages repository

Configure the yum repository for *RedHat*/*CentOS* 6:

```
$ echo "[Rudder_2.4]
name=Rudder 2.4 Repository
baseurl=http://www.rudder-project.org/rpm-2.4/RHEL_6/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-2.4/RHEL_6/repodata/repomd.xml.key
" > /etc/yum.repos.d/rudder.repo
```

Or for *RedHat*/*CentOS* 5:

```
$ echo "[Rudder_2.4]
name=Rudder 2.4 Repository
baseurl=http://www.rudder-project.org/rpm-2.4/RHEL_5/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-2.4/RHEL_5/repodata/repomd.xml.key
" > /etc/yum.repos.d/rudder.repo
```

3.3.3 Install your Rudder Root Server

Install the package:

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

Warning
Rudder don't support SELinux yet (see http://www.rudder-project.org/redmine/issues/2882), then you should set it as permissive with this command:

```
setenforce 0
```

3.4 Initial configuration of your Rudder Root Server

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

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

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

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

Allowed networks A list of IP networks authorized to connect to the server. We recommend that you specify all the networks of your infrastructure. The syntax is the standard network/mask notation, for instance `192.168.0.0/24` or `10.0.0.-0/8`. To add several networks, first type the first network, then press the return key - the script will ask if you wish to add some more networks.

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

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

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

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

3.5 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, then a login screen. Only two demo accounts are configured, without any right restriction as of now.

Files installed by the application

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

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

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

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

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

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

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

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

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

Chapter 4

Install Rudder Agent

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

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

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 <<*Nodes*_supported_OS, the list of supported Operating Systems for the *Nodes*>>.

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

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

Add *Rudder* project repository:

- on *Debian Squeeze*:

```
sudo tee /etc/apt/sources.list.d/rudder.list <<EOF
deb http://www.rudder-project.org/apt-2.4/ $(lsb_release -cs) main contrib non-free
EOF
```

- on *Ubuntu* 11.10 and following, or *Debian* wheezy and following:

```
sudo apt-add-repository http://www.rudder-project.org/apt-2.4/
```

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

```
sudo aptitude update
```

Install the `rudder-agent` package:

```
sudo aptitude install rudder-agent
```

4.2 Install Rudder Agent on RedHat or CentOS

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

```
http://www.rudder-project.org/rpm-2.4/RHEL_5/
http://www.rudder-project.org/rpm-2.4/RHEL_6/
```

Or you can define a yum repository for *RedHat*/*CentOS* 6:

```
$ echo "[Rudder_2.4]
name=Rudder 2.4 Repository
baseurl=http://www.rudder-project.org/rpm-2.4/RHEL_6/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-2.4/RHEL_6/repodata/repomd.xml.key
" > /etc/yum.repos.d/rudder.repo
```

Or for *RedHat*/*CentOS* 5:

```
$ echo "[Rudder_2.4]
name=Rudder 2.4 Repository
baseurl=http://www.rudder-project.org/rpm-2.4/RHEL_5/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-2.4/RHEL_5/repodata/repomd.xml.key
" > /etc/yum.repos.d/rudder.repo
```

Install the package:

```
rpm -Uhv rudder-agent-2.4.0-1.EL.5.x86_64.rpm
```

Or if a yum repository has been set:

```
yum install rudder-agent
```

4.3 Install Rudder Agent on SLES

Following commands are executed as the `root` user.

Add the *Rudder* packages repository:

- on a SLES 11 node:

```
zypper ar -n "Rudder RPM Repositories" \
http://www.rudder-project.org/rpm-2.4/SLES_11_SP1/ Rudder
```

- on a SLES 10 node:

```
zypper sa "http://www.rudder-project.org/rpm-2.4/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
```

4.4 Configure and validate

4.4.1 Configure Rudder Agent

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

```
sudo tee /var/rudder/cfengine-community/policy_server.dat <<EOF
@@replace_by_rudder_server_ip@@
EOF
```

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

4.4.2 Start Rudder Agent:

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

4.4.3 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*.

4.4.3.1 Force Rudder Agent execution

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

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

Chapter 5

Upgrade Rudder

This short chapter covers the upgrade of the *Rudder Server* Root and *Rudder* Agent from a version 2.3 to the latest version 2.4.

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.

5.1 Upgrade Rudder on Debian or Ubuntu

Following commands are executed as the `root` user.

Add *Rudder* project repository:

- on *Debian Squeeze* and followings or *Ubuntu* 11.10 and followings:

```
echo "deb http://www.rudder-project.org/apt-2.4/ $(lsb_release -cs) main contrib non-free"  ↩
    > /etc/apt/sources.list.d/rudder.list
```

- or on *Ubuntu* 11.10 and following, or *Debian* wheezy and following:

```
apt-add-repository http://www.rudder-project.org/apt-2.4/
```

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

- With aptitude:

```
aptitude update
```

- With apt-get:

```
apt-get update
```

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

- With aptitude:

```
aptitude install rudder-server-root
```

- With apt-get:

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

and after the upgrade of these packages, restart jetty to be sure that the changes are applied:

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

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

- With aptitude:

```
aptitude install rudder-agent
```

- With apt-get:

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

Warning
Rudder include a script for upgrading all the files which needed to. Then, you should not replace your old files by the new ones when apt-get/aptitude is asking for, unless you want to reset all your parameters.

5.2 Upgrade Rudder on RedHat or CentOS

Following commands are executed as the `root` user.

Define a yum repository for *RedHat*/*CentOS* 6:

```
$ echo "[Rudder_2.4]
name=Rudder 2.4 Repository
baseurl=http://www.rudder-project.org/rpm-2.4/RHEL_6/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-2.4/RHEL_6/repodata/repomd.xml.key
" > /etc/yum.repos.d/rudder.repo
```

Or for *RedHat*/*CentOS* 5:

```
$ echo "[Rudder_2.4]
name=Rudder 2.4 Repository
baseurl=http://www.rudder-project.org/rpm-2.4/RHEL_5/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-2.4/RHEL_5/repodata/repomd.xml.key
" > /etc/yum.repos.d/rudder.repo
```

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

```
yum update rudder-agent
```

There was no *Rudder Server* packages for version 2.3.

5.3 Upgrade Rudder on SLES

Following commands are executed as the `root` user.

Add the *Rudder* packages repository:

- With zypper on a SLES 11 system:

```
zypper ar -n "Rudder RPM Repositories" \
http://www.rudder-project.org/rpm-2.4/SLES_11_SP1/ Rudder
```

- With zypper on a SLES 10 system:

```
zypper sa "http://www.rudder-project.org/rpm-2.4/SLES_10_SP3/" Rudder
```

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

```
zypper ref
```

For *Rudder Server* (only SLES 11), upgrade all the packages associated to `rudder-server-root`:

```
zypper update rudder*
```

and after the upgrade of these packages, restart jetty to be sure that the changes are applied:

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

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

```
zypper update rudder-agent
```

5.4 Caution cases

5.4.1 Known bugs

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

Chapter 6

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.

6.1 Authentication

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

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

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

6.2 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.

6.2.1 Rudder Home

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

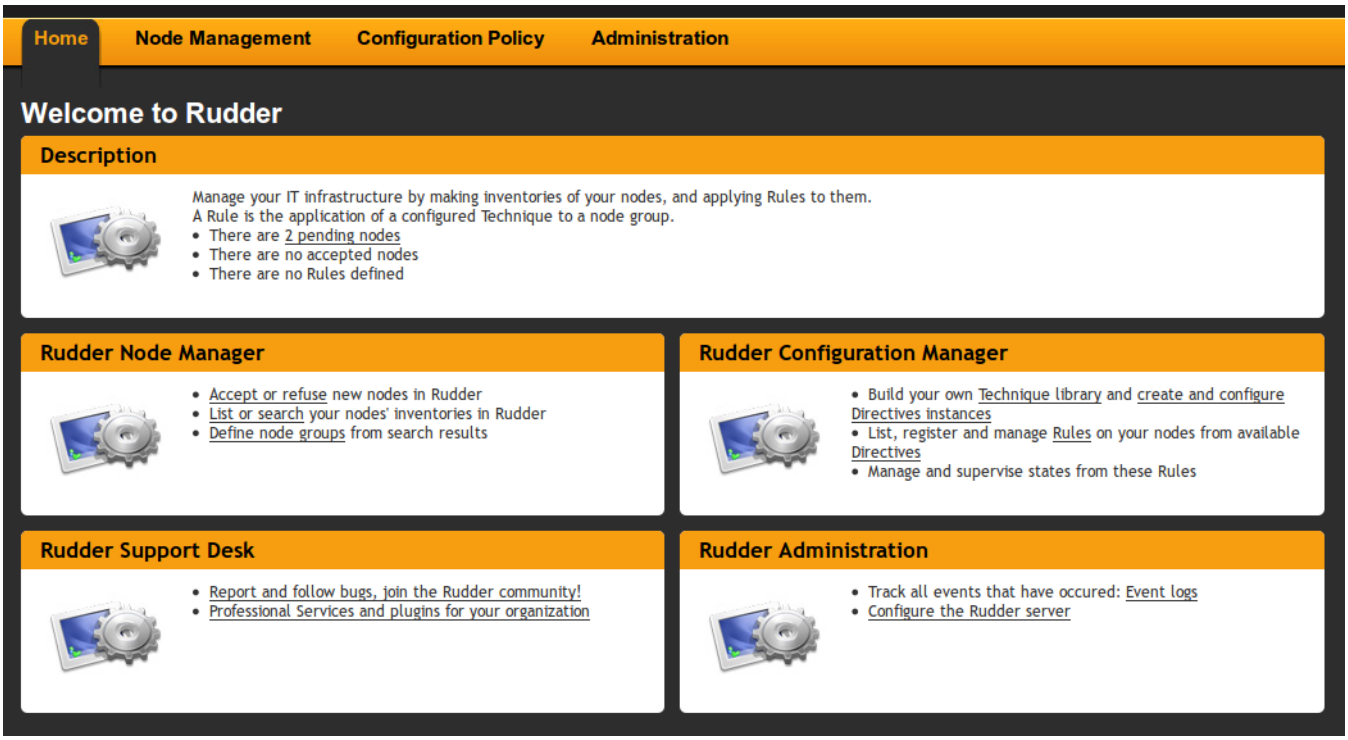

Figure 6.1: Rudder Homepage

6.2.2 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*.

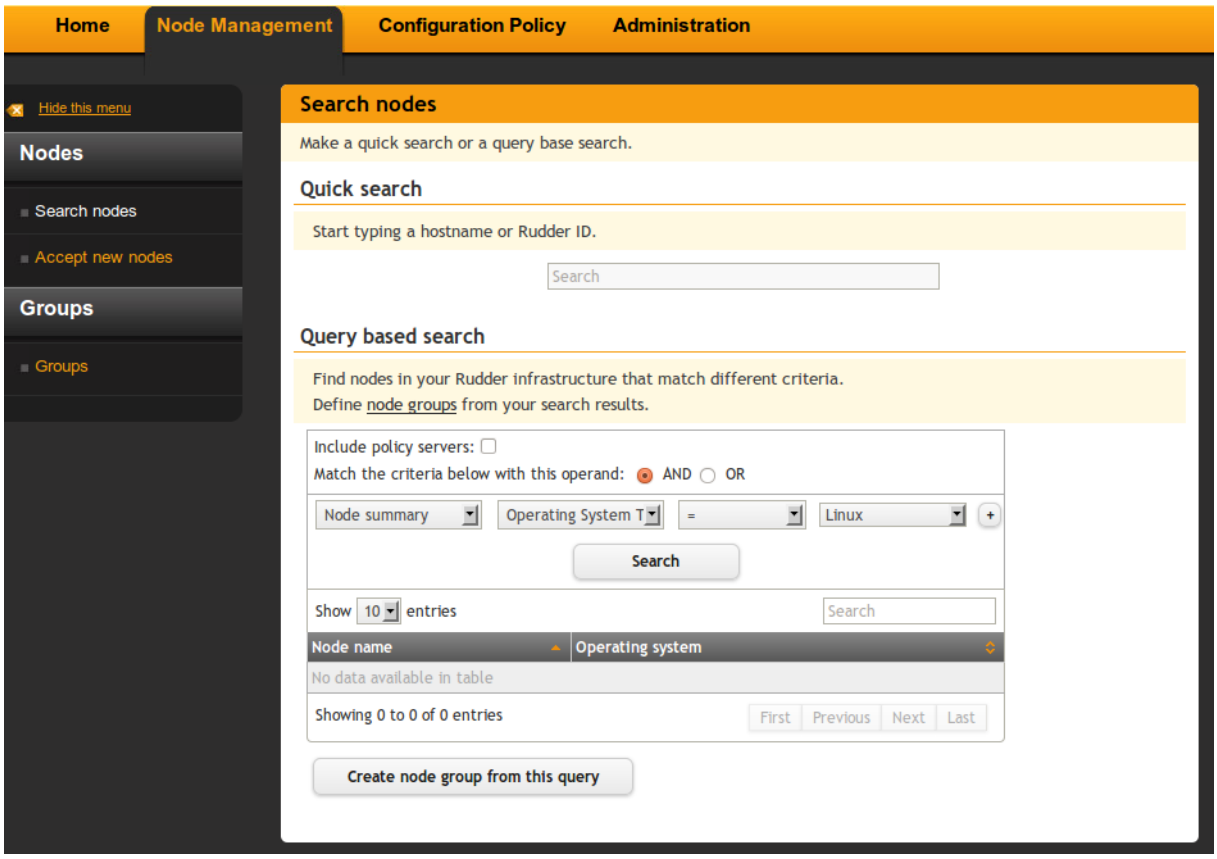

Figure 6.2: Node Management welcome screen

6.2.3 Configuration Management

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

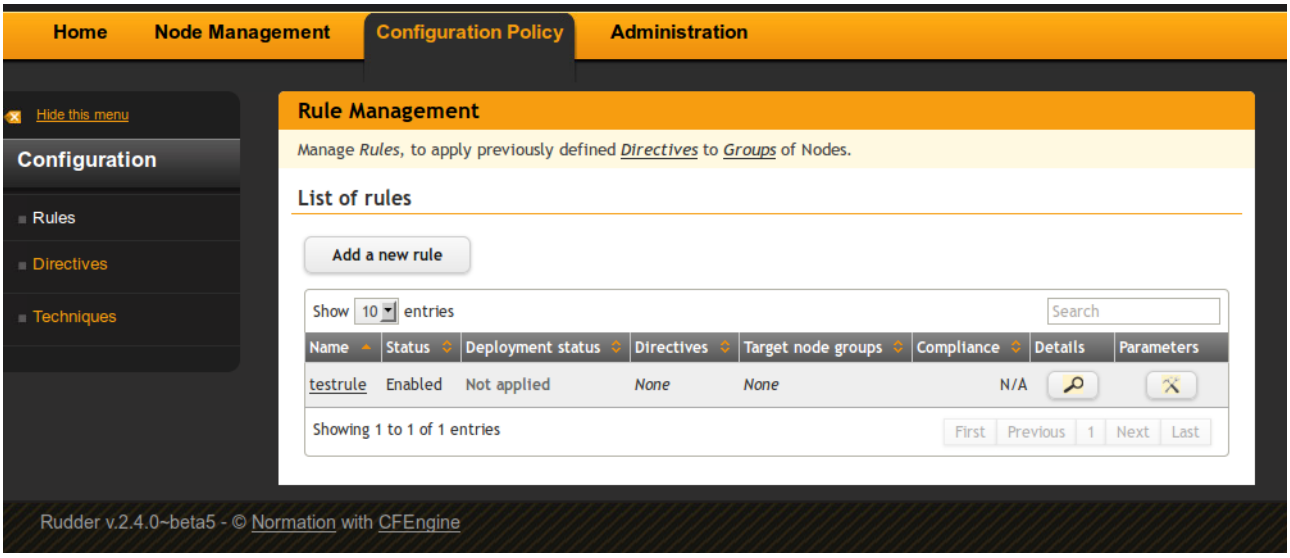

Figure 6.3: Configuration Management welcome screen

6.2.4 Administration

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

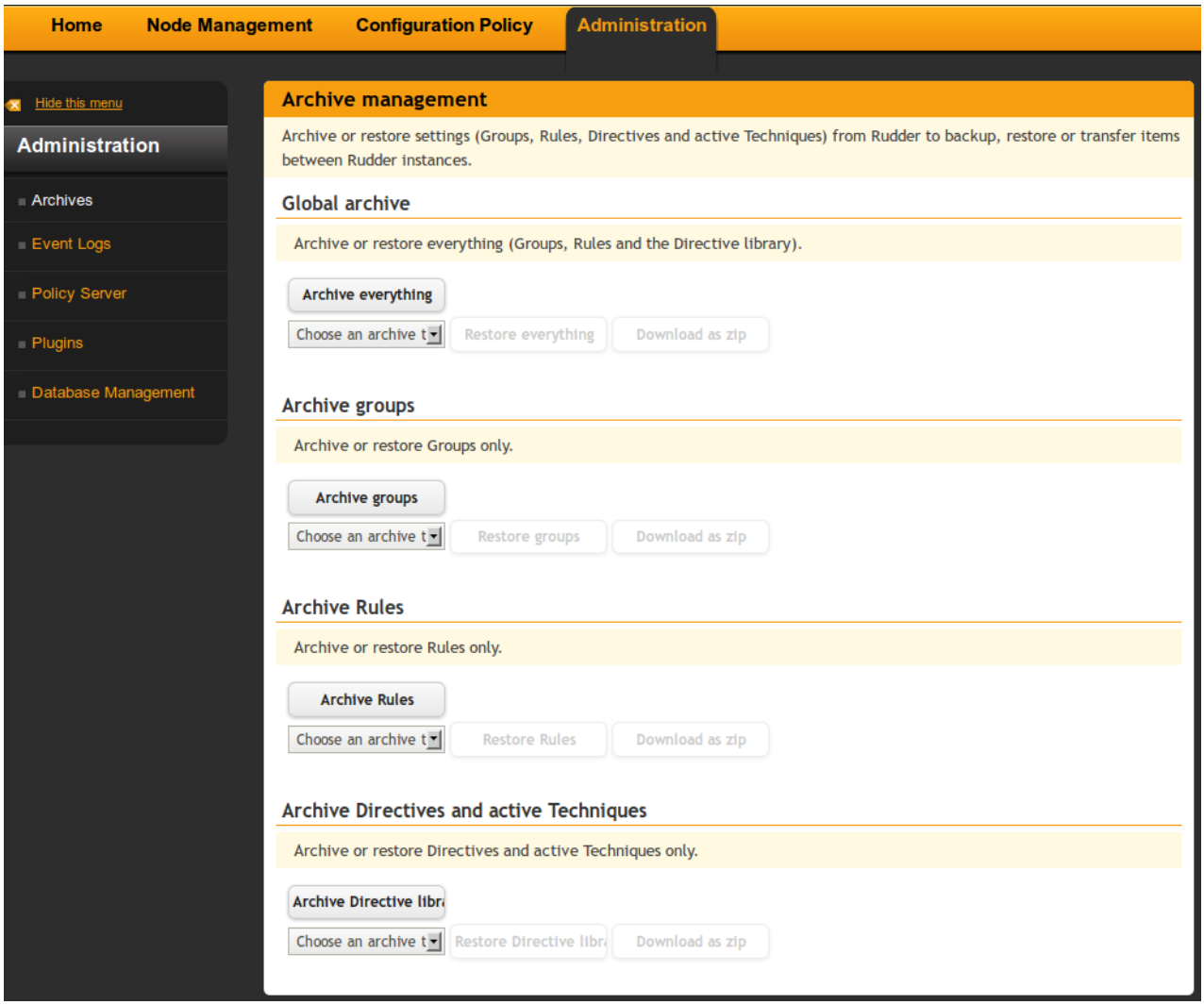

Figure 6.4: Administration welcome screen

6.3 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.

6.3.1 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.

6.3.2 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
```

6.3.3 Supported units

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

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

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

Table 6.1: Units supported by Rudder search engine

Chapter 7

Node Management

7.1 Node Inventory

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

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

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

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

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

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

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

7.2 Accept new Nodes

At the starting point, the *Rudder Server* does'nt know anything about the *Nodes*. After the installation of the *Rudder* Agent, each *Node* register itself to the *Rudder Server*, and sends a first inventory. Every new *Node* must be manually validated in the *Rudder* Web Interface to become part of *Rudder* Managed *Nodes*. This task is performed in the ***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 7.1 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.

7.3 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*.

7.3.1 Quick Search

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

Example 7.2 Quick search the Node called `debian-node`

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

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

7.3.2 Advanced Search

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

- 1. Select a field

The selection of the field upon which the criteria will apply is a two step process. The list of fields is not displayed unordered and extensively. Fields have been grouped in the same way they are displayed when you look at information about a *Node*. First you choose among these groups: Node, *Network Interface*, *Filesystem*, *Machine*, *RAM*, *Storage*, *BIOS*, *Controller*, *Port*, *Processor*, *Sound Card*, *Video Card*, *Software*, *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 nodes 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 7.3 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 informations 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 7.4 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 7.5 Advanced search for Linux Nodes with `ssh`.

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

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

7.4 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 7.6 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 it's 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 8

Configuration Management

8.1 Techniques

8.1.1 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.

8.1.2 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 every available *Techniques*. This list can't be modified: every changes made by an 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* Libraryy. *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.

8.1.3 Available Techniques

8.1.3.1 Application management

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

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

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

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

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

8.1.3.2 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.

8.1.3.3 File state configuration

Set the permissions of files Set the permissions of files

8.1.3.4 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.

8.1.3.5 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

8.1.3.6 System settings: Process

Process Management Enforce defined parameters on system processes

8.1.3.7 System settings: Remote access

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

8.1.3.8 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.

8.2 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 an 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 8.1 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`.

8.3 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*)

8.4 Compliance

A *Directive* contains one or multiple components. Each component generates one ore multiple reports, based on the number of keys in this component. For example, for a Sudoers *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 answer The system didn't sent any reports. *Rudder* waited for 10 minutes and no report was received.

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

Based on these facts, the compliance of a *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

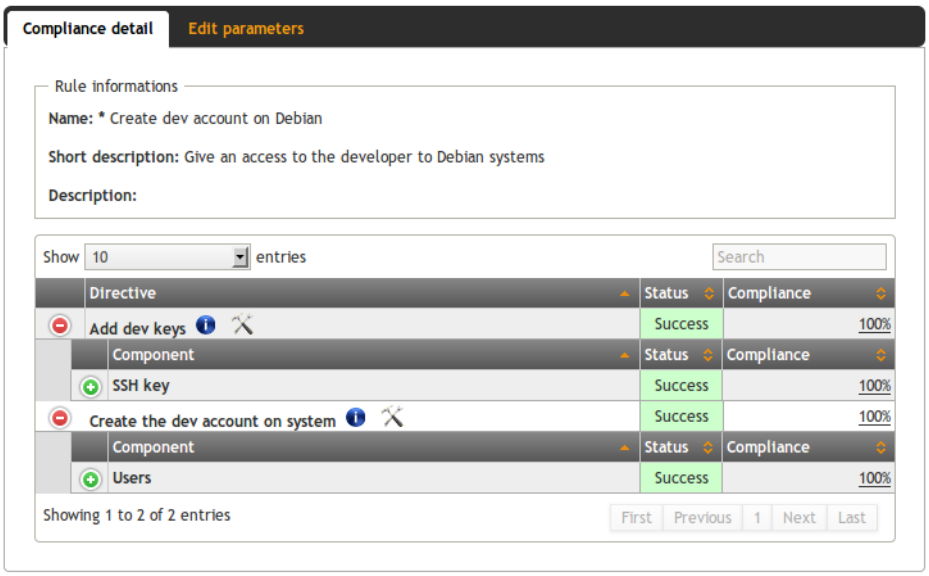

Figure 8.1: Reports

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

In the *Admnistration* > *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*.

Active *Rudder* configuration is stored in a *LDAP* tree.

The content of this tree can be exported into a file tree containing `xml` files, into `/var/rudder/configuration-repository`. This file tree is under version control, using *git*. At exportation time, a *git tag* is created in this repository, and referenced in the *Rudder* Webapp. Each change in the *Rudder* web interface is also commited in the repository.

The content of this repository can be imported into *Rudder*.

9.1.1 Archive usecases

The archive feature of *Rudder* allows to:

- Exchange configuration between multiple *Rudder* instances;
- 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 developement, qualification and production environment. You can prepare the changes on the developement instance, export an archive, deploy this archive on the qualification environment, then on the production environment.

Tip
Use *git* to copy the files from an environment to another.
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.

9.1.1.3 Deploy a preconfigured instance

Assuming you are preparing a complete Policy integration for a client.

1. In your labs:
   a. Prepare the configuration for Groups, *Directives* and *Rules*;
   b. Export the Policy
   c. Create an archive containing the content of the configuration repository (`zip` file).
2. At the client place:
   a. Unpack the archive in `/var/rudder/configuration-repository`
   b. +git commit -a
   c. Restore the configurations from the last commit

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* Agents
- Policy *Node*
- Archives

9.3 Policy Server

The **Administartion > 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 network as you want, the expected format is: `networkip/mask`, for example `42.42.0.0/16`.

9.3.2 Clear caches

Clear cached datas, 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:

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

Tip

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

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:

```
/etc/init.d/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.

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

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

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

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

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

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

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

SQL server Store the received reports from the nodes.

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

9.6 Technique upgrade

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

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

When you upgrade the *Rudder Techniques* packages to a new version, a new version of the *Technique* library is copied to /opt/rudder/share/techniques.

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

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

- Jump to the *Rudder Technique* tree

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

- Copy the reference *Technique* library to your local tree

```
cp -a /opt/rudder/share/techniques/* .
```

- Update the GIT repository to match the new tree state

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

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

9.7 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.sh, 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 correclty 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
```

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.

10.4 About Technique upgrades

10.4.1 Initial installation

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

10.4.2 Upgrade

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 make.

10.4.2.1 Upgrading the Technique library

```
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 conflicts is noticeables, 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 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.

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

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

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

11.1.2 Installation of the Rudder Agent

11.1.2.1 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*.

11.1.2.2 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.

11.1.2.3 Services

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

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

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

11.1.2.4 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
```

11.1.3 Rudder Agent interactive

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

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

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

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

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

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

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

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

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

Restart the *CFEngine* service:

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

11.1.4 Processing new inventories on the server

11.1.4.1 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/
```

11.1.4.2 Process incoming inventories

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

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

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

11.1.4.3 Validate new Nodes

User interaction is required to validate new *Nodes*.

11.1.4.4 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.

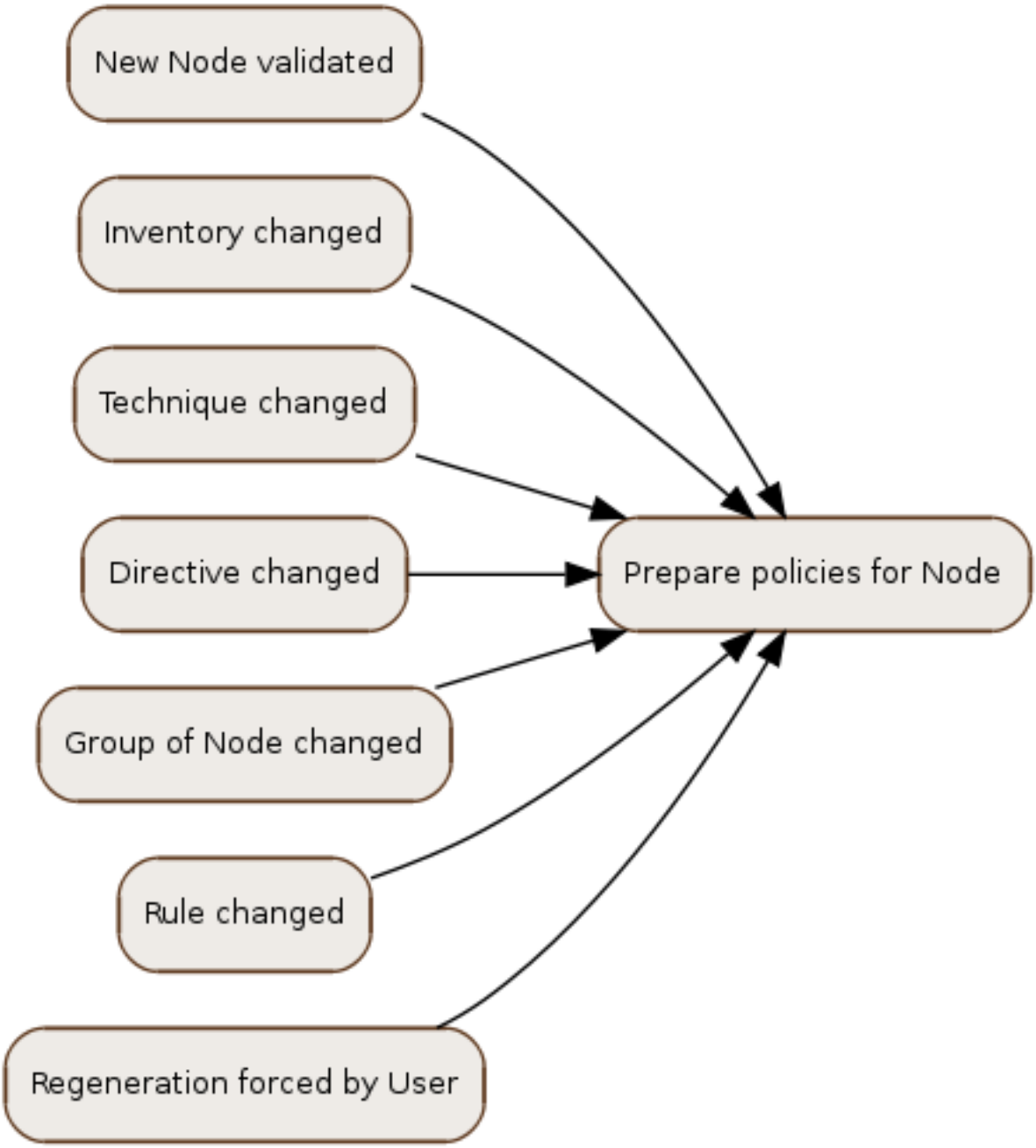

Figure 11.1: Generate policy workflow

11.2 User management

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

11.2.1 Configuration of the users using a XML file

11.2.1.1 Generality and uses of clear text password

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

```
<authentication>
  <user name="jon.doe" password="secret" role="administator"/>
  <user name="alex.bar" password="secret2" role="administation_only, node_read"/>
  <user name="custom" password="custom" 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 ommited but the user will have no permission. Only these attributes are recognized.

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

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

11.2.1.2 Use of hashed passwords

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

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

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

Table 11.1: Hashed passwords algorithms list

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

Here is an example of authentication file with hashed password:

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

11.2.2 Authorization management

For every users you can define an access level. Those authorizations allow you to access different pages or to perform different actions.

You can build custom roles with whatever permission you want, but there's some role already defined.

In xml file, the role attribute is a list of permissions/roles, separated by a comma. each add permissions to the user. If one is wrong, or not correctly spelled, the user is set to the lowest rights (NoRights).

11.2.2.1 Pre-defined roles

- 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 eveything 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.

11.2.2.2 Permissions and custom roles

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

A permission is composed of a kind and a type.

A permission kind is the level granted for a certain type. There's 4 kinds of permission : read, write, edit, all. Depending on that value you have access to different pages and action in rudder.

There's 8 type of permission : configuration, rule, directive, technique, node, group, administration, deployment. that value indicates what kind of datas will be displayed and/or could be set/updated by the user, depending of his access level.

11.2.3 Going further

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

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

11.2.4 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, there is a solution for you.

Since *Rudder* uses the SpringSecurity framework, you are able to connect to a wide range of authentication providers.

We will take *LDAP* as an example, however bear in mind that this procedure requires that you make modifications to your application that an update will replace, we do not officially support it yet.

Also 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).

An 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).

If you really want to test this feature, follow this procedure:

First, unzip your webapp to be able to modify some files inside it

```
cd /opt/rudder/jetty7/webapps
mv rudder.war /root
mkdir rudder.war
cd rudder.war && unzip /root/rudder.war
```

Then use your favorite editor to edit /opt/rudder/jetty7/webapps/rudder.war/WEB-INF/classes/applicationContext-security.xml to change <authentication-provider ref="demoAuthenticationProvider"/> to <authentication-provider ref="ldapAuthenticationProvider"/> and paste the following code block below </authentication-manager> :

```
<beans:bean id="contextSource"
        class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
  <beans:constructor-arg value="ldap://ldap.mycorp.com:389/dc=mycorp,dc=com"/>
  <beans:property name="userDn" value="cn=Rudder,ou=AppOU,dc=mycorp,dc=com"/>
  <beans:property name="password" value="myverysecretpassword"/>
</beans:bean>

<beans:bean id="ldapAuthenticationProvider"
    class="org.springframework.security.ldap.authentication.LdapAuthenticationProvider">
 <beans:constructor-arg>
   <beans:bean class="org.springframework.security.ldap.authentication.BindAuthenticator">
     <beans:constructor-arg ref="contextSource"/>
     <beans:property name="userDnPatterns">
       <beans:list><beans:value>uid={0},ou=people</beans:value></beans:list>
     </beans:property>
   </beans:bean>
 </beans:constructor-arg>
 <beans:property name="userDetailsContextMapper" ref="rudderXMLUserDetails"/>
</beans:bean>
```

You will have to adjust the ldap URI, the userDn and password to the appropriate values for your infrastructure.

The precedent exemple use direct Bind DN pattern to authenticate user. A common other solution in *LDAP* is to use a different login attribute than the RDN attribute of the user entry, and so we use a two-steps process:

- first, we search for the unique entry with the given login in a given branch and get its DN,
- then, we try to authenticate the user with the DN and the password.

If you want to use that approach, you will have to change the precedent XML code so that in place of "userDnPatterns", you have the search logic, as it is demonstrated in the next example:

```
<beans:bean id="contextSource"
        class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
  <beans:constructor-arg value="ldap://ldap.mycorp.com:389/dc=mycorp,dc=com"/>
  <beans:property name="userDn" value="cn=Rudder,ou=AppOU,dc=mycorp,dc=com"/>
  <beans:property name="password" value="myverysecretpassword"/>
</beans:bean>


<beans:bean id="userLookup" class="org.springframework.security.ldap.search. ↩
    FilterBasedLdapUserSearch">
  <beans:constructor-arg index="0" value="ou=people"/>
  <beans:constructor-arg index="1" value="(&amp;(uuid={0})(objectclass=user))"/>
  <beans:constructor-arg index="2" ref="contextSource" />
</beans:bean>

<beans:bean id="ldapAuthenticationProvider"
    class="org.springframework.security.ldap.authentication.LdapAuthenticationProvider">
   <beans:constructor-arg>
     <beans:bean class="org.springframework.security.ldap.authentication.BindAuthenticator ↩
         ">
       <beans:constructor-arg ref="contextSource"/>
       <beans:property name="userSearch" ref="userLookup"/>
     </beans:bean>
   </beans:constructor-arg>
 <beans:property name="userDetailsContextMapper" ref="rudderXMLUserDetails"/>
</beans:bean>
```

And you are all done !

Tip
This procedure is still hacky, we are considering several solutions to have a standardized procedure that does not require to touch the WAR archive and just edit some configuration files.

Tip
It is a best practice to use authenticated connection in place of anonymous one, even for application. Nonetheless, if you want to use an anonymous connection to your *LDAP* server, you can replace the two lines about beans:property userDn and password in contextSource by the line:
<beans:property name="anonymousReadOnly" value="true"/>

11.3 Password management

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

11.3.1 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
```

11.3.2 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
```

11.3.3 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)

11.4 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.

11.4.1 `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 usefull after some changes impacting the inventory informations.

11.4.2 Disable automatic regeneration of promises

In certain circumstances, it can be necessary to disable the automatic regeneration of the promises. It can be done by setting following property to false in `/opt/rudder/etc/rudder-web.properties`.

```
rudder.autoDeployOnModification=true
```

When switching to manual deployment of promises, the presence of pending changes is advertised on top of the `Regenerate now` button. Each modification can be reviewed before validation.

11.5 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 functionality or configure new services. This paragraph will walk you through this process.

11.5.1 Prequisites

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!

11.5.2 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*.

11.5.3 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
```

11.5.3.1 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

11.5.3.2 First test in the Rudder interface

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

11.5.4 Implement the behavior

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

11.5.4.1 Read in the variables from Rudder

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

11.5.4.2 Add reporting

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

11.6 REST API

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

11.6.1 Default setup

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

11.6.1.1 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
```

11.6.1.2 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 authentified 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 autentication at apache level.

11.6.1.3 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 authentified, the authentified user name will be used in the logs.

11.6.2 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.

11.6.3 Promises regeneration

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

11.6.4 Dynamic groups regeneration

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

11.6.5 Technique library reload

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

11.6.6 Archives manipulation

Various methods are available to import and export items:

11.6.6.1 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

11.6.6.2 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).

11.6.6.3 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.

11.6.6.4 Restoring the latest available archive (from a previously archive 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
```

11.6.6.5 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
```

11.6.6.6 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 allow 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.

11.7 Server optimization

11.7.1 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/8.x/main/postgresql.conf
```

On a *SuSE* system:

```
/var/lib/pgsql/data/postgresql.conf
```

11.7.1.1 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
```

11.7.1.2 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
```

11.8 Server migration

11.8.1 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.

11.8.2 Handle configuration files

11.8.2.1 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 deployement, your configuration should be restored

11.8.2.2 Use Archive feature of Rudder

You can download an archive of your configuration in the *Rudder* UI and use it on your new server To keep your *Technique* you will have to copy the techniques folder in /var/rudder/configuration-repository to your new server You will have a new git tree, and you will lose all the history you add before and the all the comments

Techniques

- Copy the `/var/rudder/configuration-repository/techniques` folder from your old server to you new one

Others

- On your old server UI, go to **Administration > Archives**

- In global archive, choose the latest commit, and then click on download as Zip
- Copy and extract the archive in `/var/rudder/configuration-repository`

Git

- use `git add rules techniques groups directives`
- use `git commit`
- the latest git commit will now be exactly the same as the old one

Restore

- 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 deployement, your configuration should be restored

11.8.3 Handle CFEngine keys

11.8.3.1 Keep your CFEngine keys

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

11.8.3.2 Change CFEngine keys

On every node that were using your old server rudder, 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

11.8.4 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 `/var/rudder/cfengine-community -KI -D force_inventory`

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

Your configuration is now totally migrated.

11.9 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 (ie, *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 usefull.

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*.

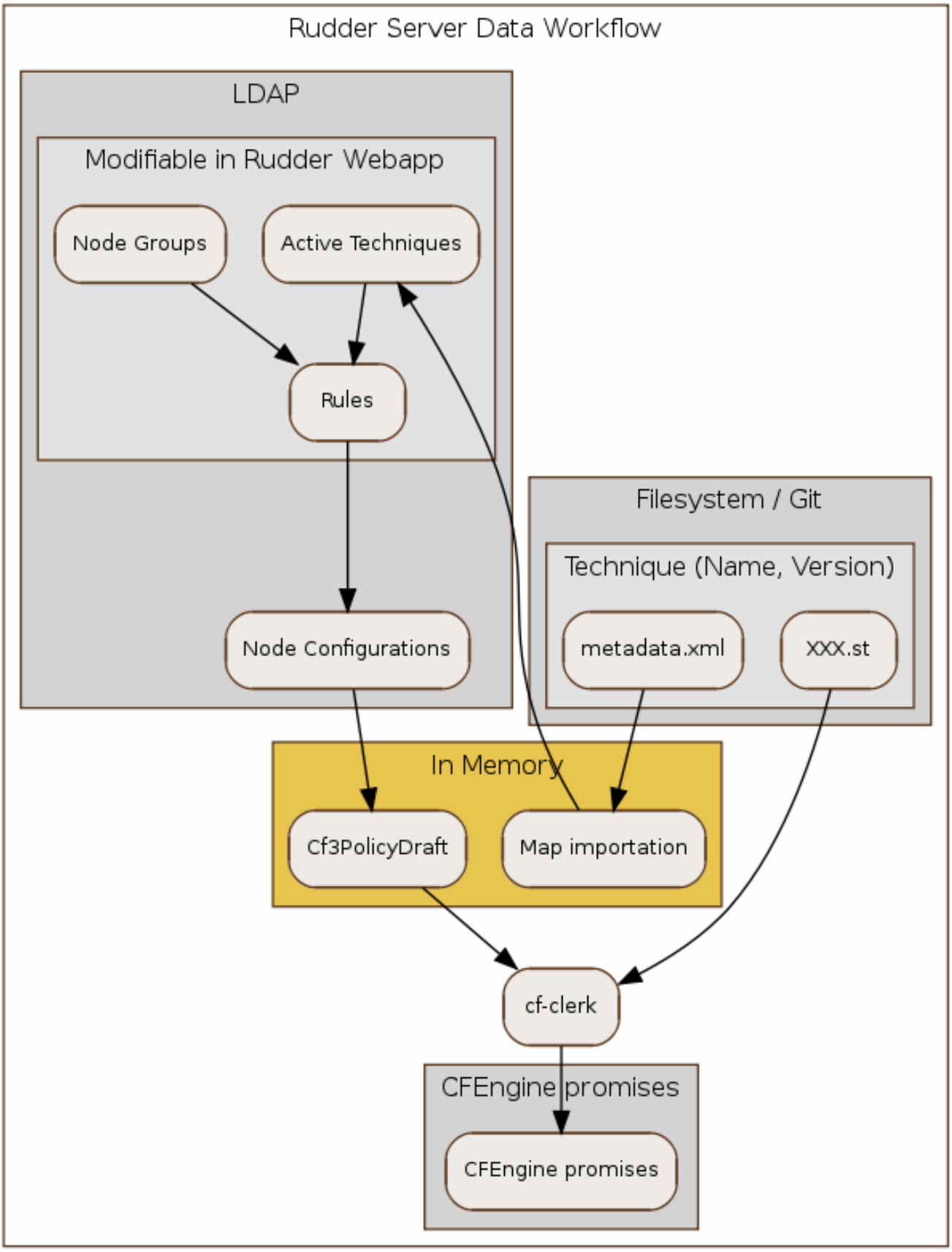

Figure 12.1: Rudder data workflow

12.2 Rudder Agent workflow

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

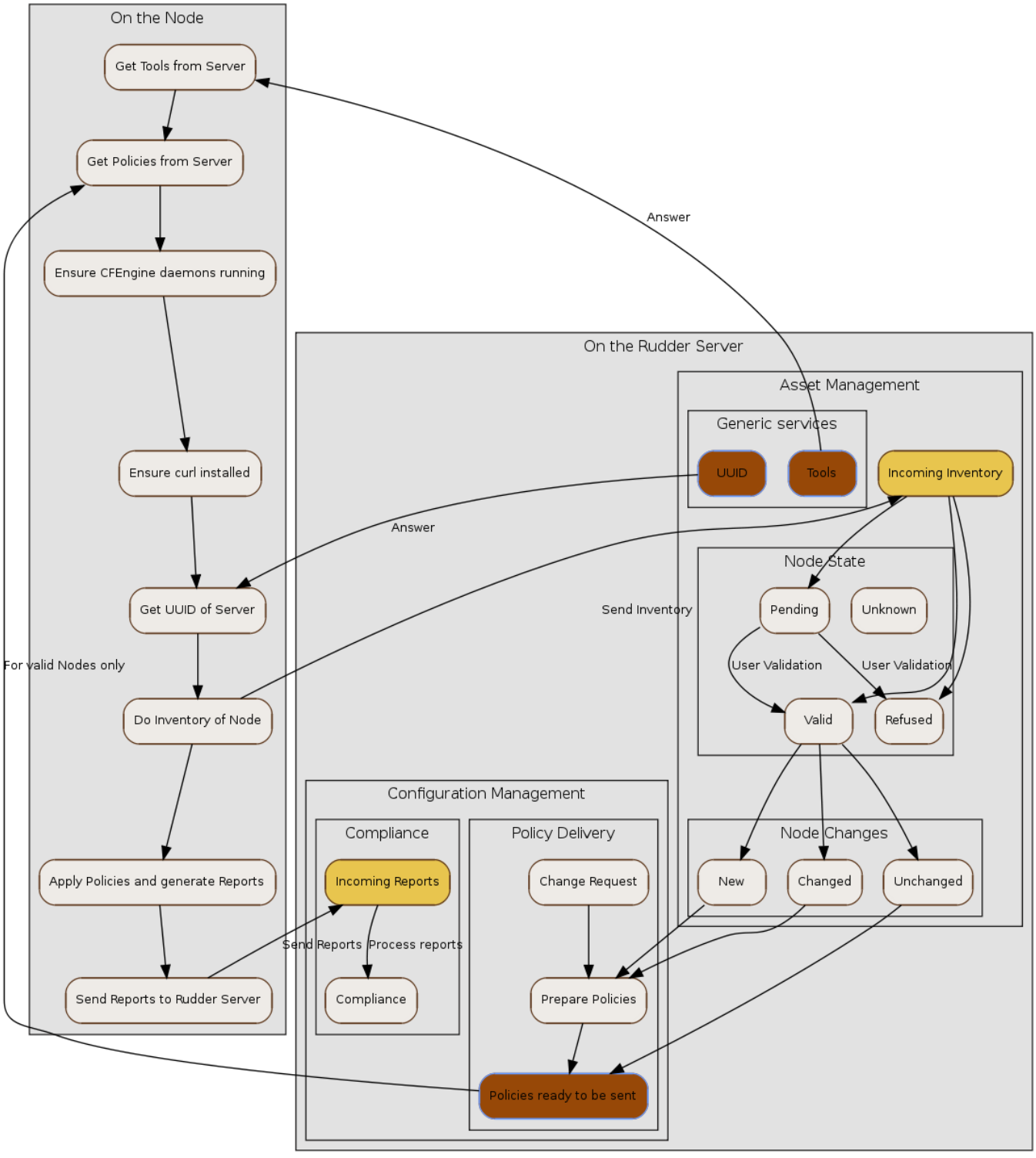

Figure 12.2: Rudder Agent workflow

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

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

12.2.1 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.

12.2.2 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.

12.2.3 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

12.2.4 Inventory

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

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

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

12.2.5 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`.

12.2.6 Apply Directives

Apply other policies and write reports locally.

12.3 Configuration files for a Node

/etc/default/rudder-agent

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

# Cfengine Community directory and files
CFENGINE_COMMUNITY_PATH="/opt/rudder"
```

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

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

12.4 Configuration files for Rudder Server

/opt/rudder/etc/htpasswd-webdav

```
rudder:vHBLbrOyfEWFg
```

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

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

#
## LDAP related configuration
#

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

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

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

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

/opt/rudder/etc/logback.xml

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

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

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

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

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

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

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

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

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

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

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

    This logger allow to trace LDAP writes operation and
    to output them in LDIF file (the output directory path
```

```
    is configured in the main configuration file)
    The trace is done only if level is set to "trace"
    WARNING: setting the level to trace may have major
    performance issue, as A LOT of LDIF files will have
    to be written.
    You should activate that log only for debugging purpose.
  -->

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


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

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

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

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

</configuration>
```

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

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

loglevel none stats

# Define global ACLs to disable default read access.

# Do not enable referrals until AFTER you have a working directory
```

```
# service AND an understanding of referrals.
#referral       ldap://root.openldap.org

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

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

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

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

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

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

database        hdb
suffix          "cn=rudder-configuration"
rootdn          "cn=Manager,cn=rudder-configuration"
# Cleartext passwords, especially for the rootdn, should
# be avoid.  See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw          secret
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory   /var/rudder/ldap/openldap-data
# Checkpoint database every 128k written or every 5 minutes
checkpoint      0       1
```

```
# Indices to maintain
index   objectClass     eq
index   confirmed       eq
index   uuid,machineUuid,nodeId,machine,hostedVm,container,node,software eq
index   mountPoint,softwareVersion,cn   eq
index   member eq

database monitor
```

/opt/rudder/etc/reportsInfo.xml

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

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

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

 For example: <authentication hash="sha">

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

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

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

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

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


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

```
##
#  LDAP properties
##

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

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

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

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

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


##
# Other Rudder Configuration properties
##

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

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

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


##
#  Promise writer directory configuration
##
rudder.dir.config=/opt/rudder/etc/
rudder.dir.policyPackages=/opt/rudder/share/policy-templates
rudder.dir.licensesFolder=/opt/rudder/etc/licenses
rudder.dir.policies=/var/rudder/
rudder.dir.backup=/var/rudder/backup/
rudder.dir.dependencies=/var/rudder/tools/
rudder.dir.sharing=/var/rudder/files/
rudder.dir.lock=/var/rudder/lock/
```

```
rudder.endpoint.cmdb=http://localhost:8080/endpoint/upload/

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


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


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

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


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

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


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

Chapter 13

Handbook

This chapter contains some tips and tricks you might want to know about using *Rudder* in a production environment, with some useful optimizations 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 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.

Howewer, 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 connexion ↩
     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.
```

13.1.2 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
/etc/init.d/slapd stop

# Reindex the databases
/opt/rudder/sbin/slapindex

# Restart OpenLDAP
/etc/init.d/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
/etc/init.d/rudder-root-server 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:
/etc/init.d/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 Application tuning

Some applications used by *Rudder* can be tuned to your needs, like *Apache* HTTPd.

13.3.1 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.

13.3.2 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 /etc/default/jetty, whereas the default ones fit the basic recommandations for the minimal *Rudder* hardware requirements.

- -Xms and Xmx: These parameters 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.

- -XX:PermSize -XX:MaxPermSize: These parameters are 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.

13.3.3 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.

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 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=128 0 0" >> /etc/fstab
mount /var/rudder/cfengine-community/state
```

Chapter 14

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* Libraryy. *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*.

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

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

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

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

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

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

Directive This is an instance of a *Technique*, which allows to set values for the parameters of the latter. Each *Directive* can have an 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 514, TCP Syslog port, used to centralize reports.

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

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

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

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

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

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

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

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

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 every available *Techniques*. This list can't be modified: every changes made by an user will be applied to the Active *Techniques*.

Technique This is a configuration skeleton, adapted to a function or a particular service (eg DNS resolver configuration). This skeleton includes the configuration logic for this function or service, and can be set according to a list of variables (in the same example: IP 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

External contributions:

I like buttons 3a icon set for admonition blocks by MazeNL77 is free for commercial usage.