Rudder 4.2 - User Manual

COLLABORATORS

	TITLE : Rudder 4.2 - User Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jonathan Clarke, Nicolas Charles, Fabrice Flore-Thebault, Matthieu Cerda, Nicolas Perron, Arthur Anglade, Vincent Membré, and François Armand	Sep 2016	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
4.2	Sep 2016		N

Contents

List of Figures

List of Tables

Chapter 1

Online version

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

Chapter 2

Introduction

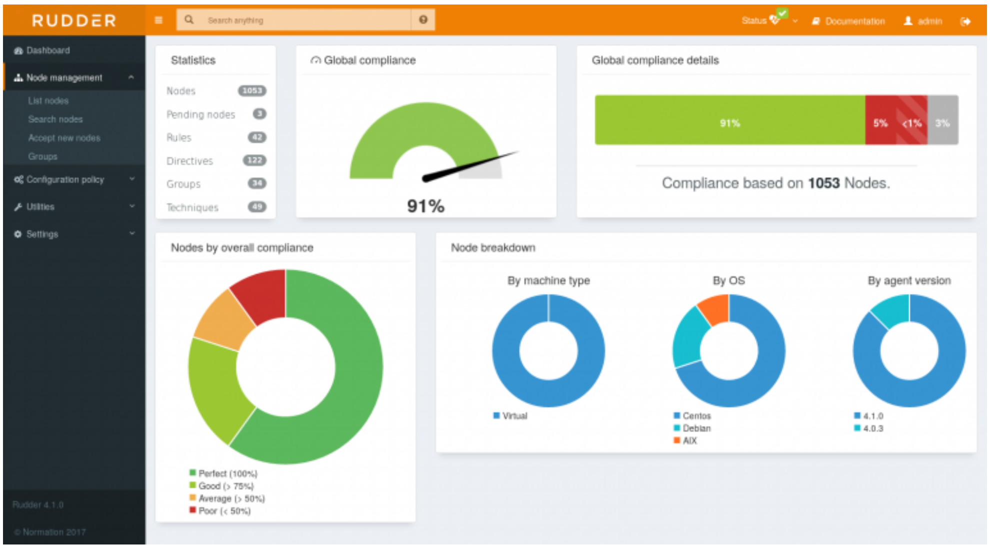
2.1 What is Rudder?

Rudder is an easy to use, web-driven, role-based solution for IT Infrastructure Automation and Compliance. With a focus on continuously checking configurations and centralising real-time status data, *Rudder* can show a high-level summary (*ISO 27001 rules are at 100%!*) and break down noncompliance issues to a deep technical level (*Host prod-web-03: SSH server configuration allows root logins*).

A few things that make *Rudder* stand out:

- A **simple framework** allows you to **extend the built-in rules** to implement specific low-level configuration patterns, however complex they may be, using simple building blocks (*ensure package installed in version X*, *ensure file content*, *ensure line in file*, etc.). A graphical builder lowers the technical level required to use this.
- Each policy can be independently set to be automatically **checked or enforced** on a policy or host level. In Enforce mode, each remediation action is recorded, showing the value of these invisible fixes.
- *Rudder* works on almost **every kind of device**, so you'll be managing physical and virtual servers in the data center, cloud instances, and embedded IoT devices in the same way.
- *Rudder* is designed for **critical environments** where a **security** breach can mean more than a blip in the sales stats. Built-in features include change requests, audit logs, and strong authentication.
- *Rudder* relies on an agent that needs to be installed on all hosts to audit. The **agent is very lightweight** (10 to 20 MB of RAM at peak) and **blazingly fast** (it's written in C and takes less than 10 seconds to verify 100 rules). Installation is self-contained, via a single package, and can auto-update to limit agent management burden.
- *Rudder* is a **true and professional open source** solution—the team behind *Rudder* doesn't believe in the dual-speed licensing approach that makes you reinstall everything and promotes open source as little more than a “demo version.”

Rudder is an established project with **several 10000s of node managed**, in companies from small to biggest-in-their-field. Typical deployments manage 100s to 1000s of nodes. The biggest known deployment in 2017 is about 7000 nodes.



Statistics

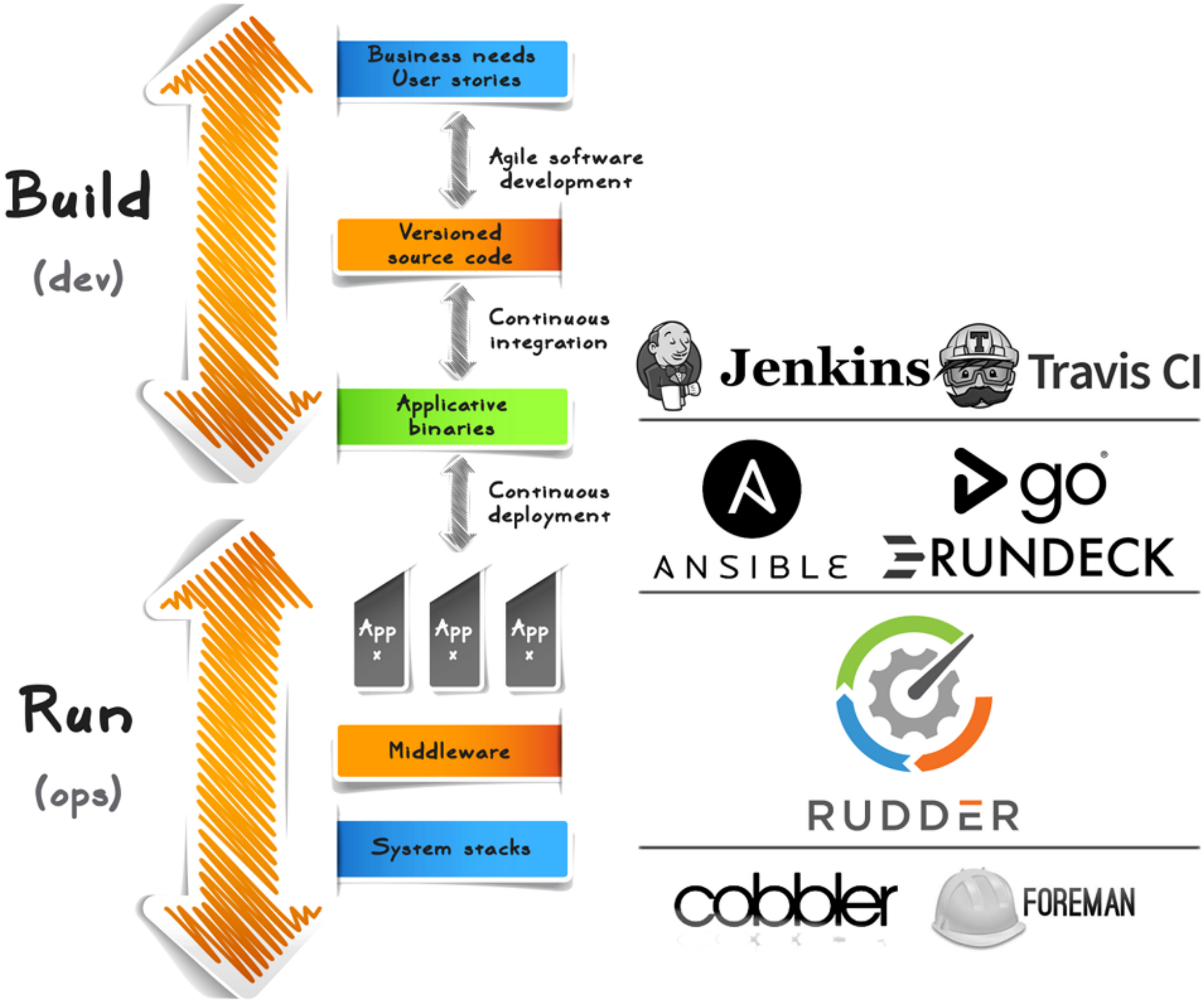
Nodes	1053
Pending nodes	3
Rules	42
Directives	122
Groups	34
Techniques	49

Compliance based on **1053** Nodes.

2.1.1 Made for production environments

We believe that there is a growing impedence mismatch between the Short Time of application development and deployement, and the Long Time of the infrastructure. The latter need rationalisation, stability and conformity before catching the hyped techno of the day, to be able to deliver reliable technical platform, continuously working with a minimum of risks.

Rudder was made for the Long Time, to help team deliver efficient infrastructures with simplicity, giving them feedback where needed, keeping them alert of possible incoming problem, continously checking conformity to their rules, and all of that whatever the infrastructure they choose to build.



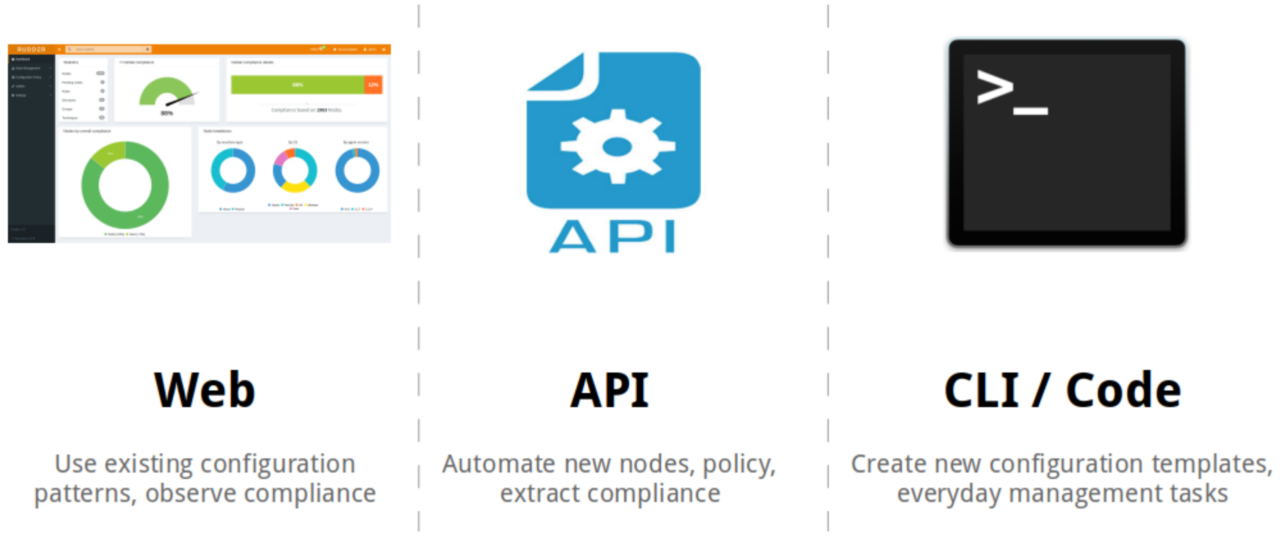
To achieve these goals, *Rudder* goes beyond simple automation of commands or configurations. *Rudder* continuously maintains your infrastructure to keep it conform with your configurations and security rules.

At each level (global, by configuration policy, by node, etc), you can choose to either **Audit** the component - and no modification at all will made on it -, or to **Enforce** the policy, automatically correcting a drift if needed.

2.1.2 Different roles for a better accessibility

Rudder was thought from the start for plug&play-ability: easy to install and to upgrade, easy to start with and growth with.

Rudder comes with a graphical interface, a standard library of configuration policy ready to use, and a graphical rule editor.



Web

Use existing configuration patterns, observe compliance

API

Automate new nodes, policy, extract compliance

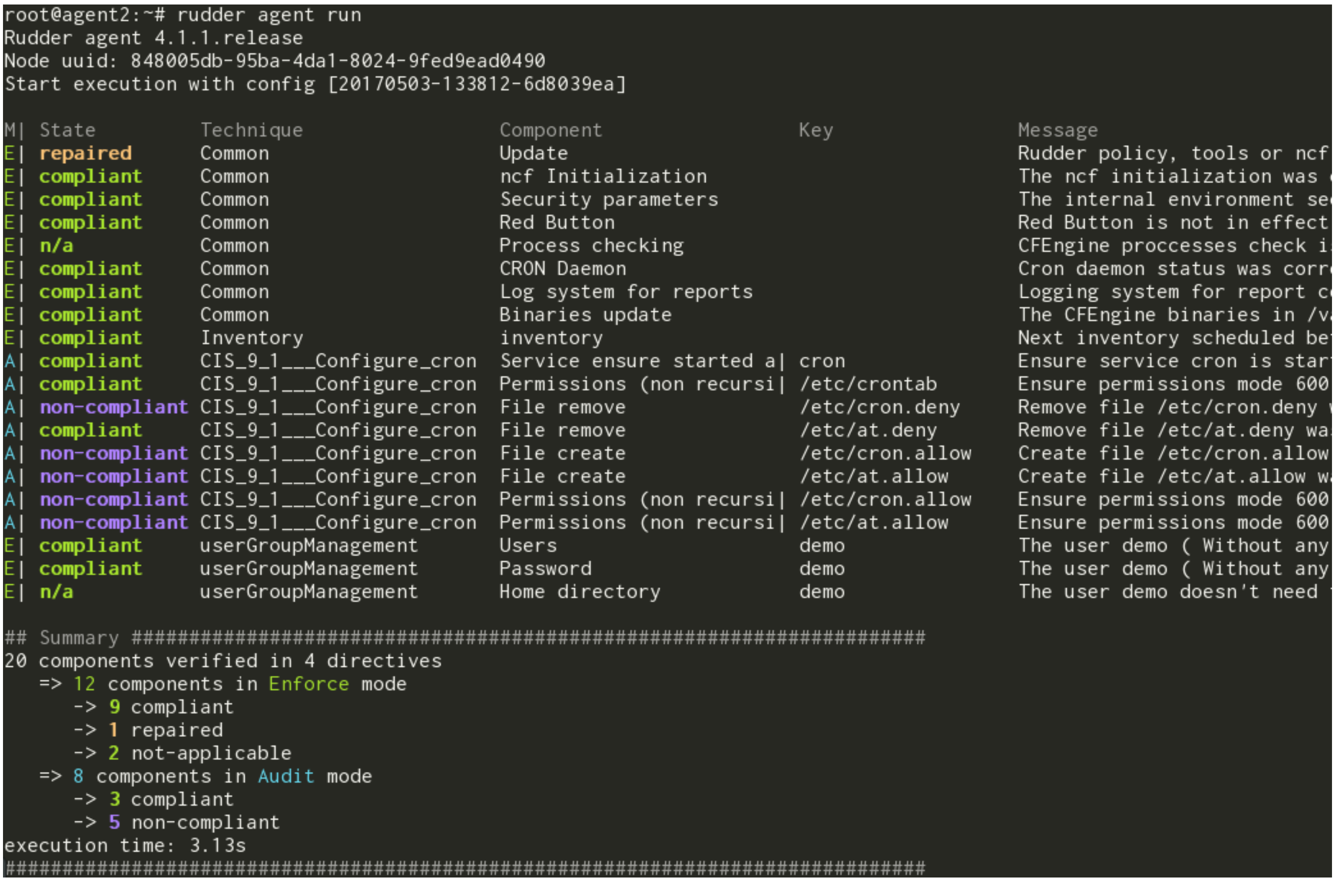
CLI / Code

Create new configuration templates, everyday management tasks

Developers can script *Rudder* through its APIs and security teams can check conformity level to their policies or inventory (both software and hardware) of a server at any time.

2.1.3 Universality

Rudder agent is extremely fast, light, and versatile. It works on a wide variety of OS or hardware, from physical server to cloud instance, user laptops or even Digital Cities and IoT objects.

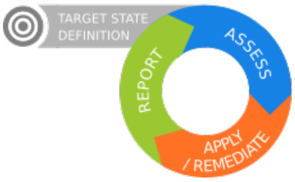


```
root@agent2:~# rudder agent run
Rudder agent 4.1.1.release
Node uuid: 848005db-95ba-4da1-8024-9fed9ead0490
Start execution with config [20170503-133812-6d8039ea]

M| State         Technique                 Component                 Key                Message
E| repaired      Common                    Update                                       Rudder policy, tools or ncf
E| compliant     Common                    ncf Initialization                           The ncf initialization was
E| compliant     Common                    Security parameters                          The internal environment se
E| compliant     Common                    Red Button                                   Red Button is not in effect
E| n/a           Common                    Process checking                             CFEngine proccesses check i
E| compliant     Common                    CRON Daemon                                  Cron daemon status was corr
E| compliant     Common                    Log system for reports                       Logging system for report c
E| compliant     Common                    Binaries update                              The CFEngine binaries in /v
E| compliant     Inventory                 inventory                                    Next inventory scheduled be
A| compliant     CIS_9_1___Configure_cron  Service ensure started a| cron               Ensure service cron is star
A| compliant     CIS_9_1___Configure_cron  Permissions (non recursi| /etc/crontab       Ensure permissions mode 600
A| non-compliant CIS_9_1___Configure_cron  File remove               /etc/cron.deny     Remove file /etc/cron.deny
A| compliant     CIS_9_1___Configure_cron  File remove               /etc/at.deny       Remove file /etc/at.deny wa
A| non-compliant CIS_9_1___Configure_cron  File create               /etc/cron.allow    Create file /etc/cron.allow
A| non-compliant CIS_9_1___Configure_cron  File create               /etc/at.allow      Create file /etc/at.allow w
A| non-compliant CIS_9_1___Configure_cron  Permissions (non recursi| /etc/cron.allow    Ensure permissions mode 600
A| non-compliant CIS_9_1___Configure_cron  Permissions (non recursi| /etc/at.allow      Ensure permissions mode 600
E| compliant     userGroupManagement       Users                     demo               The user demo ( Without any
E| compliant     userGroupManagement       Password                  demo               The user demo ( Without any
E| n/a           userGroupManagement       Home directory            demo               The user demo doesn't need

## Summary #####################################################################
20 components verified in 4 directives
   => 12 components in Enforce mode
      -> 9 compliant
      -> 1 repaired
      -> 2 not-applicable
   => 8 components in Audit mode
      -> 3 compliant
      -> 5 non-compliant
execution time: 3.13s
################################################################################
```

2.2 Key Features

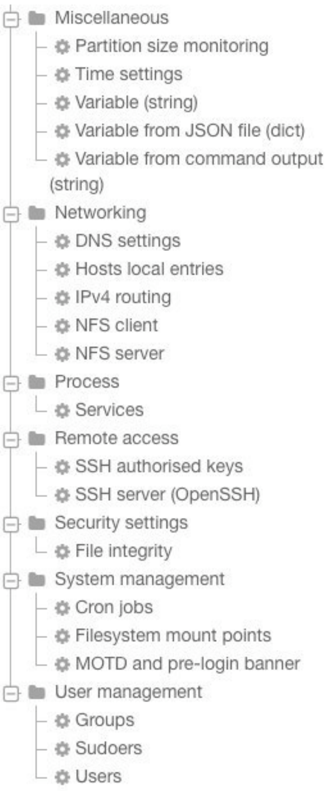
2.2.1 OS independent target configuration state definition

Rudder is able to adapt to complex process and only do the minimal required work so that the server converges to the desired state, and so whatever was the starting state point. *Rudder* works as a GPS would, adapting the path to your destination depending of the path you actually took. This process is much more resilient to changes than a step by step, procedural description of the commands to execute.



Rudder is natively integrated with the supported OS (Linux, *Windows*, AIX - see the list of supported Operating Systems for Nodes) so that it provides generic, abstract, OS independant primitives to the user who can:

- install software in OS native packaging system (RPM on *RHEL*, *Windows* software components, or even direct install from sources),
- configure OS level parameters and services like logs, DNS, NTP, etc.
- create and maintain user accounts (administrator accesses, developers) and groups with a transparent support of OS specific requirements on file format, password hashes algorithms, etc for any supported OS.
- build an hardened system by configuring and then continuously verifying the correct set-up of security rules like file system rights, file integrity checking, etc.
- configure middleware by files (for example in Linux world, whatever the file format, and be it from a template or by only specifying enforcement of some configuration parameters) or thanks to the *Windows* Regestry,
- manage service start-up at boot time and ensure that a service is correctly running at any time, starting it up again if needed.

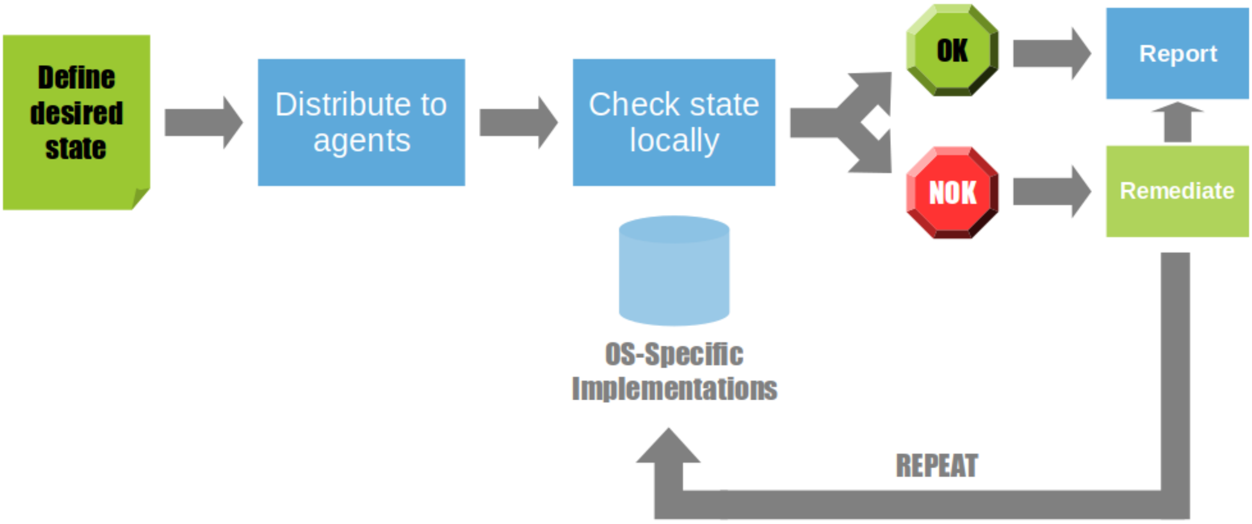


- Miscellaneous
  - Partition size monitoring
  - Time settings
  - Variable (string)
  - Variable from JSON file (dict)
  - Variable from command output (string)
- Networking
  - DNS settings
  - Hosts local entries
  - IPv4 routing
  - NFS client
  - NFS server
- Process
  - Services
- Remote access
  - SSH authorised keys
  - SSH server (OpenSSH)
- Security settings
  - File integrity
- System management
  - Cron jobs
  - Filesystem mount points
  - MOTD and pre-login banner
- User management
  - Groups
  - Sudoers
  - Users

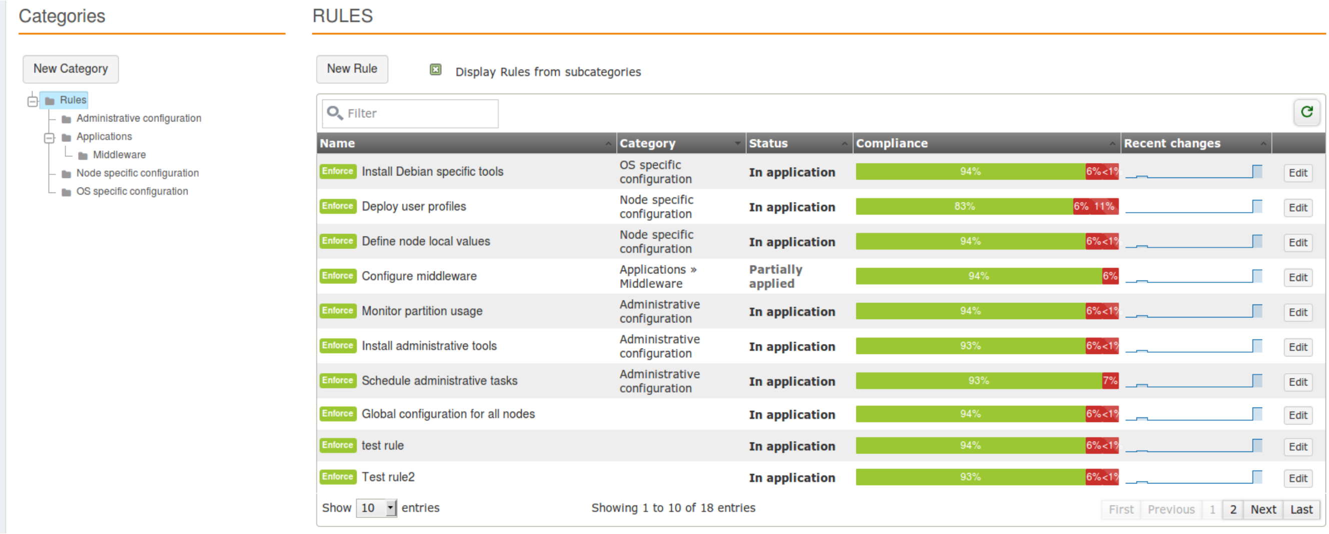
The simple primitives can be simply mixed and extended to provide solutions for any and all of your unique use cases of software stacks, deployments, IT services or configuration that can't be natively supported.

2.2.2 Centralize and aggregate real configuration states

The nominal working mode of *Rudder* is a **continuous verification** mode, which makes *Rudder* manage the whole application life cycle and check that configurations remain valid at any time.



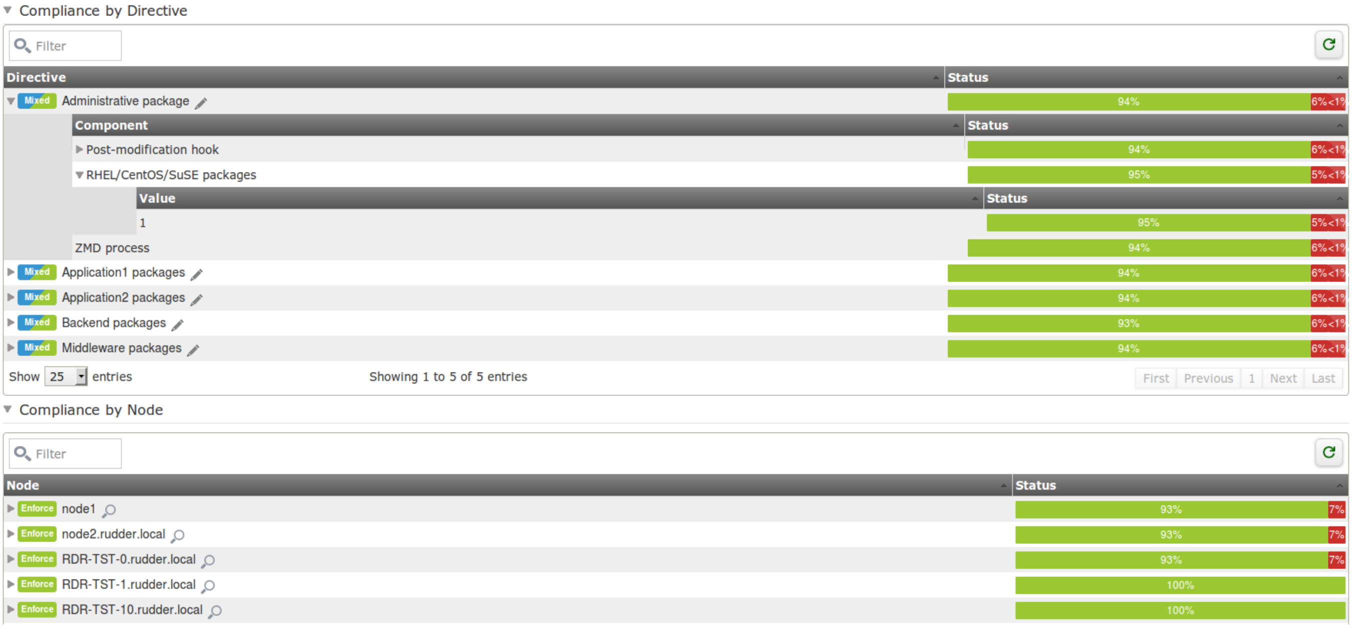
Rudder can also **continuously check** that rules are valid and **proactively** correct any drift from the desired application state when needed. A **graphical reporting** displays what happened and when.


Categories
New Category
Rules
Administrative configuration
Applications
Middleware
Node specific configuration
OS specific configuration
RULES
New Rule
Display Rules from subcategories
Filter

Name	Category	Status	Compliance	Recent changes	
Enforce Install Debian specific tools	OS specific configuration	In application	94% 6% <1%		Edit
Enforce Deploy user profiles	Node specific configuration	In application	83% 6% 11%		Edit
Enforce Define node local values	Node specific configuration	In application	94% 6% <1%		Edit
Enforce Configure middleware	Applications » Middleware	Partially applied	94% 6%		Edit
Enforce Monitor partition usage	Administrative configuration	In application	94% 6% <1%		Edit
Enforce Install administrative tools	Administrative configuration	In application	93% 6% <1%		Edit
Enforce Schedule administrative tasks	Administrative configuration	In application	93% 7%		Edit
Enforce Global configuration for all nodes		In application	94% 6% <1%		Edit
Enforce test rule		In application	94% 6% <1%		Edit
Enforce Test rule2		In application	93% 6% <1%		Edit

Show 10 entries Showing 1 to 10 of 18 entries First Previous 1 2 Next Last

Rudder can notify the ops team about a drift from the desired configuration state. Understanding what the problem is is made simpler by the graphical reporting which allows to drill down toward the technical root cause and see in a blink where the drift comes from.



▼ Compliance by Directive

Filter

Directive	Status
▼ Mixed Administrative package	94% 6% <1%

Component	Status
▶ Post-modification hook	94% 6% <1%
▼ RHEL/CentOS/SuSE packages	95% 5% <1%

Value	Status
1	95% 5% <1%

Component	Status
ZMD process	94% 6% <1%

Directive	Status
▶ Mixed Application1 packages	94% 6% <1%
▶ Mixed Application2 packages	94% 6% <1%
▶ Mixed Backend packages	93% 6% <1%
▶ Mixed Middleware packages	94% 6% <1%

Show 25 entries Showing 1 to 5 of 5 entries First Previous 1 Next Last

▼ Compliance by Node

Filter

Node	Status
▶ Enforce node1	93% 7%
▶ Enforce node2.rudder.local	93% 7%
▶ Enforce RDR-TST-0.rudder.local	93% 7%
▶ Enforce RDR-TST-1.rudder.local	100%
▶ Enforce RDR-TST-10.rudder.local	100%

2.2.3 Automatic inventory

Rudder automatically does a technical, detailed inventory of the servers on which the agent is installed. That inventory contains hardware information (like server kind, CPU, RAM, hard drives, etc), networks information (network interface and configuration), OS level data (OS type and name, version and patch level, etc) and software information (installed software with their versions).

These informations are available in *Rudder* configuration data base and can be used to defined coniguration rule targets. Typically, some configurations are linked to the kind of server (physical or virtual), the quantity of RAM available, the version of an OS library which contains a security bug, etc.

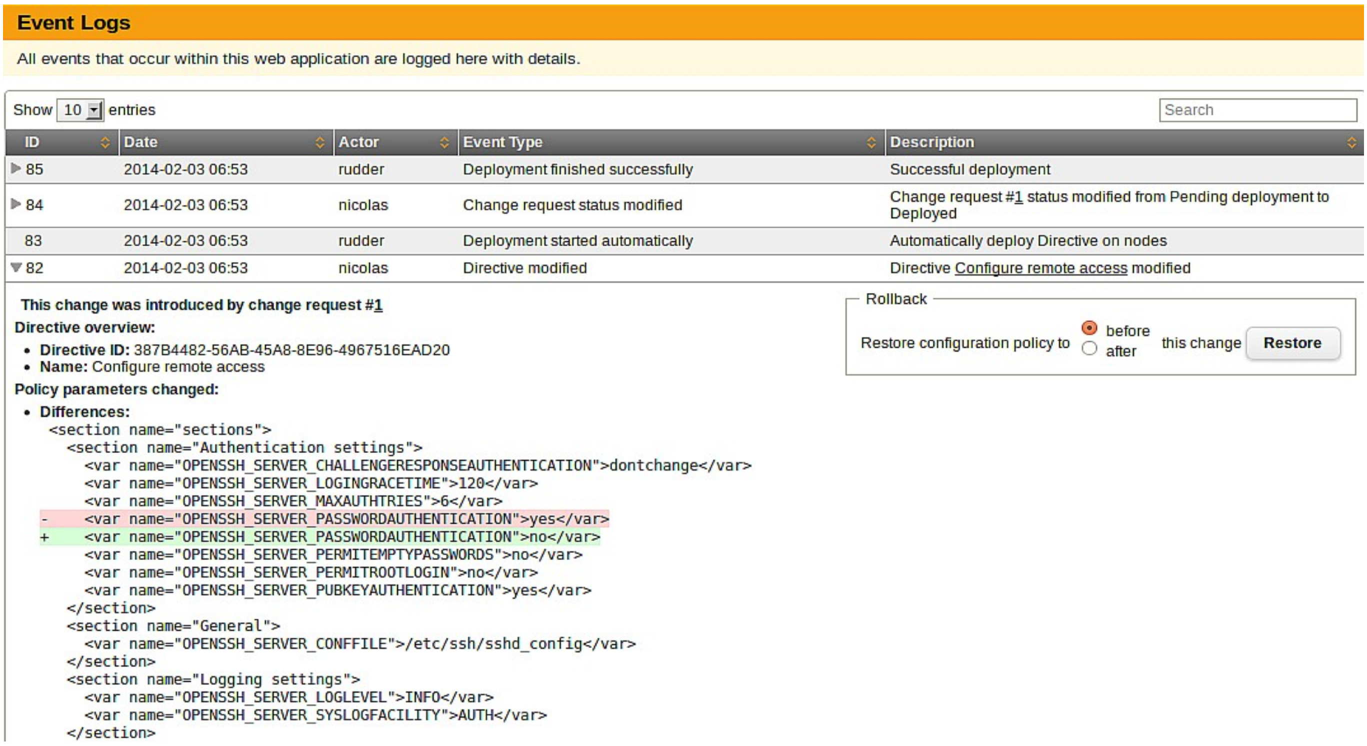
All of these data are also available through Rudder APIs.

2.2.4 REST API

All *Rudder* commands are available through an exhaustive *REST API*. That API is fully documented online and can be used to quickly and smoothly integrate Rudder with your existing infrastructure.

2.2.5 Audit trace and Change Requests

Any change done thanks to *Rudder* in your infrastructure is automatically recorded in an **Audit Log** which allows a full traceability of all changes. That feature also allows rollbacks of the recorded change.



Event Logs

All events that occur within this web application are logged here with details.

Show 10 entries · Search

ID	Date	Actor	Event Type	Description
▶ 85	2014-02-03 06:53	rudder	Deployment finished successfully	Successful deployment
▶ 84	2014-02-03 06:53	nicolas	Change request status modified	Change request #1 status modified from Pending deployment to Deployed
83	2014-02-03 06:53	rudder	Deployment started automatically	Automatically deploy Directive on nodes
▼ 82	2014-02-03 06:53	nicolas	Directive modified	Directive Configure remote access modified

This change was introduced by change request #1

Directive overview:

- **Directive ID:** 387B4482-56AB-45A8-8E96-4967516EAD20
- **Name:** Configure remote access

Policy parameters changed:

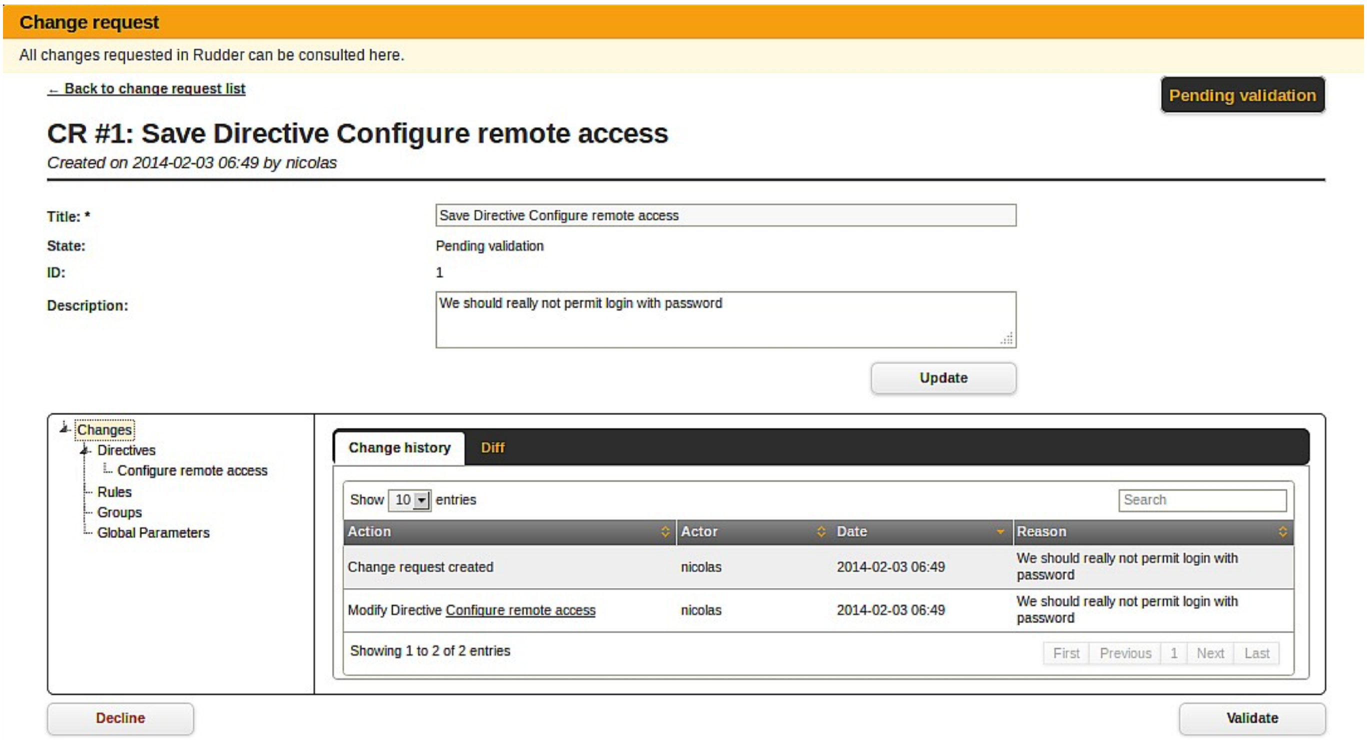
- **Differences:**

```
  <section name="sections">
    <section name="Authentication settings">
      <var name="OPENSSH_SERVER_CHALLENGERESPONSEAUTHENTICATION">dontchange</var>
      <var name="OPENSSH_SERVER_LOGINGRACETIME">120</var>
      <var name="OPENSSH_SERVER_MAXAUTHTRIES">6</var>
-     <var name="OPENSSH_SERVER_PASSWORDAUTHENTICATION">yes</var>
+     <var name="OPENSSH_SERVER_PASSWORDAUTHENTICATION">no</var>
      <var name="OPENSSH_SERVER_PERMITEMPTYPASSWORDS">no</var>
      <var name="OPENSSH_SERVER_PERMITROOTLOGIN">no</var>
      <var name="OPENSSH_SERVER_PUBKEYAUTHENTICATION">yes</var>
    </section>
    <section name="General">
      <var name="OPENSSH_SERVER_CONFFILE">/etc/ssh/sshd_config</var>
    </section>
    <section name="Logging settings">
      <var name="OPENSSH_SERVER_LOGLEVEL">INFO</var>
      <var name="OPENSSH_SERVER_SYSLOGFACILITY">AUTH</var>
    </section>
```

Rollback

Restore configuration policy to ● before ○ after this change

All changes can be forced to go through a peer review or validation step and so be part of a conformity process.



Change request

All changes requested in Rudder can be consulted here.

← Back to change request list

Pending validation

CR #1: Save Directive Configure remote access

Created on 2014-02-03 06:49 by nicolas

Title: *	Save Directive Configure remote access
State:	Pending validation
ID:	1
Description:	We should really not permit login with password

Show 10 entries · Search

Action	Actor	Date	Reason
Change request created	nicolas	2014-02-03 06:49	We should really not permit login with password
Modify Directive Configure remote access	nicolas	2014-02-03 06:49	We should really not permit login with password

Showing 1 to 2 of 2 entries

The validation process can be externalized to third party ticketing system, like a CMDB, so that it can integrated into an existing company workflow. This integration is done thanks to an existing plugin or a dedicated synchronisation tool.

2.2.6 Centralized authentication (LDAP, Active Directory, plugins)

Rudder can use enterprise directories (*LDAP*, *Active Directory*) or be connected to an SSO to manage users authentication.

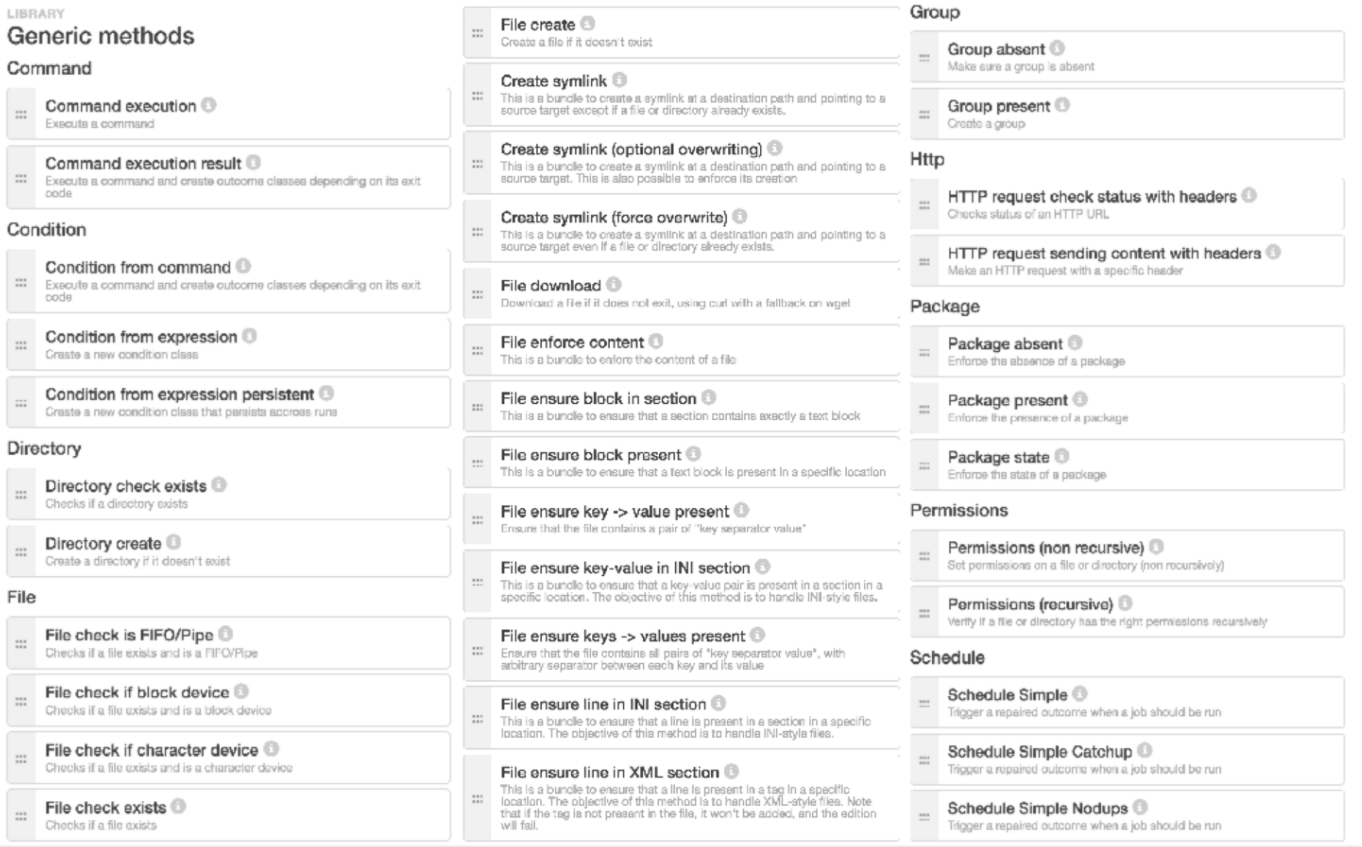
Moreover, *Rudder* authentication layer is plugable and can be extended to other authentication protocol like Radius or SPNEGO with plugins.

2.2.7 Extensibilty

Rudder has a built-in library of common software components and configuration. But of course, your infrastructure is not limited to that handful of standard components and that's why *Rudder* was made to be extremelly simply extended so that it can manage services, process or software specific to your company and your workflows.

To achieve that goal, *Rudder* provided a big set of OS independent and generic, unitary modules. *Rudder* agent is able to translate these abstract modules to native OS specific commands and configurations.

Modules are atomic tasks, that can be extremelly simple (for example, check the existence of a file, create an user or a group, update a software package) or more complexe (for example, import JSON data from a *REST API*). For information, the following image provides a NON-exhaustive list of available modules:



LIBRARY

Generic methods

Command

Command execution
Execute a command

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

Condition

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

Condition from expression
Create a new condition class

Condition from expression persistent
Create a new condition class that persists accross runs

Directory

Directory check exists
Checks if a directory exists

Directory create
Create a directory if it doesn't exist

File

File check is FIFO/Pipe
Checks if a file exists and is a FIFO/Pipe

File check if block device
Checks if a file exists and is a block device

File check if character device
Checks if a file exists and is a character device

File check exists
Checks if a file exists

File create
Create a file if it doesn't exist

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

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

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

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

File enforce content
This is a bundle to enfore the content of a file

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

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

File ensure key -> value present
Ensure that the file contains a pair of "key separator value"

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

File ensure keys -> values present
Ensure that the file contains all pairs of "key separator value", with arbitrary separator between each key and its value

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

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

Group

Group absent
Make sure a group is absent

Group present
Create a group

Http

HTTP request check status with headers
Checks status of an HTTP URL

HTTP request sending content with headers
Make an HTTP request with a specific header

Package

Package absent
Enforce the absence of a package

Package present
Enforce the presence of a package

Package state
Enforce the state of a package

Permissions

Permissions (non recursive)
Set permissions on a file or directory (non recursively)

Permissions (recursive)
Verify if a file or directory has the right permissions recursively

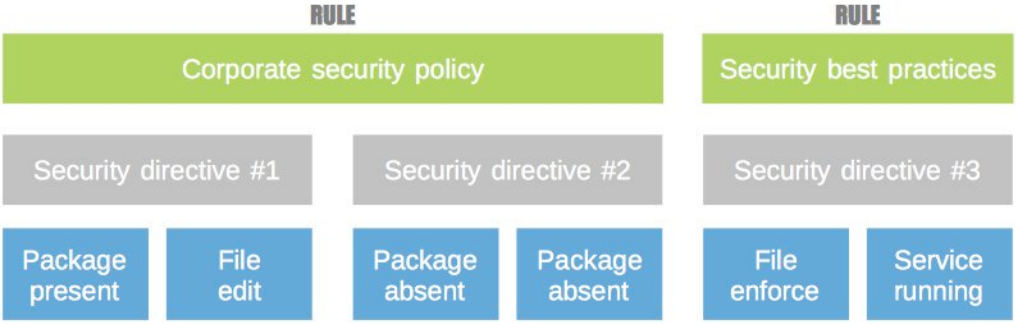
Schedule

Schedule Simple
Trigger a repaired outcome when a job should be run

Schedule Simple Catchup
Trigger a repaired outcome when a job should be run

Schedule Simple Nodups
Trigger a repaired outcome when a job should be run

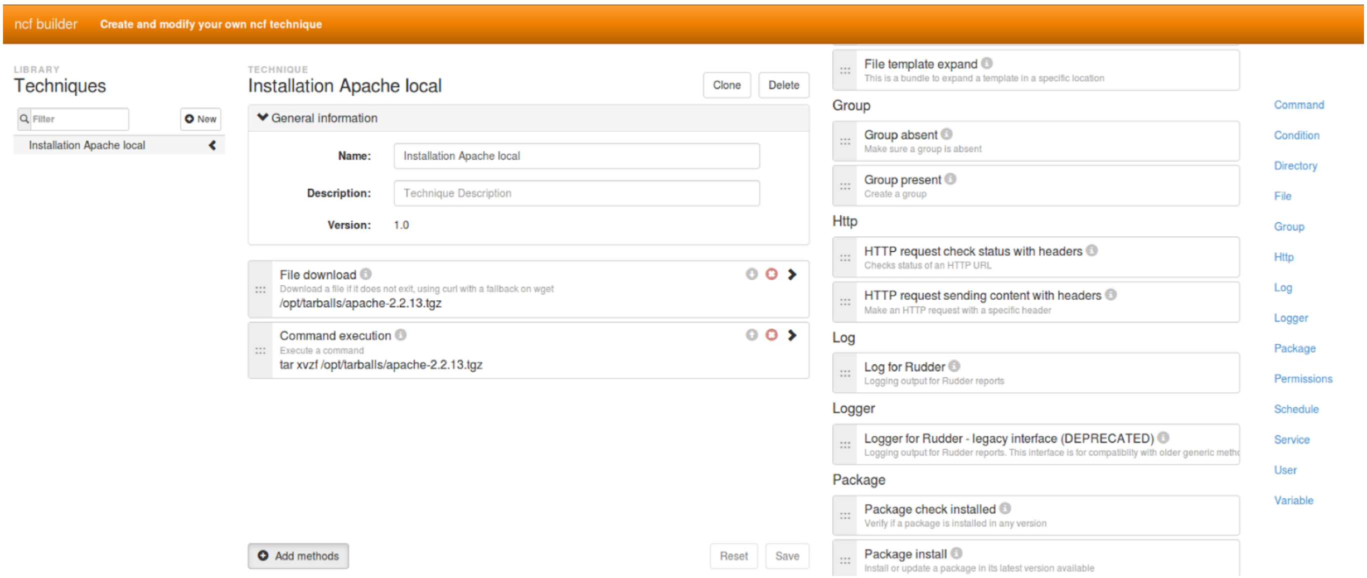
These generic, unitary modules can be used to build new higher level, OS independent, parametrizable configuration modules. By combining these module, you are able to manage any configuration and build advanced configuration policies for your IT services:



The unitary configuration modules can be configured thanks to a high level programming language:

```
# Install the latest apache package
package_install("apache")

# Check that ntp deamon is started (and start it if needed)
service_ensure_started("ntp")

# Check that hosts file exists
file_check_exists("/etc/hosts")
```

But the **natural**, **common** strategy to use them is with the "provided graphical editor" which allows to use all the same modules, but with a **web UI** and with **drag'n'drop**. Of course, you can configure each unitary module to use data from a node and behave specifically on each one.



Installation Apache local

Name: Installation Apache local

Description: Technique Description

Version: 1.0

File download
Download a file if it does not exit, using curl with a fallback on wget
/opt/tarballs/apache-2.2.13.tgz

Command execution
Execute a command
tar xvzf /opt/tarballs/apache-2.2.13.tgz

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

Group

Group absent
Make sure a group is absent

Group present
Create a group

Http

HTTP request check status with headers
Checks status of an HTTP URL

HTTP request sending content with headers
Make an HTTP request with a specific header

Log

Log for Rudder
Logging output for Rudder reports

Logger

Logger for Rudder - legacy interface (DEPRECATED)
Logging output for Rudder reports. This interface is for compatiblity with older generic meth

Package

Package check installed
Verify if a package is installed in any version

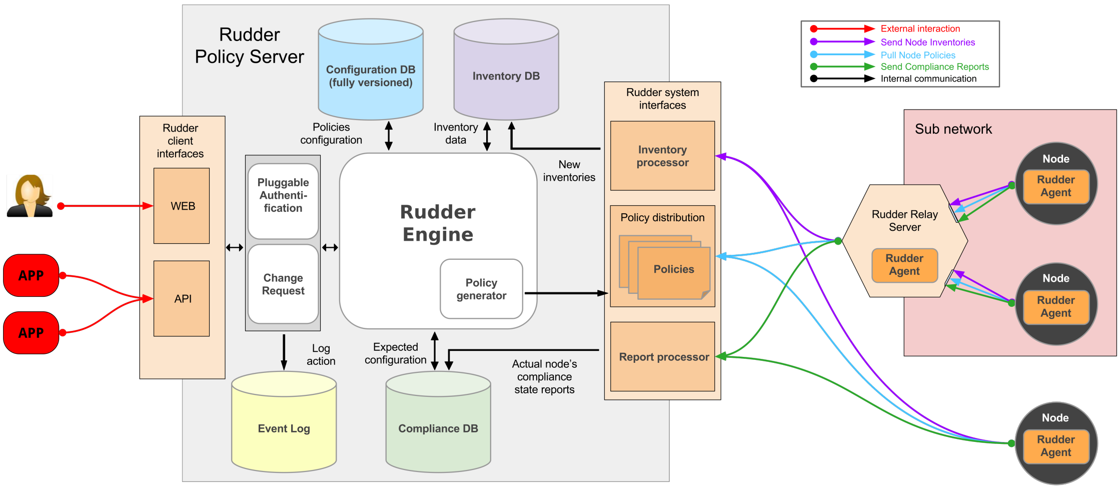
Package install
Install or update a package in its latest version available

2.3 Technical architecture and software dependencies

2.3.1 Functional architecture of Rudder

Rudder contains several functionaly independant components, illustrated in the diagram below:

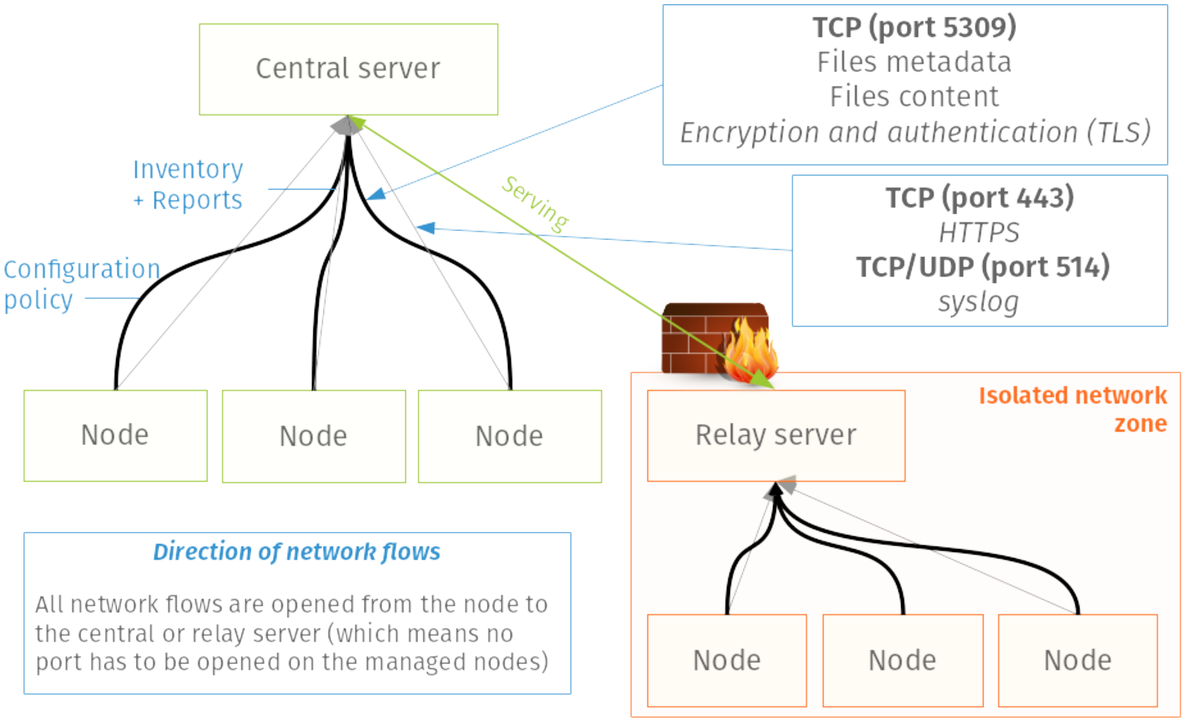
- *Inventory* database
- Configuration policies database
- Compliance database
- Event logs database
- User interface: Web and *REST API*
- *Node* interface: inventory reception, state reports reception, configuration policy sharing
- Relay server to centralize networks flows of an isolated network zone



2.3.2 Network architecture in client/server mode

The *Rudder* server listens for incoming connections from the agents installed on the nodes to manage, and communicates with them. The connection frequency from nodes to server is configurable, from several minutes to several hours.

The following diagram shows the network architecture of a *Rudder* installation:



You can see that relay server allow separating some network areas (for example a DMZ, a specific datacenter or remote site) using a local server for each area to distribute configuration policies and centralize agent reports and inventories.

2.3.3 Agents

*Agent*s can be installed using a simple software package (`.exe`, `.deb` or `.rpm`).

The agent has a very small memory footprint (< 20MB), and is very fast applying configurations (complete runtime below 10 seconds for hundreds of configuration components). It requires at most 500MB of free disk space.

Rudder is currently supported on all major Linux distributions (Red Hat *Enterprise* Linux and derivatives like *CentOS* and Scientific Linux, *Debian*, *Ubuntu*, SUSE Linux *Enterprise*, etc.) for all supported versions, but also for older unsupported ones, *Windows* for desktops and servers (Server 2008 R2 or newer) and AIX (5.3 or newer). Experimental builds for Solaris, FreeBSD, Android and Mac OS X also exist, as well as a version for ARM architecture.

Chapter 3

Installation

3.1 Quick installation

Warning
This is a bit like the insecure `curl | sh`, this is because it is intended for quick testing. Please read the full installation section if you want the complete procedure.
However the URL is HTTPS with a valid certificate.

We have a quick procedure for people who just want to test *Rudder*:

```
su -
wget https://www.rudder-project.org/tools/rudder-setup
chmod +x rudder-setup
./rudder-setup setup-server latest
```

This will setup *Rudder* repository on your system and the use the package manager to install rudder server in latest version.

rudder-setup can also install an agent or a relay in any version, here is the full usage:

```
Usage rudder-setup (add-repository|setup-agent|setup-server|upgrade-agent|upgrade-server) < ↩
    rudder_version> [<policy_server>]
  Adds a repository and setup rudder on your OS
  Should work on as many OS as possible
  Currently suported : Debian, Ubuntu, RHEL, Fedora, Centos, Amazon, Oracle, SLES

  rudder_version : x.y or x.y.z or x.y-nightly or ci/x.y or lts or latest
       x.y:         the last x.y release (ex: 3.2)
       x.y.z:       the exact x.y.z release (ex: 3.2.1)
       x.y.z-t:     the exact x.y.z release with a retag number t (ex: 3.2.1-1)
       x.y.z.a:     the last x.y.z pre-release where a can be alpha1, beta1, rc1... (ex:  ↩
           4.0.0.rc1)
       x.y-nightly: the last public x.y nightly build (ex: 3.2-nightly)
       ci/x.y:      the last private x.y nightly build (ex: ci/3.2)
       ci/x.y.z:    the last private x.y.z release build (ex: ci/3.2.16)
       ci/x.y.z-t:  the last private x.y.z release build with a retag number t (ex: ci ↩
           /3.2.16-1)
       ci/x.y.z.a:  the last private x.y.z pre-release build (ex: ci/4.0.0.rc1)
       lts:         the last long term support version
       latest:      the last stable version
```

3.2 Requirements

3.2.1 Networking

To	**From**	**Port**	**Usage**
Root Server	User or API client	**tcp/443** (https)	Access Web interface/API
Policy Server	Any *Node*	**udp/514** (or tcp/514)	Send reports
	Linux or AIX *Node*	**tcp/443** (https/WebDAV)	Send inventories
		tcp/5309	Fetch policies
		tcp/5310 (optional)	Debug policy copy
	AIX *Node*	**tcp/80** (http/WebDAV)	Send inventories
	Windows DSC *Node*	**tcp/443** (https/WebDAV)	Send inventories and fetch policies
Linux or AIX *Node*	Policy Server	*tcp/5309 (optional)*	Trigger remote agent run

Table 3.1: Network Flows

Note: The Policy Server is the server configured to manage the node, and can be either a Root Server or a Relay Server.

3.2.1.1 DNS - Name resolution

If you want to be able to trigger agent runs from the Root Server (without having to wait for regular automated run), you will need your Root Server (or Relay Server) to be able to resolve your nodes using the provided hostname.

3.2.2 JVM Security Policy

Rudder needs `unlimited strength` security policy because it uses a variety of advanced hashing and cryptographic algorithms only available in that mode.

Any recent JVM (JDK 8 > 8u161, all JDK 9 and more recent) is configured by default with this policy.

You can check your case by running the following command on your server:

```
jrunscript -e 'exit (javax.crypto.Cipher.getMaxAllowedKeyLength("RC5") >= 256 ? 0 : 1);';  ↩
    echo $?
```

If it returns 0, you have the correct policy. In other cases, you will need to change it.

For that, you can download the `unlimited strength` policy for JDK 8 here.

Then, simply copy the `java.policy` file into `$JAVA_HOME/jre/lib/security/java.policy`.

3.2.3 Fully supported Operating Systems

Fully supported Operating Systems are systems that are frequently built and tested on our servers. Partially suported Operating Systems are systems that have been built and tested at least once but that have not seen continuous flow of fixes.

3.2.3.1 For Rudder Nodes

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

GNU/Linux:

- *Debian* 5 to 9

- RedHat *Enterprise* Linux (*RHEL*) / *RHEL*-like 3 and 5 to 7
- *Fedora* 18
- *SuSE* Linux *Enterprise* Server (SLES) 10 SP3, 11 SP1, 11 SP2, 11 SP3, 11 SP4, 12, 12 SP1, 12 SP2
- *Ubuntu* 10.04 LTS (Lucid), 12.04 LTS (Precise), 12.10 (Quantal), 14.04 LTS (Trusty), 16.04 LTS (Xenial), 18.04 LTS (Bionic)

Other Unix systems:

- IBM AIX 5.3, 6.1 and 7.1

Windows:

- Microsoft Windows Server 2008 R2, 2012, 2012 R2, 2016

Windows and AIX Nodes

- On *Windows*, installing *Rudder* requires the DSC (Desired State Configuration) plugin and Powershell 4.0 or more
- For IBM AIX, pre-built RPM packages are distributed by *Normation* only

Hence, as a starting point, we suggest that you only use Linux machines. Once you are accustomed to *Rudder*, contact *Normation* to obtain a demo version for these platforms.

3.2.3.2 For Rudder Root Server

The following operating systems are supported as a Root server:

GNU/Linux:

- *Debian* 8 and 9
- RedHat *Enterprise* Linux (*RHEL*) / *RHEL*-like 6 and 7
- *SuSE* Linux *Enterprise* Server (SLES) 11 SP1 and SP3, 12 SP1, 12 SP2
- *Ubuntu* 14.04 LTS (Trusty), 16.04 LTS (Xenial)

3.2.4 Partially supported Operating Systems

Fully supported Operating Systems are systems that are frequently built and tested on our servers. Partially suported Operating Systems are systems that have been built and tested at least once but that have not seen continuous flow of fixes.

Partially supported Operating Systems

It is possible to use *Rudder* on other platforms than the fully supported ones. However, we haven't tested the application on them, and can't currently supply any packages for them. Moreover, some *Techniques* may not work properly. If you wish to get *Rudder* support on those systems, please get in touch with us!

A reference about how to manually build a *Rudder* agent is available on *Rudder*'s documentation here: Building the Rudder Agent

3.2.4.1 For Rudder Nodes

The following operating systems have had an agent built using Building the Rudder Agent:

- FreeBSD
- Slackware
- Solaris 10 and 11
- Raspbian, based on jessie (via dpkg)
- *Debian* 8 on ARM (armhf version) (via dpkg)
- OpenSUSE (via rpm)

You can also follow the documentation instructions to build and install *Rudder* Agent locally on your favorite linux distribution. Even if this distribution has not been tested by us, it has a reasonable chance of success.

3.2.4.2 For Rudder Root Server

We advise against using an unsupported OS for *Rudder* server because the server contains much more code than the agent. This code is tailored against specific OS versions to work around many system limitations and specificities.

3.2.5 Cloud compatibility

The agent provides an abstraction that permits a high level management of the infrastructure. This abstraction is independant of the underlying hardware. This also works for the cloud - we can define configuration rules in *Rudder* that will be applied as well inside a cloud instance as in a virtual server or in a physical machine of a datacenter.

Any cloud instance based on one of the supported operating system is automatically supported.

3.2.6 Hardware specifications for Rudder Agent

Rudder agent has a very small footprint, and only consumes:

- 10 to 20 MB of RAM during an agent run
- a few kB on the network to check or update its policies
- a few kB on the network to report
- around 100 MB of disk space for the installed files and the workspace

These figures will vary depending on your configuration (backup retention, number of configured components to check, etc…).

3.2.7 Hardware specifications and sizing for Rudder Root Server

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

Note
Rudder does not fear big infrastructures. It is currently used in production in infrastructure with more than **7000** nodes.

3.2.7.1 Memory

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

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

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

When your server has more than 2 GB of RAM, you have to configure the RAM allocated to the Java Virtual Machine as explained in the section about webapplication RAM configuration.

When your server has more than 4 GB, you may need to also tune the PostgresSQL server, as explained in the Optimize PostgreSQL Server section.

Tip
As an example, a *Rudder* server which manages 2600 nodes (with a lot of policies checked) will need:

- A server with 8 GB of RAM,
- 4 GB of RAM will be allocated to the JVM.

In our load-tests, with such a configuration, the server is not stressed and the user experience is good.

3.2.7.2 Disk

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

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

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

Follow the Reports Retention section to configure the retention duration.

Warning
Be careful to correctly size your /**var** partition. Compliance data are growing fast, and PostgreSQL doesn't like at all to encounter a write error because the disk is full. It is also adviced to set-up your monitoring to check for available space on that partition.

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

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

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

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

3.3.1 Install Rudder Root server on Debian or Ubuntu

Warning

Any nodes running **syslogd** (not syslog-ng or rsyslog) will **fail** to send any reports about the configuration rules they have applied to a *Rudder Server* running on *Ubuntu* (and only on *Ubuntu*). *Rudder* will apply rules on nodes but will never get reports from them. Therefore *Rudder* will not be able to calculate compliance.

The only supported platform using syslogd by default is ***RHEL***/***CentOS*** **5**, and several workarounds are available to fix this:

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

3.3.1.1 Add the Rudder packages repository

Rudder 4.2 requires Java RE (version 8 at least) which is not packaged by default on *Debian* 7 nor *Ubuntu* 14.04.

The Java RE 8 for *Debian* or *Ubuntu* can be found through *Oracle*'s website: https://www.java.com

Each package that is published by *Rudder* Project is signed with our GPG signature. To ensure the packages you will install are official builds and have not been altered, import our key into apt using the following command:

```
wget --quiet -O- "https://www.rudder-project.org/apt-repos/rudder_apt_key.pub" | sudo apt- ↩
    key add -
```

Our key fingerprint is:

```
pub   4096R/474A19E8 2011-12-15 Rudder Project (release key) <security@rudder-project.org>
      Key fingerprint = 7C16 9817 7904 212D D58C  B4D1 9322 C330 474A 19E8
```

Then run the following commands as root:

```
echo "deb http://www.rudder-project.org/apt-4.2/ $(lsb_release -cs) main" > /etc/apt/ ↩
    sources.list.d/rudder.list
apt-get update
```

This will add the package repository and finally update the local package cache.

3.3.1.2 Install your Rudder Root Server

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

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

3.3.2 Initial configuration of your Rudder Root Server

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

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

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

Allowed networks A list of IP networks authorized to connect to the server. It uses the network/CIDR mask notation, for instance `192.168.0.0/24` or `10.0.0.0/8`. To add several networks, first type the first network, then press the return key - the script will ask if you wish to add some more networks. Also, the allowed networks can be adjusted later in the web interface in the Administration - Settings tab without having to run the script again.

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

3.3.3 Validate the installation

Once all these steps have been completed, use your web browser to go to the URL given in the output of `rudder-init`.

You should see a loading screen, then a login prompt. The default login is "admin" with password "admin", authenticating you in the *Rudder* web interface with full administrative privileges. You are strongly advised to change this password as soon as possible.

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

3.3.4 Install Rudder Root server on SLES

3.3.4.1 Configure the package manager

Rudder requires Java RE (version 7 at least) that is not always packaged by *SuSE* on all versions

- PostgreSQL 9
- Java RE (version 8 at least).

It is also recommended to use PostgreSQL >= 9.2 for optimal performances.

PostgreSQL 9.4 can be installed through the Open*SuSE* build service: https://build.opensuse.org/project/show/server:database:postgresql or through the system repositories, on SLES 11 SP4 and later systems.

The Java RE 8 for SLES11 can be found through *Oracle*'s website: https://www.java.com

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

Warning

SLES 11 pre SP4 will try to install PostgreSQL 8.x by default, which is not recommended for *Rudder* and will cause serious performance degradation, and requires much more disk space in the long run.
It is really recommended to either add the Open*SuSE* build service repository, or install postgresql9x-server (if available) beforehand to prevent the system from choosing the default PostgreSQL version.

Warning

You may encounter a segmentation fault in Zypper in the following cases:

- On SLES 11 when trying to install *Rudder* rpm files locally with Zypper (for example with *zypper install rudder-agent-version.release-1.SLES.11.x86_64.rpm*)
- On SLES 12 GA when installing *Rudder* packages, locally or from the repository

This is due to a bug (bnc#929483 on *SuSE* bugtracker) in Zypper's RPM headers parsing. You can either:

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

Warning

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

3.3.4.2 Add the Rudder packages repository

Each package that is published by *Rudder* Project is signed with our GPG signature. To ensure the packages you will install are official builds and have not been altered, import our key into rpm using the following command:

```
rpm --import https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub
```

Our key fingerprint is:

```
pub  1024R/6F07D355 2012-11-09 Rudder Project (RPM release key) <security@rudder-project. ↩
    org>
      Key fingerprint = 1141 A947 CDA0 4E83 82C1  B9C4 ADAB 3BD3 6F07 D355
```

Then run the following commands as root:

```
zypper ar -n "Rudder SLES repository" http://www.rudder-project.org/rpm-4.2/SLES_11/ Rudder
zypper refresh
```

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

3.3.4.3 Install your Rudder Root Server

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

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

3.3.5 Initial configuration of your Rudder Root Server

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

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

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

Allowed networks A list of IP networks authorized to connect to the server. It uses the network/CIDR mask notation, for instance `192.168.0.0/24` or `10.0.0.0/8`. To add several networks, first type the first network, then press the return key - the script will ask if you wish to add some more networks. Also, the allowed networks can be adjusted later in the web interface in the Administration - Settings tab without having to run the script again.

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

3.3.6 Validate the installation

Once all these steps have been completed, use your web browser to go to the URL given in the output of `rudder-init`.

You should see a loading screen, then a login prompt. The default login is "admin" with password "admin", authenticating you in the *Rudder* web interface with full administrative privileges. You are strongly advised to change this password as soon as possible.

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

3.3.7 Install Rudder Root server on RHEL-like systems

3.3.7.1 Add the Rudder packages repository

Each package that is published by *Rudder* Project is signed with our GPG signature. To ensure the packages you will install are official builds and have not been altered, import our key into rpm using the following command:

```
rpm --import https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub
```

Our key fingerprint is:

```
pub  1024R/6F07D355 2012-11-09 Rudder Project (RPM release key) <security@rudder-project. ↩
    org>
      Key fingerprint = 1141 A947 CDA0 4E83 82C1  B9C4 ADAB 3BD3 6F07 D355
```

Then run the following command as root:

```
echo '[Rudder_4.2]
name=Rudder 4.2 EL repository
baseurl=http://www.rudder-project.org/rpm-4.2/RHEL_$releasever/
gpgcheck=1
gpgkey=https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub' > /etc/yum.repos.d/ ↩
    rudder.repo
```

3.3.7.2 Install your Rudder Root Server

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

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

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

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

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

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

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

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

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

3.3.8 Initial configuration of your Rudder Root Server

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

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

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

Allowed networks A list of IP networks authorized to connect to the server. It uses the network/CIDR mask notation, for instance `192.168.0.0/24` or `10.0.0.0/8`. To add several networks, first type the first network, then press the return key - the script will ask if you wish to add some more networks. Also, the allowed networks can be adjusted later in the web interface in the Administration - Settings tab without having to run the script again.

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

3.3.9 Validate the installation

Once all these steps have been completed, use your web browser to go to the URL given in the output of `rudder-init`.

You should see a loading screen, then a login prompt. The default login is "admin" with password "admin", authenticating you in the *Rudder* web interface with full administrative privileges. You are strongly advised to change this password as soon as possible.

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

Files installed by the application

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

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

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

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

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

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

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

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

3.4 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 detailed description of the workflow, please refer to the Node Management documentation.

Components
This agent contains the following tools:

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

These components are recognized for their reliability and minimal impact on performances. Our tests showed their memory consumption is usually under 10 MB of RAM during their execution. So you can safely install them on your servers.
We grouped all these tools in one package, to ease the *Rudder* Agent installation.
To get the list of supported Operating systems, please refer to the list of supported Operating Systems for the Nodes.

3.4.1 Install Rudder Agent on Debian or Ubuntu

The *Rudder* agent requires that a syslog server is installed on the node. This syslog server can be either:

- syslogd
- syslog-ng
- rsyslog

One of them is generally installed by default, but it may not be the case with minimal images. In this case you should install one of them (preferably syslog-ng or rsyslog).

Each package that is published by *Rudder* Project is signed with our GPG signature. To ensure the packages you will install are official builds and have not been altered, import our key into apt using the following command:

```
wget --quiet -O- "https://www.rudder-project.org/apt-repos/rudder_apt_key.pub" | sudo apt- ↩
    key add -
```

Our key fingerprint is:

```
pub  4096R/474A19E8 2011-12-15 Rudder Project (release key) <security@rudder-project.org>
      Key fingerprint = 7C16 9817 7904 212D D58C  B4D1 9322 C330 474A 19E8
```

Then add *Rudder*'s package repository:

```
echo "deb http://www.rudder-project.org/apt-4.2/ $(lsb_release -cs) main" > /etc/apt/ ↩
    sources.list.d/rudder.list
```

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

```
sudo apt-get update
```

Install the `rudder-agent` package:

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

You can now configure the agent.

3.4.2 Install Rudder Agent on RHEL-like systems

The *Rudder* agent requires that a syslog server is installed on the node. This syslog server can be either:

- syslogd
- syslog-ng
- rsyslog

One of them is generally installed by default, but it may not be the case with minimal images. In this case you should install one of them (preferably syslog-ng or rsyslog).

Each package that is published by *Rudder* Project is signed with our GPG signature. To ensure the packages you will install are official builds and have not been altered, import our key into rpm using the following command:

```
rpm --import https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub
```

Our key fingerprint is:

```
pub  1024R/6F07D355 2012-11-09 Rudder Project (RPM release key) <security@rudder-project. ↩
    org>
      Key fingerprint = 1141 A947 CDA0 4E83 82C1  B9C4 ADAB 3BD3 6F07 D355
```

Then define a yum repository for *Rudder*:

```
echo '[Rudder_4.2]
name=Rudder 4.1 EL repository
baseurl=http://www.rudder-project.org/rpm-4.2/RHEL_$releasever/
gpgcheck=1
gpgkey=https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub' > /etc/yum.repos.d/ ↩
    rudder.repo
```

Tip
The RPM can be directly downloaded for a standalone installation, from the following URL: http://www.rudder-project.org/rpm-4.1/*RHEL*_7/ (or *RHEL*_6, *RHEL*_5, etc, depending on your host's OS version)

Install the package:

```
yum install rudder-agent
```

Or:

```
yum install rudder-agent-4.2.0-1.EL.7.x86_64.rpm
```

You can now configure the agent.

3.4.3 Install Rudder Agent on SLES

The *Rudder* agent requires that a syslog server is installed on the node. This syslog server can be either:

- syslogd
- syslog-ng
- rsyslog

One of them is generally installed by default, but it may not be the case with minimal images. In this case you should install one of them (preferably syslog-ng or rsyslog).

Following commands are executed as the `root` user.

Warning
You may encounter a segmentation fault in Zypper in the following cases:

- On SLES 11 when trying to install *Rudder* rpm files locally with Zypper (for example with *zypper install rudder-agent-version.release-1.SLES.11.x86_64.rpm*)
- On SLES 12 GA when installing *Rudder* packages, locally or from the repository

This is due to a bug (bnc#929483 on *SuSE* bugtracker) in Zypper's RPM headers parsing. You can either:

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

Each package that is published by *Rudder* Project is signed with our GPG signature. To ensure the packages you will install are official builds and have not been altered, import our key into rpm using the following command:

```
rpm --import https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub
```

Our key fingerprint is:

```
pub  1024R/6F07D355 2012-11-09 Rudder Project (RPM release key) <security@rudder-project. ↩
    org>
      Key fingerprint = 1141 A947 CDA0 4E83 82C1  B9C4 ADAB 3BD3 6F07 D355
```

Then add the *Rudder* packages repository:

- on SLES 12:

```
zypper ar -n 'Rudder SLES 12 repository' http://www.rudder-project.org/rpm-4.2/SLES_12/  ↩
    Rudder
```

- on SLES 11:

```
zypper ar -n 'Rudder SLES repository' http://www.rudder-project.org/rpm-4.2/SLES_11_SP1/  ↩
    Rudder
```

- on SLES 10:

```
zypper sa 'http://www.rudder-project.org/rpm-4.2/SLES_10_SP3/' Rudder
```

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

```
zypper ref
```

Install the `rudder-agent` package:

```
zypper install rudder-agent
```

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

You can now configure the agent.

3.4.4 Configure and validate

3.4.4.1 Configure Rudder Agent

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

```
echo '<rudder server ip or hostname>' > /var/rudder/cfengine-community/policy_server.dat
```

Tip
We advise you to use the `IP address` of the *Rudder Root Server*. The DNS name of this server can also be accepted if you have a trusted DNS infrastructure with proper reverse resolutions.

You can now start the *Rudder* service with:

```
service rudder start
```

3.4.4.2 Validate new Node

Several minutes after the start of the agent, a new *Node* should be pending in the *Rudder* web interface. You will be able to browse its inventory, and accept it to manage its configuration with *Rudder*.

You may force the agent to run and send an inventory by issuing the following command:

```
rudder agent inventory
```

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

```
rudder agent run
```

3.5 Install Rudder Relay (optional)

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

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

Using a relay, you are able to:

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

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

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

3.5.1 On the relay

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

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

3.5.2 On the root server

You have to tell the *Rudder Root* server that a node will be a relay. To do so, launch the rudder-node-to-relay script on the root server, supplying the UUID of the host to be considered as a relay. You can find the UUID of your node with the *rudder agent info* command.

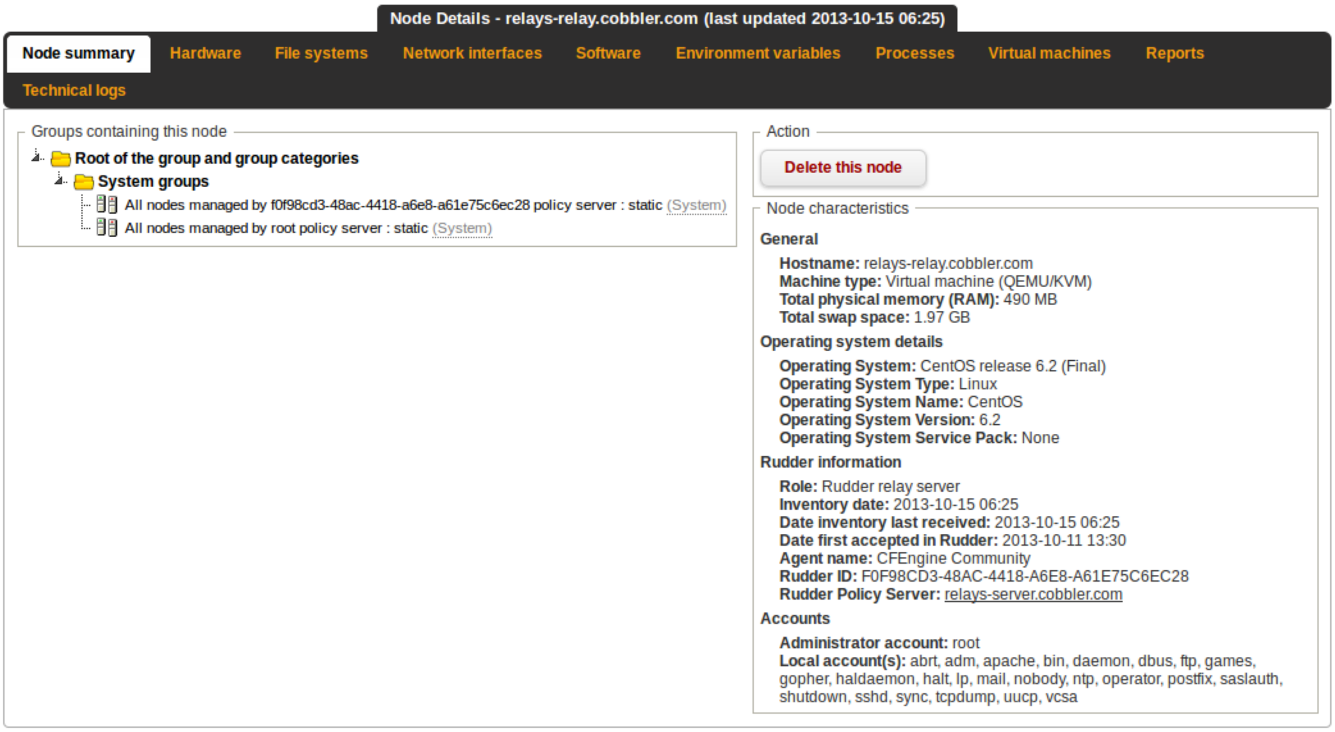
```
/opt/rudder/bin/rudder-node-to-relay aaaaaaaa-bbbb-cccc-dddd-eeeeeeee
```

3.5.3 Validation

When every step has completed successfully:

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

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



Node Details - relays-relay.cobbler.com (last updated 2013-10-15 06:25)

Groups containing this node

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

Action

Node characteristics

General

Hostname: relays-relay.cobbler.com
Machine type: Virtual machine (QEMU/KVM)
Total physical memory (RAM): 490 MB
Total swap space: 1.97 GB

Operating system details

Operating System: CentOS release 6.2 (Final)
Operating System Type: Linux
Operating System Name: CentOS
Operating System Version: 6.2
Operating System Service Pack: None

Rudder information

Role: Rudder relay server
Inventory date: 2013-10-15 06:25
Date inventory last received: 2013-10-15 06:25
Date first accepted in Rudder: 2013-10-11 13:30
Agent name: CFEngine Community
Rudder ID: F0F98CD3-48AC-4418-A6E8-A61E75C6EC28
Rudder Policy Server: relays-server.cobbler.com

Accounts

Administrator account: root
Local account(s): abrt, adm, apache, bin, daemon, dbus, ftp, games, gopher, haldaemon, halt, lp, mail, nobody, ntp, operator, postfix, saslauth, shutdown, sshd, sync, tcpdump, uucp, vcsa

Figure 3.1: Rudder relay node

3.5.4 Adding nodes to a relay server

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

You then have two possible cases:

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

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

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

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

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

```
rudder agent inventory
```

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

Chapter 4

Upgrade

This short chapter covers the upgrade of the *Rudder Server* Root and *Rudder* Agent from older versions to the latest version, 4.2.

The upgrade is quite similar to the installation.

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

4.1 Upgrade notes

4.1.1 Upgrade from Rudder 3.0 or older

Direct upgrades from 3.0.x and older are no longer supported on 4.1. If you are still running one of those, either on servers or nodes, please first upgrade to one of the supported versions above, and then upgrade to 4.1.

4.1.2 Upgrade from Rudder 3.1, 3.2 4.0 or 4.1

Migration from 3.1, 3.2, 4.0 or 4.1 are supported, so you can upgrade directly to 4.2.

Warning

In *Rudder* 4.0, we changed the default communication protocol between agent and server, but still stay compatible with the old protocol. Hence, you can perfectly keep using pre-4.0 agents with a 4.0, 4.1 or 4.2 server.

However, some networking issues may appear when using 4.0, 4.1 or 4.2 agents with older servers with the reverse DNS lookup option disabled in the settings (**Security** → **Use reverse DNS lookups on nodes to reinforce authentication to policy server**).

Therefore, you need to upgrade your server to 4.1 **before** upgrading the nodes so that the configuration distributed to the nodes include the use of the new protocol.

Caution

In *Rudder* 4.1, we changed the name of */opt/rudder/etc/ssl/rudder-webapp.crt* to */opt/rudder/etc/ssl/rudder.crt* and the name of */opt/rudder/etc/ssl/rudder-webapp.key* to */opt/rudder/etc/ssl/rudder.key*.

These certificates are used for *Rudder* internal implementation, but if you are using them for anything else, please update the paths to the files.

For example, if you were using these certificates for configuring sasl in slapd with:

```
TLSCertificateFile /opt/rudder/etc/ssl/rudder-webapp.crt
TLSCertificateKeyFile /opt/rudder/etc/ssl/rudder-webapp.key
```

Then, you now need to use:

```
TLSCertificateFile /opt/rudder/etc/ssl/rudder.crt
TLSCertificateKeyFile /opt/rudder/etc/ssl/rudder.key
```

4.1.3 Compatibility between Rudder agent 4.2 and older server versions

4.1.3.1 4.0.x and 4.1.x servers

Rudder agents 4.2.x are compatible with 4.0 *Rudder* servers.

4.1.3.2 3.1.x and 3.2.x servers

Rudder agents 4.2.x are not compatible with *Rudder* servers older than 4.0.0. You need to upgrade your server to 4.2 before the agents.

4.1.4 Compatibility between Rudder server 4.2 and older agent versions

4.1.4.1 4.0.x and 4.1.x agents

Rudder agent 4.0.x and 4.1.x are fully compatible with *Rudder* server 4.2.x. It is therefore not strictly necessary to update your agents to 4.2.x.

4.1.4.2 3.1.x and 3.2.x agents

Rudder agent 3.1.x and 3.2.x are compatible with *Rudder* server 4.2.x, but they do not support the new "Audit" policy mode. It is therefore not strictly necessary to update your agents to 4.2.x, unless you want to be able to use the "Audit" policy mode.

4.1.4.3 3.0.x or older

These agents are not compatible with *Rudder* 4.2, and you have to upgrade them. Be careful to follow the upgrade path explained above.

4.1.5 Protocol for reporting

Rudder uses syslog messages over UDP by default for reporting (since 3.1), but if you upgraded your server from a previous version, you will keep the previous setting which uses syslog messages over TCP.

You should consider switching to UDP (in **Administration** → **Settings** → **Protocol**), as it will prevent breaking your server in case of networking or load issues, or if you want to manage a lot of nodes. The only drawback is that you can lose reports in these situations. It does not affects the reliability of policy enforcement, but may only temporarily affects reporting on the server. Read perfomance notes about rsyslog for detailed information.

4.1.6 Upgrade manually installed relays (installed before 3.0)

With *Rudder* 2.11, there were no relay package and the configuration had to be done by hand.

To migrate a manually installed relay to 3.1 using the package, run the following intructions:

- Delete the previous *Apache* configuration file:
  - `/etc/httpd/conf.d/rudder-default.conf file` on *RHEL*-like
  - `/etc/apache2/sites-enabled/rudder-default` file on *Debian*-like
  - `/etc/apache2/vhosts.d/rudder-default.conf` file on *SuSE*
- Install the relay package named **rudder-server-relay**.

This is enough to replace the relay configuration, and no change is needed on the root server.

4.1.7 Known issues

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

4.2 On Debian or Ubuntu

Following commands are executed as the `root` user.

Add the *Rudder* project repository:

```
echo "deb http://www.rudder-project.org/apt-4.2/ $(lsb_release -cs) main" > /etc/apt/ ↩
    sources.list.d/rudder.list
```

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

```
apt-get update
```

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

- With apt-get:

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

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

```
service rudder-jetty restart
```

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

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

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

Warning
Rudder 4.1 requires Java RE version 8 or more, which is not packaged be default on *Debian* 7 nor *Ubuntu* 14.04 On these platforms, prior to upgrade *Rudder*, you will need to install Java RE 8, either from *Oracle* site https://www.java.com or through any other means of your choice

You can now upgrade your local techniques.

4.3 On RHEL or CentOS

Following commands are executed as the `root` user.

Update your yum repository:

```
echo '[Rudder_4.2]
name=Rudder 4.2 Repository
baseurl=http://www.rudder-project.org/rpm-4.2/RHEL_$releasever/
gpgcheck=1
gpgkey=https://www.rudder-project.org/rpm-repos/rudder_rpm_key.pub' > /etc/yum.repos.d/ ↩
    rudder.repo
```

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

4.3.1 Rudder server

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

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

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

```
service rudder-jetty restart
```

From version 3.1, *Rudder* provides an SELinux policy. You can enable it after upgrading your server with:

```
sed -i "s%^\s*SELINUX=.*%SELINUX=enabled%" /etc/sysconfig/selinux
setenforce 1
```

4.3.2 Rudder agent

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

```
yum update rudder-agent
```

You can now upgrade your local techniques.

4.4 On SLES

Following commands are executed as the `root` user.

Add the *Rudder* packages repository:

- On a SLES 11 system:

```
zypper ar -n "Rudder SLES repository" http://www.rudder-project.org/rpm-4.2/SLES_11_SP1/  ↩
    Rudder
```

- On a SLES 10 system:

```
zypper sa "http://www.rudder-project.org/rpm-4.2/SLES_10_SP3/" Rudder
```

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

```
zypper ref
```

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

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

Warning
SLES 11 pre SP4 uses PostgreSQL 8.x by default, which is not recommended for *Rudder* and will cause serious performance degradation, and requires much more disk space in the long run.
Rudder 4.0 is tested for PostgreSQL 9.2 and higher. It still works with version 8.4 or 9.1, but not warranties are made that this will hold in the future. It is really recommanded to migrate to PostgreSQL 9.2 at least.
Please look at Install Rudder Root server on SLES for details.

Warning
Rudder 4.1 requires Java RE version 8 or more, which is not packaged be default on SLES 11 On this platform, prior to upgrade *Rudder*, you will need to install Java RE 8, either from *Oracle* site https://www.java.com or through any other means of your choice

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

```
service rudder-jetty restart
```

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

```
zypper update rudder-agent
```

You can now upgrade your local techniques.

4.5 Technique upgrade

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

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

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

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

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

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

This last command will reload the *Technique* library and trigger a full redeployment on nodes.

Please check that the deployment is successful in the *Rudder* web interface.

Chapter 5

Web interface usage

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

5.1 Authentication

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

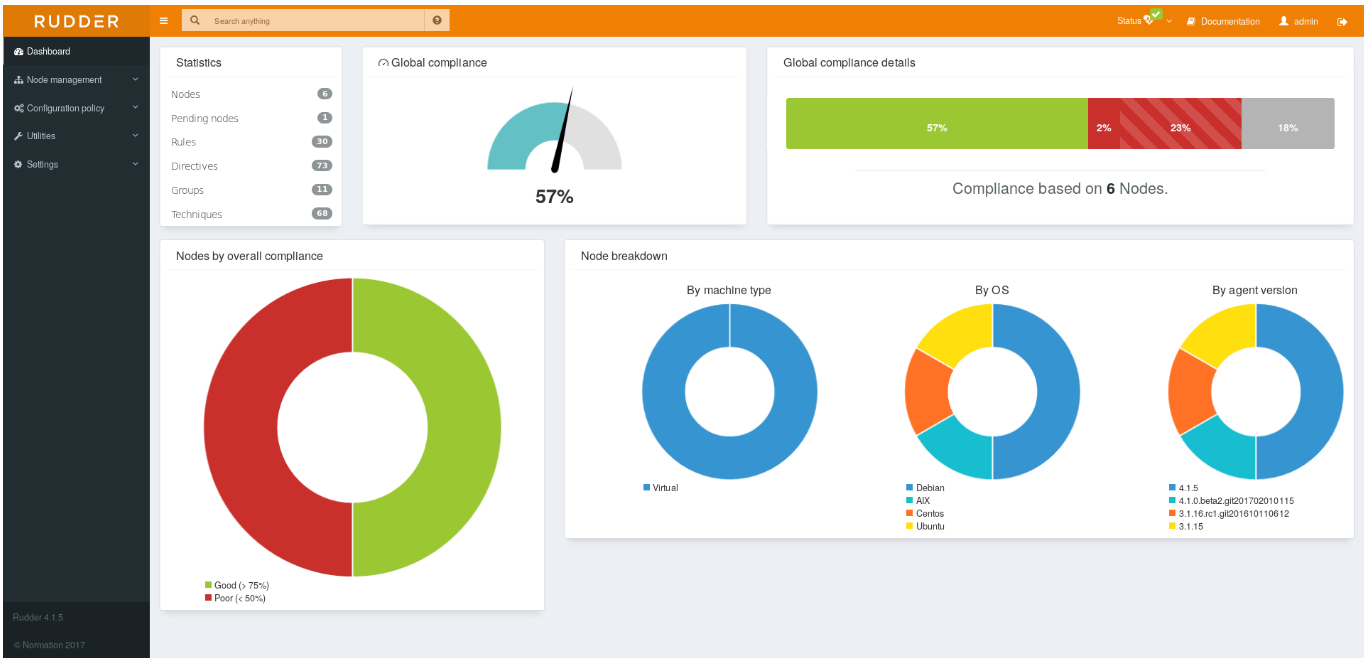
You can change the user accounts by following the User management procedure.

5.2 Presentation of Rudder Web Interface

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

5.2.1 Rudder Home

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



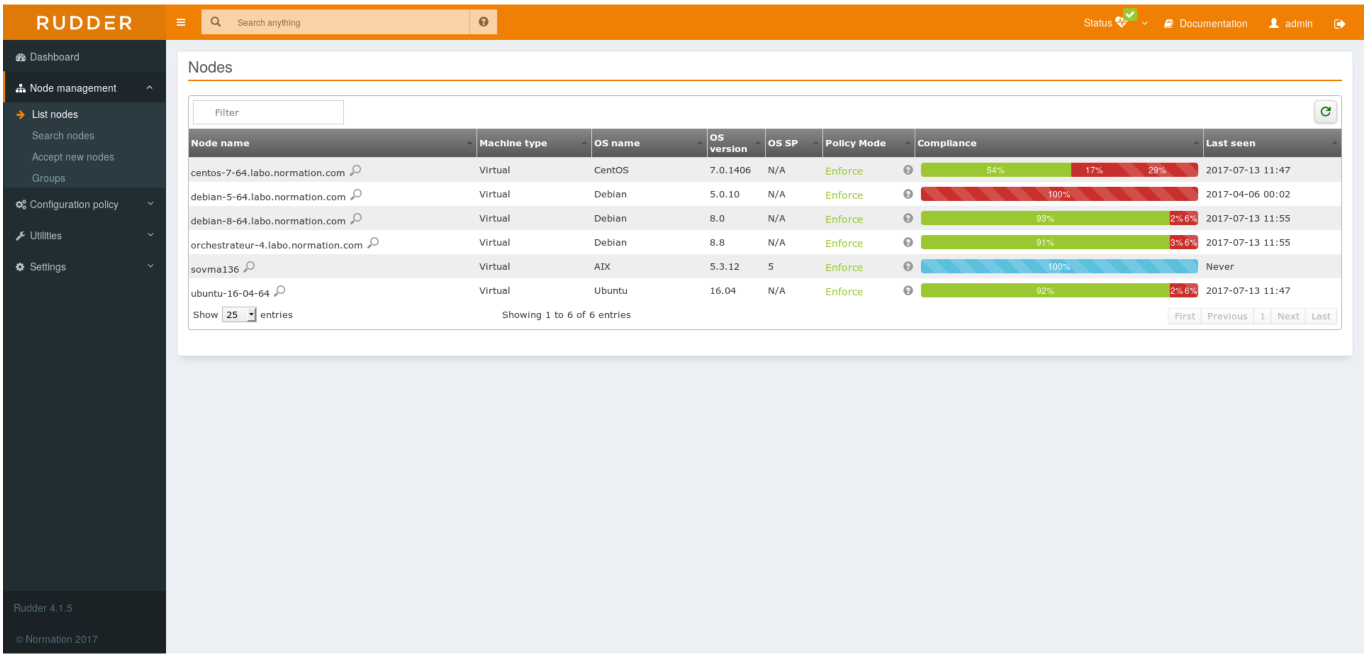
Statistics	
Nodes	6
Pending nodes	1
Rules	30
Directives	73
Groups	11
Techniques	68

Compliance based on **6** Nodes.

Figure 5.1: Rudder Homepage

5.2.2 Node Management

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

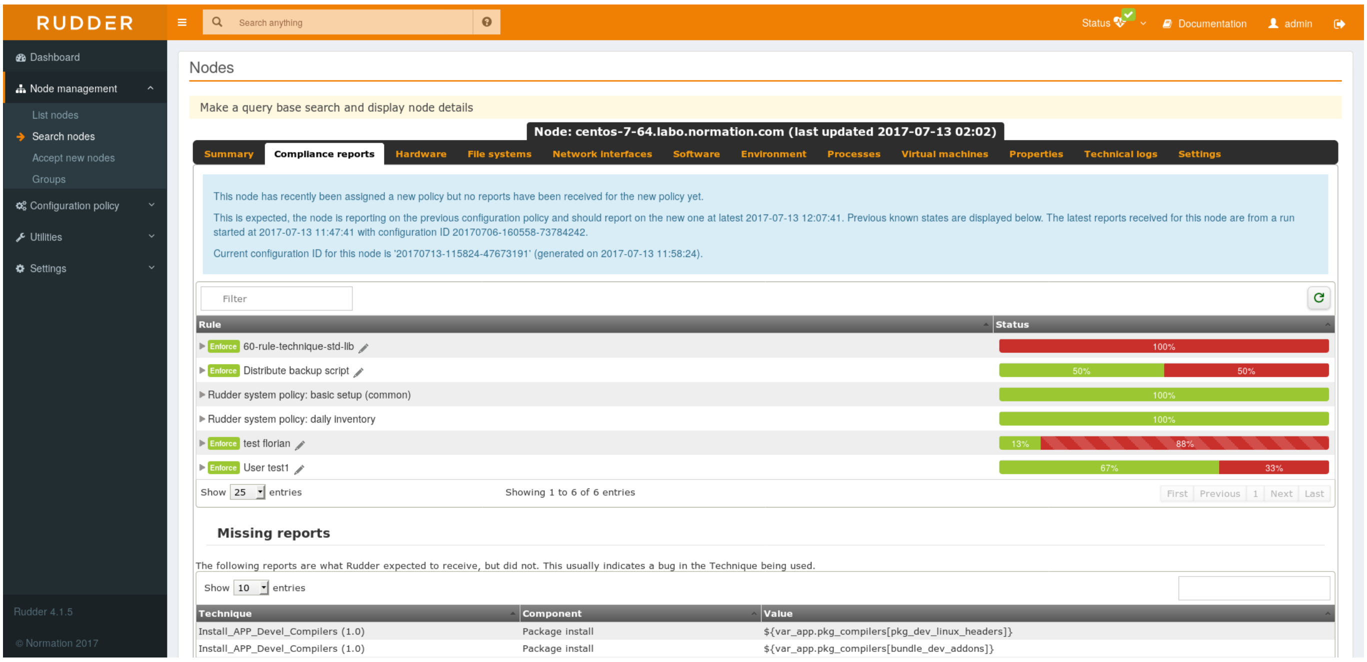


Nodes

Node name	Machine type	OS name	OS version	OS SP	Policy Mode	Compliance	Last seen
centos-7-64.labo.normation.com	Virtual	CentOS	7.0.1406	N/A	Enforce	54% 17% 29%	2017-07-13 11:47
debian-5-64.labo.normation.com	Virtual	Debian	5.0.10	N/A	Enforce	100%	2017-04-06 00:02
debian-8-64.labo.normation.com	Virtual	Debian	8.0	N/A	Enforce	93% 2% 6%	2017-07-13 11:55
orchestrateur-4.labo.normation.com	Virtual	Debian	8.8	N/A	Enforce	91% 3% 6%	2017-07-13 11:55
sovma136	Virtual	AIX	5.3.12	5	Enforce	100%	Never
ubuntu-16-04-64	Virtual	Ubuntu	16.04	N/A	Enforce	92% 2% 6%	2017-07-13 11:47

Show 25 entries Showing 1 to 6 of 6 entries

Figure 5.2: List of Nodes



Nodes

Make a query base search and display node details

Node: centos-7-64.labo.normation.com (last updated 2017-07-13 02:02)

This node has recently been assigned a new policy but no reports have been received for the new policy yet.

This is expected, the node is reporting on the previous configuration policy and should report on the new one at latest 2017-07-13 12:07:41. Previous known states are displayed below. The latest reports received for this node are from a run started at 2017-07-13 11:47:41 with configuration ID 20170706-160558-73784242.

Current configuration ID for this node is '20170713-115824-47673191' (generated on 2017-07-13 11:58:24).

Rule	Status
Enforce 60-rule-technique-std-lib	100%
Enforce Distribute backup script	50% 50%
Rudder system policy: basic setup (common)	100%
Rudder system policy: daily inventory	100%
Enforce test florian	13% 88%
Enforce User test1	67% 33%

Show 25 entries Showing 1 to 6 of 6 entries

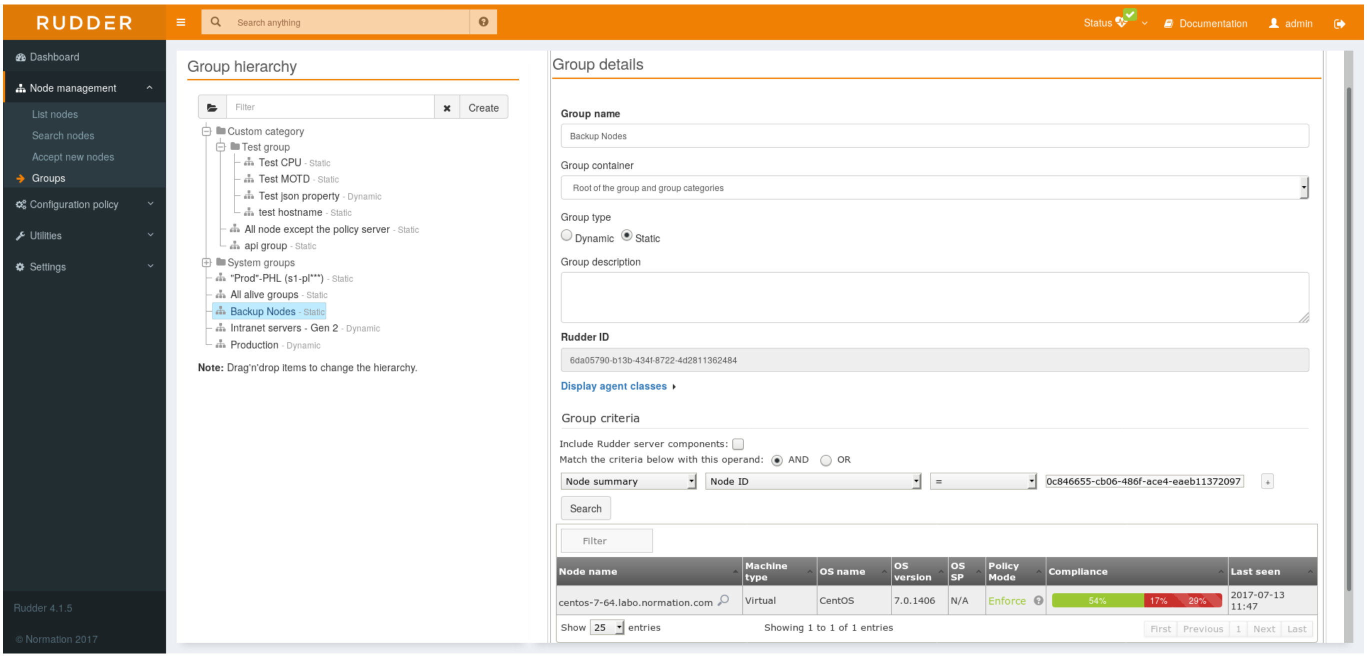
Missing reports

The following reports are what Rudder expected to receive, but did not. This usually indicates a bug in the Technique being used.

Show 10 entries

Technique	Component	Value
Install_APP_Devel_Compilers (1.0)	Package install	${var_app.pkg_compilers[pkg_dev_linux_headers]}
Install_APP_Devel_Compilers (1.0)	Package install	${var_app.pkg_compilers[bundle_dev_addons]}

Figure 5.3: Node compliance



Group hierarchy

- Custom category
  - Test group
    - Test CPU - Static
    - Test MOTD - Static
    - Test json property - Dynamic
    - test hostname - Static
  - All node except the policy server - Static
  - api group - Static
- System groups
- "Prod"-PHL (s1-pl***) - Static
- All alive groups - Static
- Backup Nodes - Static
- Intranet servers - Gen 2 - Dynamic
- Production - Dynamic

Note: Drag'n'drop items to change the hierarchy.

Group details

Group name: Backup Nodes

Group container: Root of the group and group categories

Group type: ○ Dynamic ● Static

Group description

Rudder ID: 6da05790-b13b-434f-8722-4d2811362484

Group criteria

Include Rudder server components: □

Match the criteria below with this operand: ● AND ○ OR

Node summary | Node ID | = | 0c846655-cb06-486f-ace4-eaeb11372097

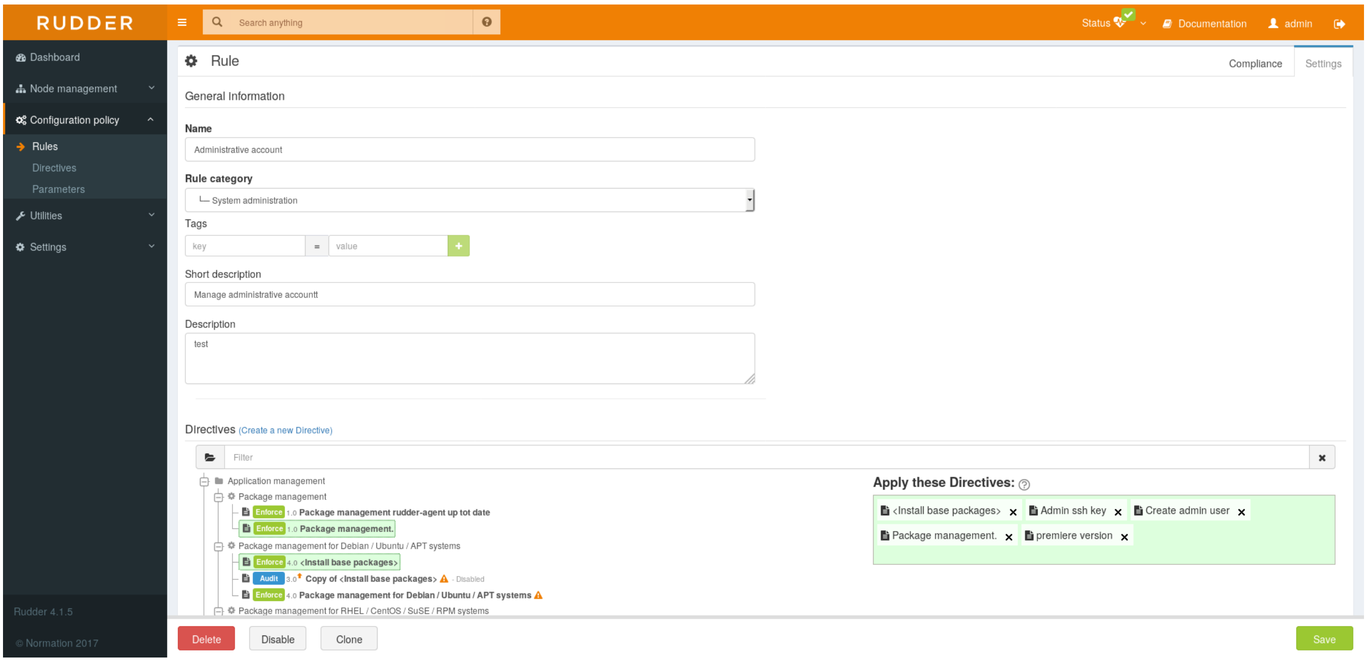
Node name	Machine type	OS name	OS version	OS SP	Policy Mode	Compliance	Last seen
centos-7-64.labo.normation.com	Virtual	CentOS	7.0.1406	N/A	Enforce	54% 17% 29%	2017-07-13 11:47

Show 25 entries Showing 1 to 1 of 1 entries

Figure 5.4: Groups

5.2.3 Configuration Management

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



Rule

General Information

Name
Administrative account

Rule category
└─ System administration

Tags
key = value

Short description
Manage administrative accountt

Description
test

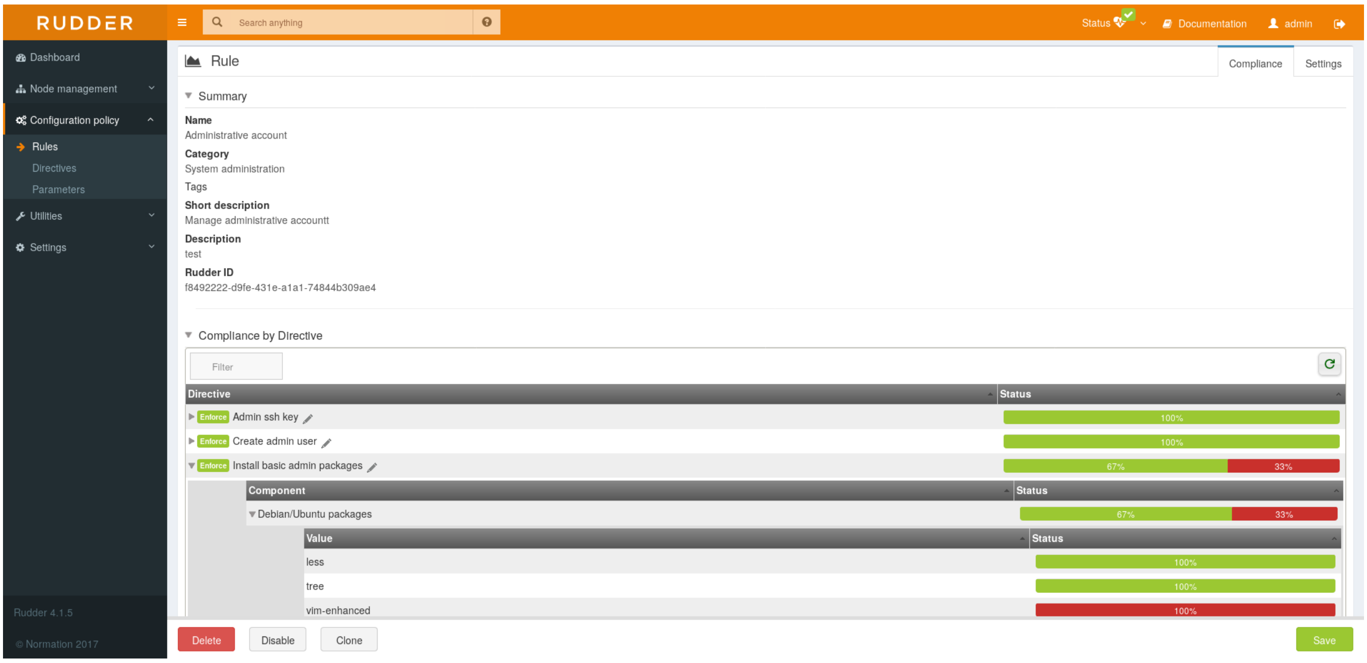
Directives (Create a new Directive)

Filter

- Application management
  - Package management
    - Enforce 1.0 **Package management rudder-agent up tot date**
    - Enforce 1.0 **Package management.**
  - Package management for Debian / Ubuntu / APT systems
    - Enforce 4.0 **<Install base packages>**
    - Audit 3.0 **Copy of <Install base packages>** ⚠ - Disabled
    - Enforce 4.0 **Package management for Debian / Ubuntu / APT systems** ⚠
  - Package management for RHEL / CentOS / SuSE / RPM systems

Apply these Directives:
<Install base packages> ×
Admin ssh key ×
Create admin user ×
Package management. ×
premiere version ×

Figure 5.5: Rules screen



Rule

Summary

Name
Administrative account
Category
System administration
Tags
Short description
Manage administrative accountt
Description
test
Rudder ID
f8492222-d9fe-431e-a1a1-74844b309ae4

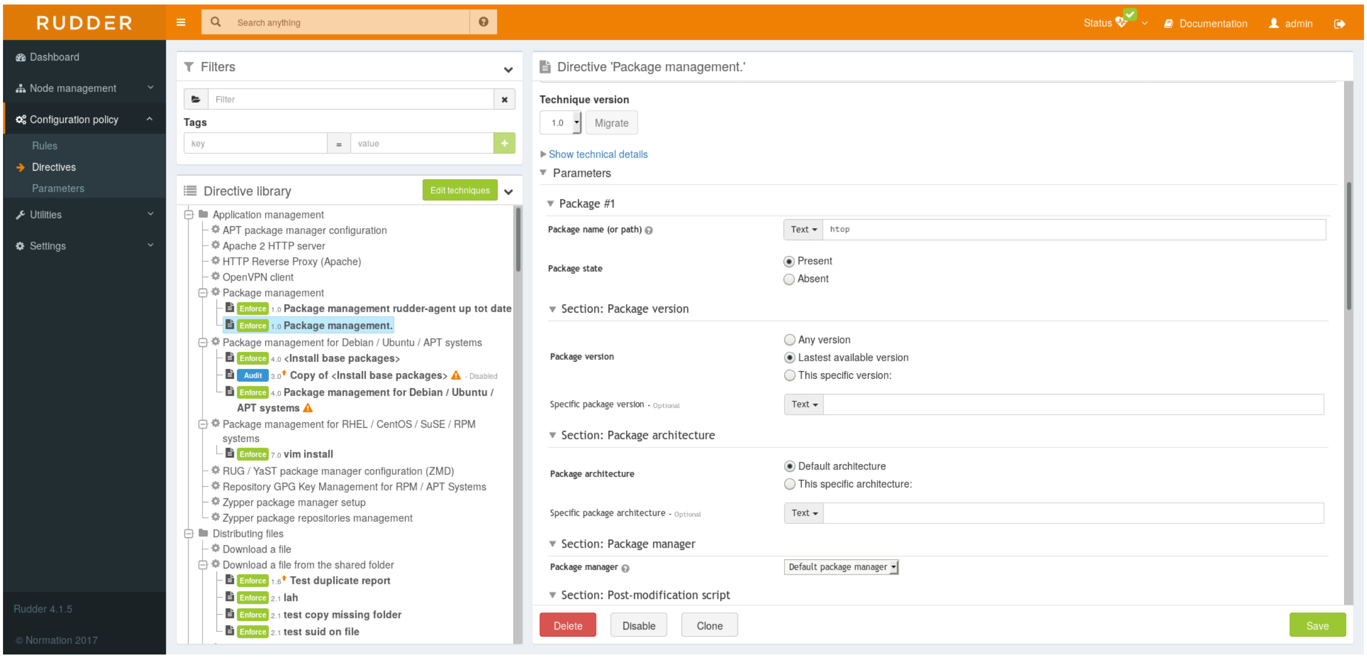
Compliance by Directive

Directive	Status
Enforce Admin ssh key	100%
Enforce Create admin user	100%
Enforce Install basic admin packages	67% 33%

Component	Status
Debian/Ubuntu packages	67% 33%

Value	Status
less	100%
tree	100%
vim-enhanced	100%

Figure 5.6: Rule compliance



- Application management
  - APT package manager configuration
  - Apache 2 HTTP server
  - HTTP Reverse Proxy (Apache)
  - OpenVPN client
  - Package management
    - Enforce 1.0 **Package management rudder-agent up tot date**
    - Enforce 1.0 **Package management.**
  - Package management for Debian / Ubuntu / APT systems
    - Enforce 4.0 **<Install base packages>**
    - Audit 3.0 **Copy of <Install base packages>** - Disabled
    - Enforce 4.0 **Package management for Debian / Ubuntu / APT systems**
  - Package management for RHEL / CentOS / SuSE / RPM systems
    - Enforce 7.0 **vim install**
  - RUG / YaST package manager configuration (ZMD)
  - Repository GPG Key Management for RPM / APT Systems
  - Zypper package manager setup
  - Zypper package repositories management
- Distributing files
  - Download a file
  - Download a file from the shared folder
    - Enforce 1.6 **Test duplicate report**
    - Enforce 2.1 **lah**
    - Enforce 2.1 **test copy missing folder**
    - Enforce 2.1 **test suid on file**

Directive 'Package management.'

Technique version 1.0 Migrate

▶ Show technical details

▼ Parameters

▼ Package #1

Package name (or path) Text htop

Package state ● Present ○ Absent

▼ Section: Package version

Package version ○ Any version ● Lastest available version ○ This specific version:

Specific package version - Optional Text

▼ Section: Package architecture

Package architecture ● Default architecture ○ This specific architecture:

Specific package architecture - Optional Text

▼ Section: Package manager

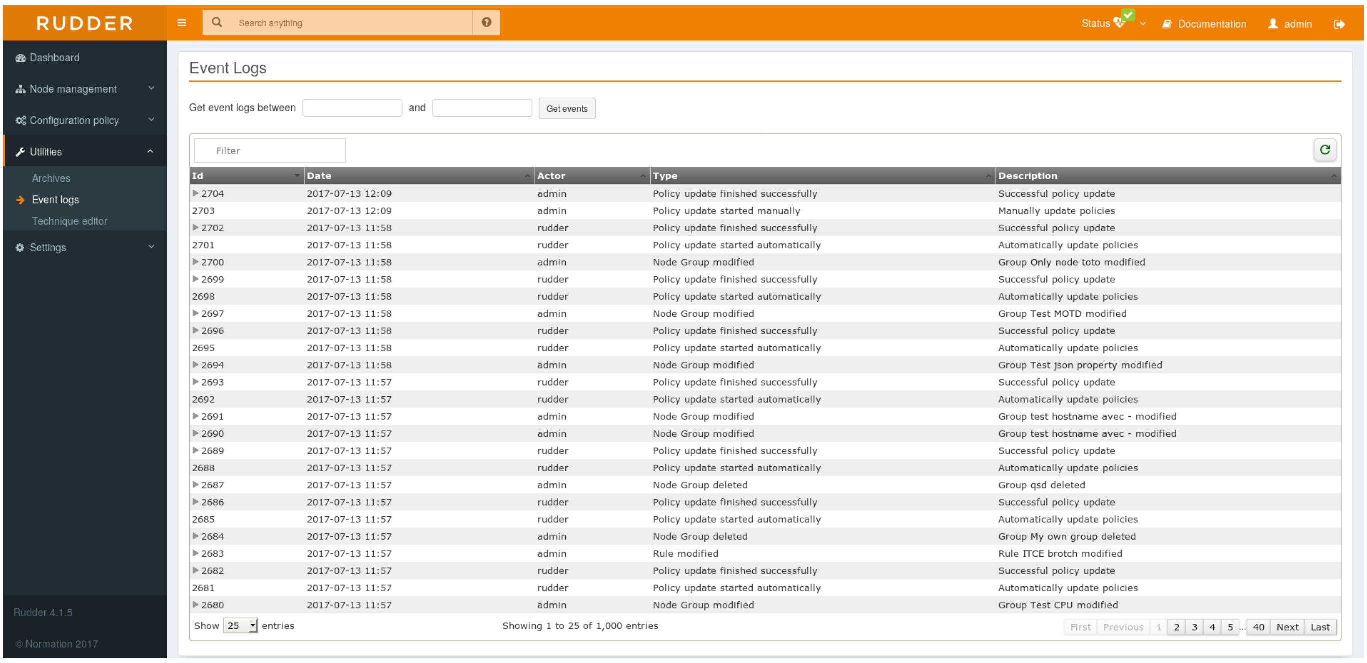
Package manager Default package manager

▼ Section: Post-modification script

Figure 5.7: Directive list

5.2.4 Utilities

This section contains tools useful for your everyday usage of *Rudder*. This is where you will find the technique editor, the event logs table or the change requests if you have enabled that feature.



Event Logs

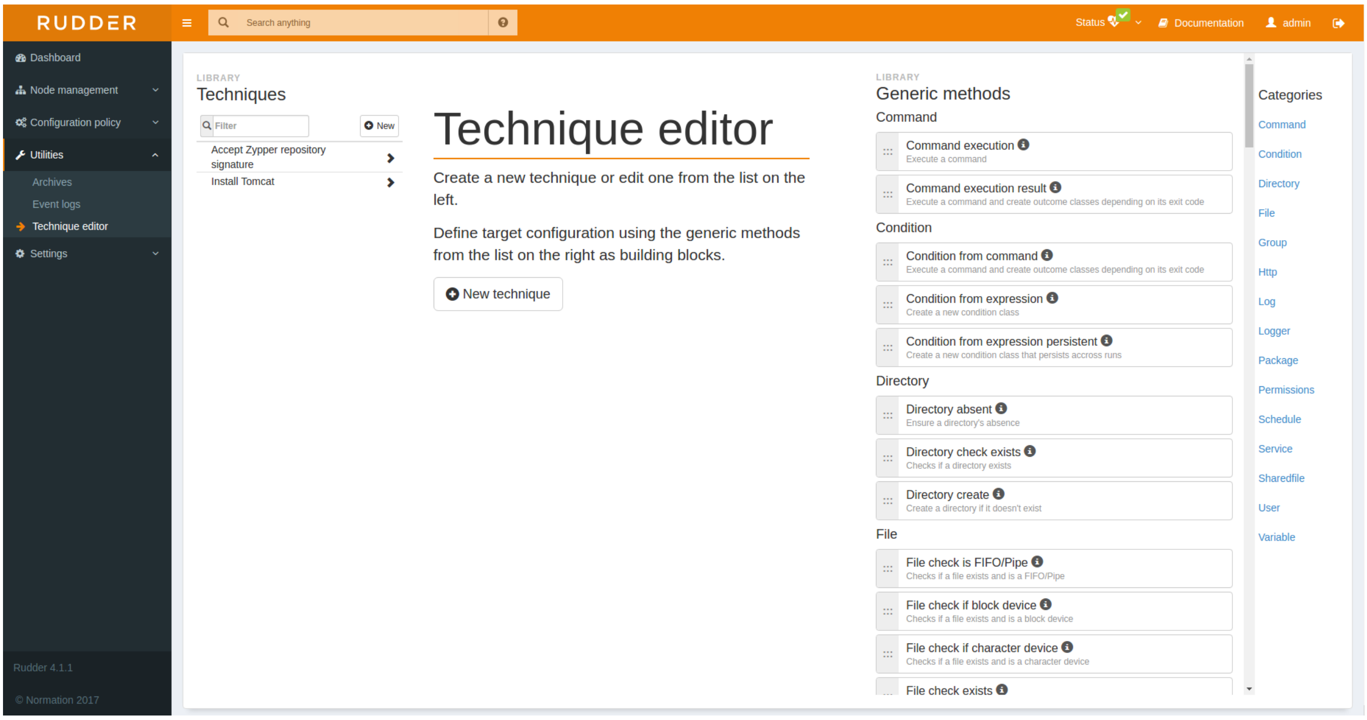
Get event logs between and

Id	Date	Actor	Type	Description
▶ 2704	2017-07-13 12:09	admin	Policy update finished successfully	Successful policy update
2703	2017-07-13 12:09	admin	Policy update started manually	Manually update policies
▶ 2702	2017-07-13 11:58	rudder	Policy update finished successfully	Successful policy update
2701	2017-07-13 11:58	rudder	Policy update started automatically	Automatically update policies
▶ 2700	2017-07-13 11:58	admin	Node Group modified	Group Only node toto modified
▶ 2699	2017-07-13 11:58	rudder	Policy update finished successfully	Successful policy update
2698	2017-07-13 11:58	rudder	Policy update started automatically	Automatically update policies
▶ 2697	2017-07-13 11:58	admin	Node Group modified	Group Test MOTD modified
▶ 2696	2017-07-13 11:58	rudder	Policy update finished successfully	Successful policy update
2695	2017-07-13 11:58	rudder	Policy update started automatically	Automatically update policies
▶ 2694	2017-07-13 11:58	admin	Node Group modified	Group Test json property modified
▶ 2693	2017-07-13 11:57	rudder	Policy update finished successfully	Successful policy update
2692	2017-07-13 11:57	rudder	Policy update started automatically	Automatically update policies
▶ 2691	2017-07-13 11:57	admin	Node Group modified	Group test hostname avec - modified
▶ 2690	2017-07-13 11:57	admin	Node Group modified	Group test hostname avec - modified
▶ 2689	2017-07-13 11:57	rudder	Policy update finished successfully	Successful policy update
2688	2017-07-13 11:57	rudder	Policy update started automatically	Automatically update policies
▶ 2687	2017-07-13 11:57	admin	Node Group deleted	Group qsd deleted
▶ 2686	2017-07-13 11:57	rudder	Policy update finished successfully	Successful policy update
2685	2017-07-13 11:57	rudder	Policy update started automatically	Automatically update policies
▶ 2684	2017-07-13 11:57	admin	Node Group deleted	Group My own group deleted
▶ 2683	2017-07-13 11:57	admin	Rule modified	Rule ITCE brotch modified
▶ 2682	2017-07-13 11:57	rudder	Policy update finished successfully	Successful policy update
2681	2017-07-13 11:57	rudder	Policy update started automatically	Automatically update policies
▶ 2680	2017-07-13 11:57	admin	Node Group modified	Group Test CPU modified

Show 25 entries

Showing 1 to 25 of 1,000 entries

Figure 5.8: Event Logs



Technique editor

Create a new technique or edit one from the list on the left.

Define target configuration using the generic methods from the list on the right as building blocks.

Command

- Command execution — Execute a command
- Command execution result — Execute a command and create outcome classes depending on its exit code

Condition

- Condition from command — Execute a command and create outcome classes depending on its exit code
- Condition from expression — Create a new condition class
- Condition from expression persistent — Create a new condition class that persists accross runs

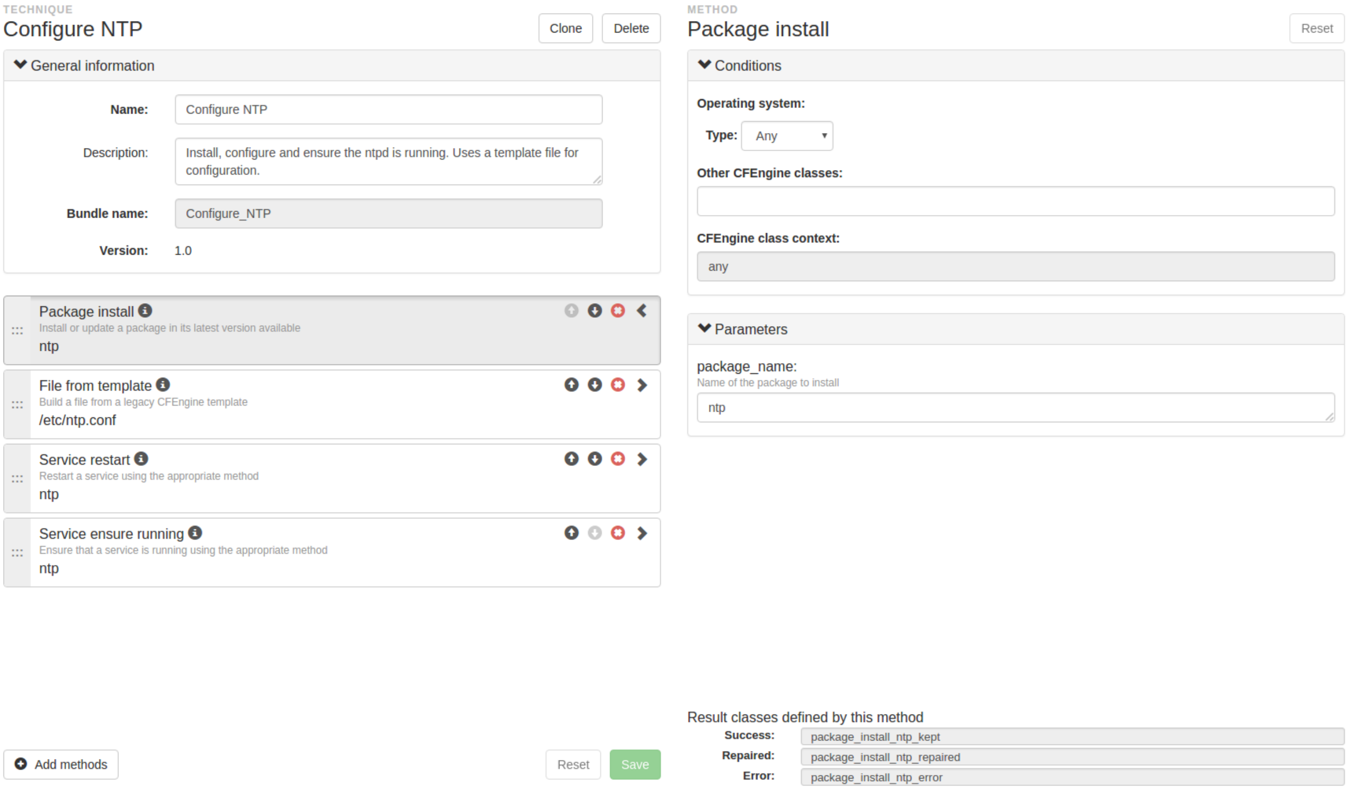
Directory

- Directory absent — Ensure a directory's absence
- Directory check exists — Checks if a directory exists
- Directory create — Create a directory if it doesn't exist

File

- File check is FIFO/Pipe — Checks if a file exists and is a FIFO/Pipe
- File check if block device — Checks if a file exists and is a block device
- File check if character device — Checks if a file exists and is a character device
- File check exists

Figure 5.9: Technique Editor



Configure NTP

Name: Configure NTP

Description: Install, configure and ensure the ntpd is running. Uses a template file for configuration.

Bundle name: Configure_NTP

Version: 1.0

- Package install — Install or update a package in its latest version available — ntp
- File from template — Build a file from a legacy CFEngine template — /etc/ntp.conf
- Service restart — Restart a service using the appropriate method — ntp
- Service ensure running — Ensure that a service is running using the appropriate method — ntp

Package install

Operating system:

Type: Any

Other CFEngine classes:

CFEngine class context: any

package_name: Name of the package to install — ntp

Result classes defined by this method

Success: package_install_ntp_kept

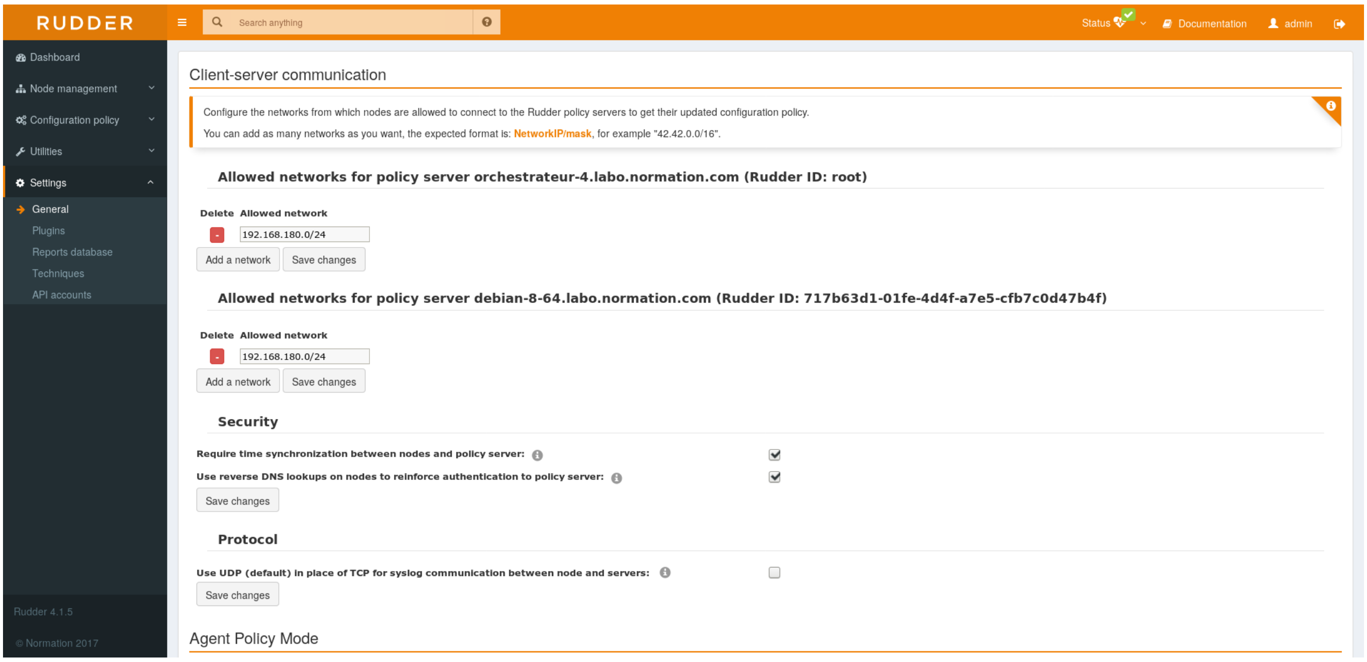
Repaired: package_install_ntp_repaired

Error: package_install_ntp_error

Figure 5.10: Technique details

5.2.5 Settings

The Settings section provides you a way to modify your *Rudder* setup: you can setup the available networks for the Policy Server, configure agent run and policy mode, enable web interface options and manage installed plugins.



Client-server communication

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

Allowed networks for policy server orchestrateur-4.labo.normation.com (Rudder ID: root)

Delete	Allowed network
-	192.168.180.0/24

Add a network Save changes

Allowed networks for policy server debian-8-64.labo.normation.com (Rudder ID: 717b63d1-01fe-4d4f-a7e5-cfb7c0d47b4f)

Delete	Allowed network
-	192.168.180.0/24

Add a network Save changes

Security

Require time synchronization between nodes and policy server: ☑

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

Save changes

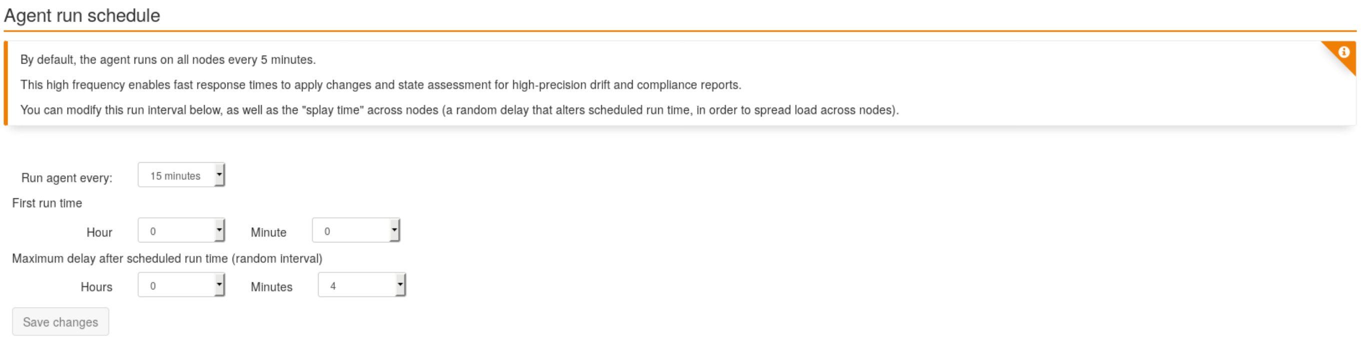
Protocol

Use UDP (default) in place of TCP for syslog communication between node and servers: ☐

Save changes

Agent Policy Mode

Figure 5.11: Settings screen



Agent run schedule

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

Run agent every: 15 minutes

First run time

Hour 0 Minute 0

Maximum delay after scheduled run time (random interval)

Hours 0 Minutes 4

Save changes

Figure 5.12: Changing global agent run

5.3 Units supported as search parameters

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

5.3.1 Bytes and multiples

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

5.3.2 Convenience notation

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

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

The following number is not valid:

```
_1234
```

5.3.3 Supported units

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

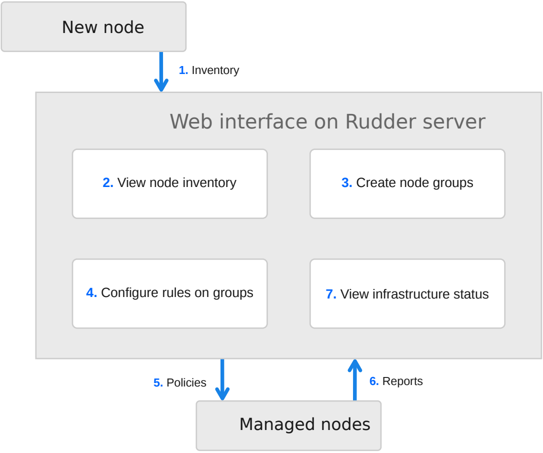
In detail, the following units are supported (provided in lower case, see above):

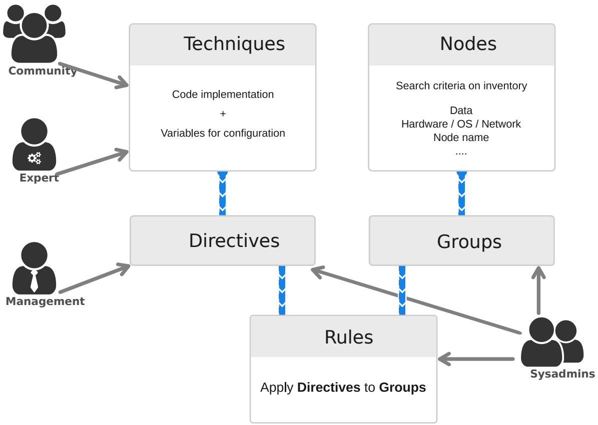
Notation	Alternate	Value
`b`	`o`	bytes (equivalent to not specifying a unit)
`kb`	`ko`	1024 bytes
`mb`	`mo`	1024^2 bytes
`gb`	`go`	1024^3 bytes
`tb`	`to`	1024^4 bytes
`pb`	`po`	1024^5 bytes
`eb`	`eo`	1024^6 bytes
`zb`	`zo`	1024^7 bytes
`yb`	`yo`	1024^8 bytes

Table 5.1: Units supported by Rudder search engine

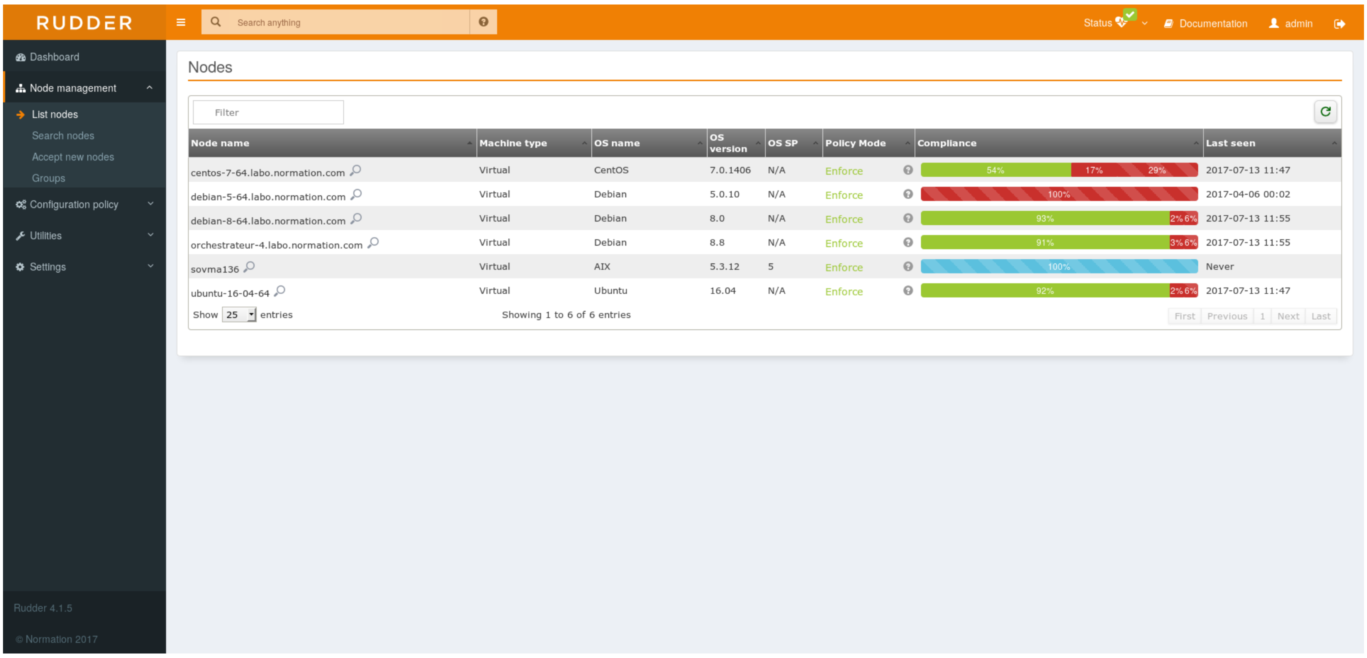
Chapter 6

Node management





6.1 Node inventory



Nodes

Filter

Node name	Machine type	OS name	OS version	OS SP	Policy Mode	Compliance	Last seen
centos-7-64.labo.normation.com	Virtual	CentOS	7.0.1406	N/A	Enforce	54% 17% 29%	2017-07-13 11:47
debian-5-64.labo.normation.com	Virtual	Debian	5.0.10	N/A	Enforce	100%	2017-04-06 00:02
debian-8-64.labo.normation.com	Virtual	Debian	8.0	N/A	Enforce	93% 2% 6%	2017-07-13 11:55
orchestrateur-4.labo.normation.com	Virtual	Debian	8.8	N/A	Enforce	91% 3% 6%	2017-07-13 11:55
sovma136	Virtual	AIX	5.3.12	5	Enforce	100%	Never
ubuntu-16-04-64	Virtual	Ubuntu	16.04	N/A	Enforce	92% 2% 6%	2017-07-13 11:47

Show 25 entries

Showing 1 to 6 of 6 entries

First Previous 1 Next Last

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

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

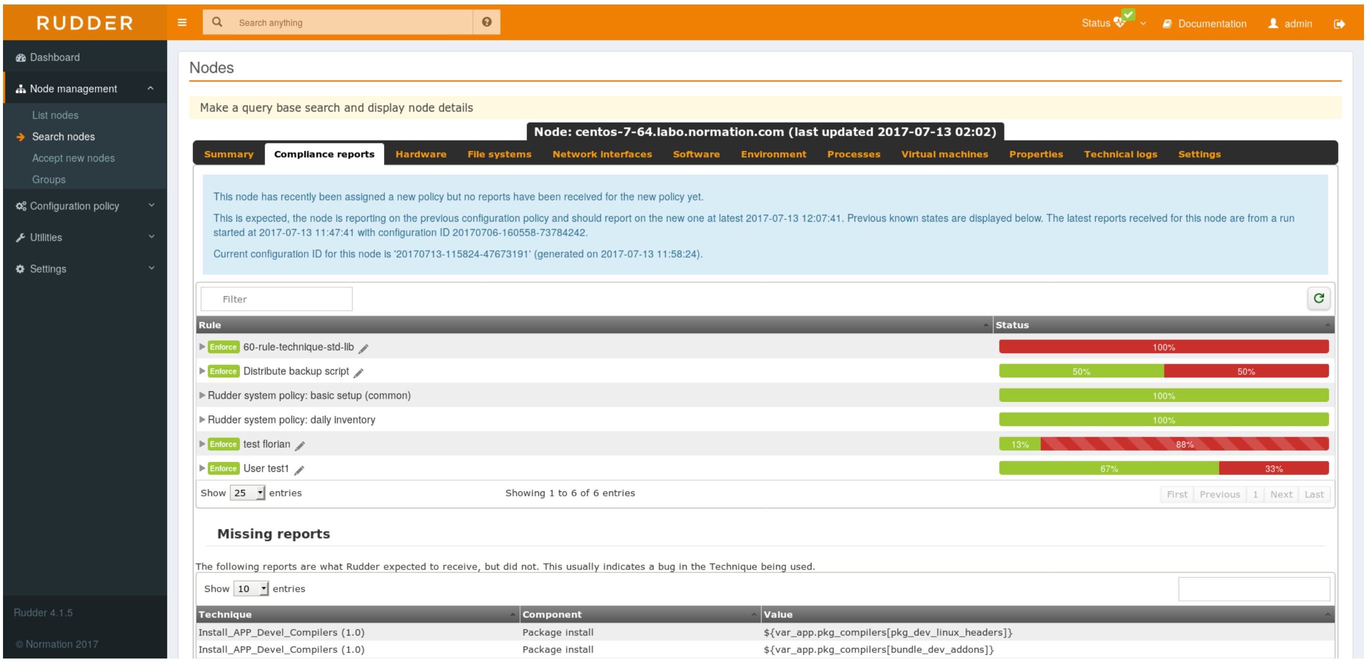
The Node *Summary* presents administrative information like the *Node Hostname*, *Operating System*, Rudder *Client name*, Rudder *ID* and *Date* when the inventory was *last received*. When the *Node* has been validated, some more information is displayed like the *Node Name* and the *Date first accepted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

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

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

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

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



Make a query base search and display node details

Node: centos-7-64.labo.normation.com (last updated 2017-07-13 02:02)

This node has recently been assigned a new policy but no reports have been received for the new policy yet.

This is expected, the node is reporting on the previous configuration policy and should report on the new one at latest 2017-07-13 12:07:41. Previous known states are displayed below. The latest reports received for this node are from a run started at 2017-07-13 11:47:41 with configuration ID 20170706-160558-73784242.

Current configuration ID for this node is '20170713-115824-47673191' (generated on 2017-07-13 11:58:24).

Rule	Status
Enforce 60-rule-technique-std-lib	100%
Enforce Distribute backup script	50% 50%
Rudder system policy: basic setup (common)	100%
Rudder system policy: daily inventory	100%
Enforce test florian	13% 88%
Enforce User test1	67% 33%

Showing 1 to 6 of 6 entries

Missing reports

The following reports are what Rudder expected to receive, but did not. This usually indicates a bug in the Technique being used.

Technique	Component	Value
Install_APP_Devel_Compilers (1.0)	Package install	${var_app.pkg_compilers[pkg_dev_linux_headers]}
Install_APP_Devel_Compilers (1.0)	Package install	${var_app.pkg_compilers[bundle_dev_addons]}

6.2 Accept new Nodes

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

Example 6.1 Accept the new Node `debian-node.rudder-project.org`

1. Install and configure the *Rudder* Agent on the new *Node* `debian-node.rudder-project.org`

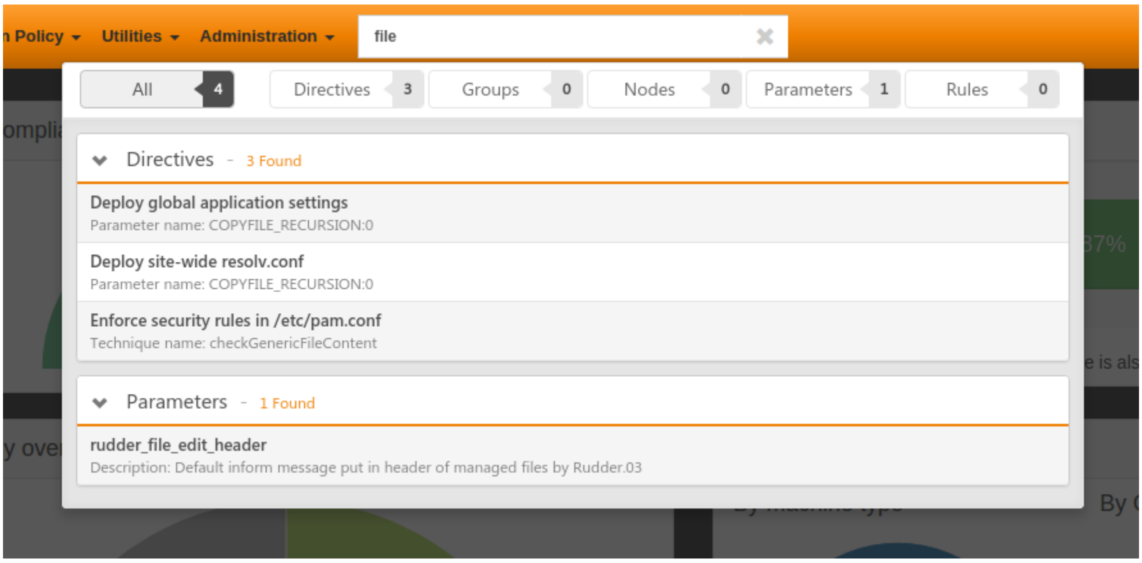
2. Wait a few minutes for the first run of the *Rudder* Agent.
3. Navigate to ***Node* Management > Accept new *Nodes***.
4. Select the new *Node* in the list.
5. Validate the *Node*.
6. The *Node* is now integrated in *Rudder*, you can search it using the search tools.

6.3 Search Nodes

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

6.3.1 Quick search

You might have noticed the small text area at the top of the *Rudder* interface: it is the Quick Search bar. Its purpose is to enable a user to easily search for *Rudder* elements (*Nodes*, *Groups*, *Directives*, *Parameters*, *Rules*) based on their name, id, description, inventory…



Directives - 3 Found

Deploy global application settings
Parameter name: COPYFILE_RECURSION:0

Deploy site-wide resolv.conf
Parameter name: COPYFILE_RECURSION:0

Enforce security rules in /etc/pam.conf
Technique name: checkGenericFileContent

Parameters - 1 Found

rudder_file_edit_header
Description: Default inform message put in header of managed files by Rudder.03

An autocompletion list will appear as soon as *Rudder* detects an element it can identify, you just have to click on it to be redirected to the element configuration page.

More complex search queries can be input using the "in:" and "is:" keywords, "is" targets *Rudder* objects by type, and "in" targets elements like name, description…

Those keywords are used to refine a research in case a search query returns too many results.

The available search keywords are:

Example 6.2 Example: Search for a Node called `debian-node`

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

1. Type in the Quick Search field the `de` or `d0`.
2. The search result will return this *Node*: `debian-node.rudder-project.org -- d06b1c6c-f59b-4e5e-8049-d55f769ac33f [d06b1c6c-f59b-4e5e-8049-d55f769ac33f]`.

keyword	**Description**
node	*Nodes*
group	*Groups*
parameter	*Parameters*
directive	*Directives*
rule	*Rules*

Table 6.1: is: keywords

keyword	**Search for**
name	Names
id	IDs
description, long_description	Descriptions
enabled	Enabled elements (true or false)

Table 6.2: in: keywords (common)

keyword	**Search for**
hostname	Hostnames
os_type	OS types (windows, linux, aix. . .)
os_name	OS Names (*Windows* Server 2012, *Debian*. . .)
os_version	OS versions (8, 2008 R2, . . .)
os	OS Full Names (*Debian* GNU/Linux 6.0.10 (squeeze). . .)
os_kernel_version	OS Kernel versions (3.16, 5.1. . .)
os_service_pack	OS Service Packs (for *Windows* and *SuSE* Linux)
architecture	OS Architectures (amd64, x86_64, i386)
ram	Machine memory
ips	Network IP addresses
policy_server_id	ID of a node’s policy server (root. . .)
properties	*Node* properties (arbitrary key=values associated to a node)
rudder_roles	*Rudder* roles (rudder-reports, rudder-ldap. . .)

Table 6.3: in: keywords (nodes)

keyword	**Search for**
dynamic	Dynamic groups

Table 6.4: in: keywords (groups)

keyword	**Search for**
dir_param_name	*Directive* parameter names, as in the *Techniques* metadata.xml ("GENERIC_FILE_CONTENT_PATH". . .)
dir_param_value	*Directive* parameter values
technique_id	*Technique* IDs
technique_name	*Technique* names ("Enforce a file content". . .)
technique_version	*Technique* version

Table 6.5: in: keywords (directives)

keyword	**Search for**
parameter_name	*Parameter* names
parameter_value	*Parameter* values

Table 6.6: in: keywords (parameters)

keyword	Search for
directives	*Rules* containing those *Directive* IDs
groups	*Rules* containing those *Group* IDs

Table 6.7: in: keywords (rules)

Example 6.3 Example: Search for a directive called `Common users`

Assuming you have one *Directive* called `Common users`, whose ID in *Rudder* is `6e8ce05b-3f77-4fed-a424-edf0fdaa4231`.

1. Type in the Quick Search field `is:directive common`.
2. The search result will return this *Directive*: `Common users -- 4a6aaea7-6471-4ca9-8c27-9ee9f44ed882 [6e8ce05b-3f77-4fed-a424-edf0fdaa4231]`.

Tip

This feature is enabled by default on `new` installations of *Rudder* from the following versions:

- 3.1.14
- 3.2.7
- 4.0.0

and may be enabled after an upgrade of a *Rudder* installation in the Settings tab.

6.3.2 Advanced Search

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

- 1. Select a field

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

- 2. Select the matching rule

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

- a. Regex matching rule

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

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

Example 6.4 Search node having an ip address matching `192.168.x.y`

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

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

- b. Composite search

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

Example 6.5 Search Environment Variable `LANG=C`.

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

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

- 3. Add another rule

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

Example 6.6 Advanced search for Linux Nodes with `ssh`.

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

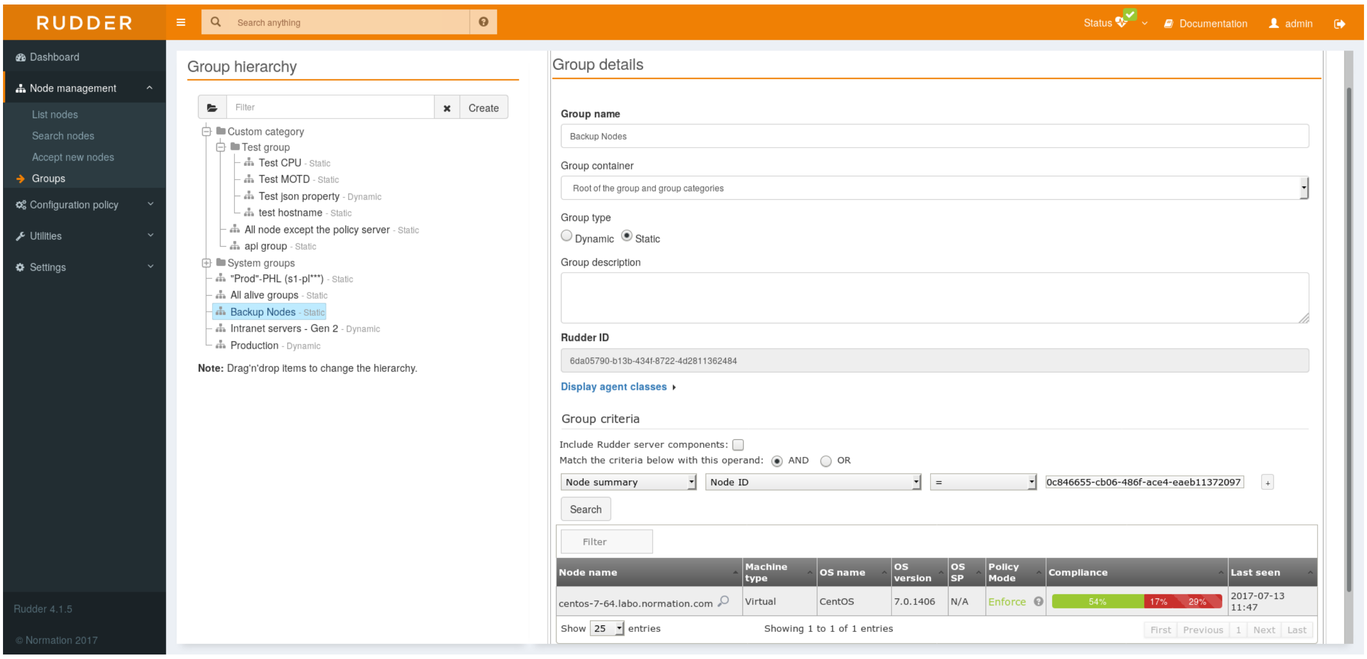
1. Operator: `AND`.
2. First search line: Node, *Operating System*, =, *Linux*.
3. Second search line: *Software*, *Name*, =, `ssh`.

6.4 Group of Nodes

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

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

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



Group hierarchy

- Custom category
  - Test group
    - Test CPU - Static
    - Test MOTD - Static
    - Test json property - Dynamic
    - test hostname - Static
  - All node except the policy server - Static
  - api group - Static
- System groups
- "Prod"-PHL (s1-pl***) - Static
- All alive groups - Static
- Backup Nodes - Static
- Intranet servers - Gen 2 - Dynamic
- Production - Dynamic

Note: Drag'n'drop items to change the hierarchy.

Group details

Group name: Backup Nodes

Group container: Root of the group and group categories

Group type: ○ Dynamic ● Static

Group description

Rudder ID: 6da05790-b13b-434f-8722-4d2811362484

Group criteria

Include Rudder server components: □

Match the criteria below with this operand: ● AND ○ OR

Node summary | Node ID | = | 0c846655-cb06-486f-ace4-eaeb11372097

Node name	Machine type	OS name	OS version	OS SP	Policy Mode	Compliance	Last seen
centos-7-64.labo.normation.com	Virtual	CentOS	7.0.1406	N/A	Enforce	54% 17% 29%	2017-07-13 11:47

Show 25 entries Showing 1 to 1 of 1 entries First Previous 1 Next Last

Example 6.7 Create a dynamic group for Linux Nodes with `ssh` having an ip address in 192.18.42.x.

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

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

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

Chapter 7

Configuration concepts

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

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

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

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

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

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

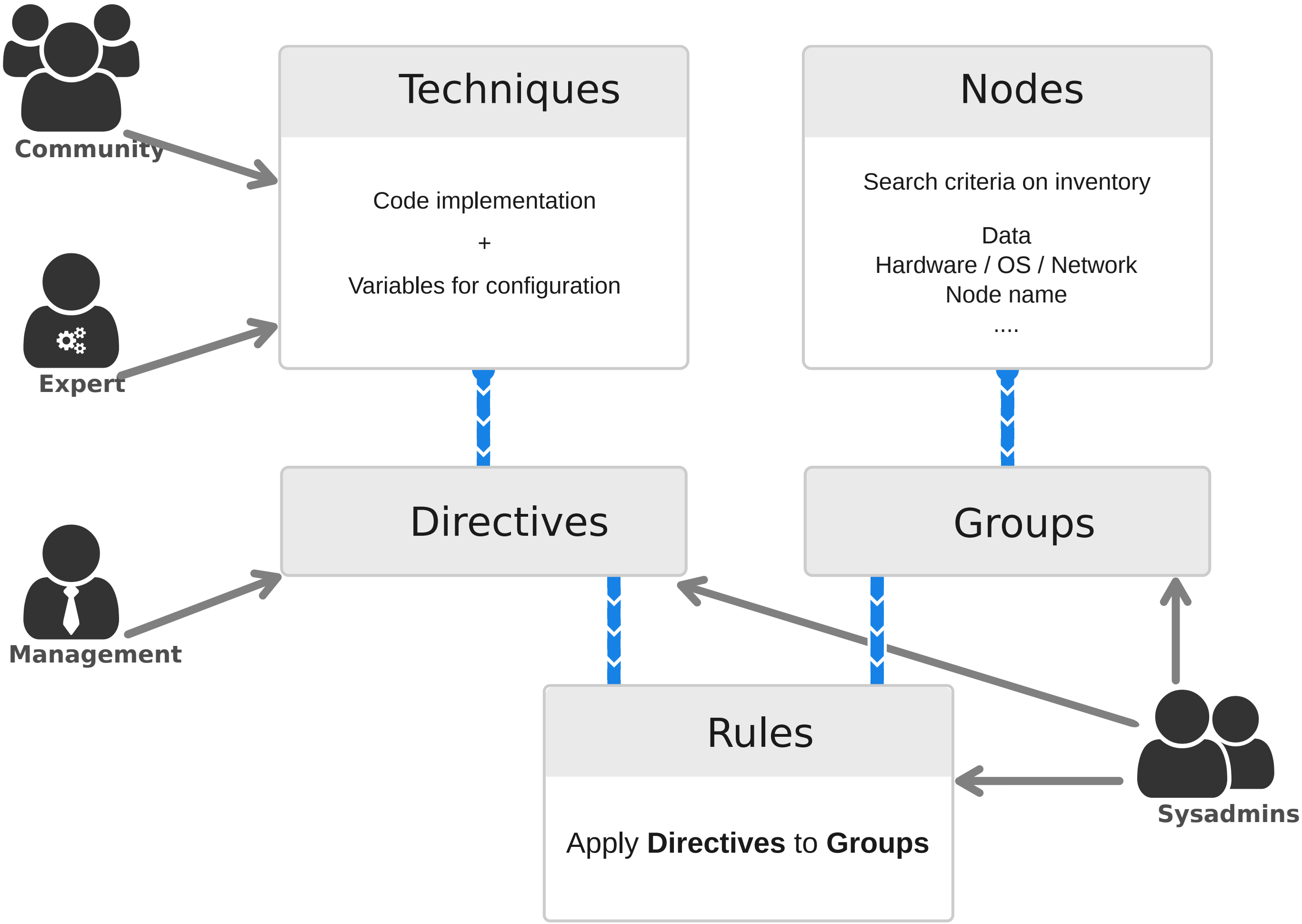


Figure 7.1: Concepts diagram

7.1 Techniques

7.1.1 Concepts

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

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

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

7.1.2 Manage the Techniques

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

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

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

7.1.3 Create new Techniques

The standard library only provides the most common *Techniques*. You can create new *Technique* with the Technique Editor.

7.2 Directives

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

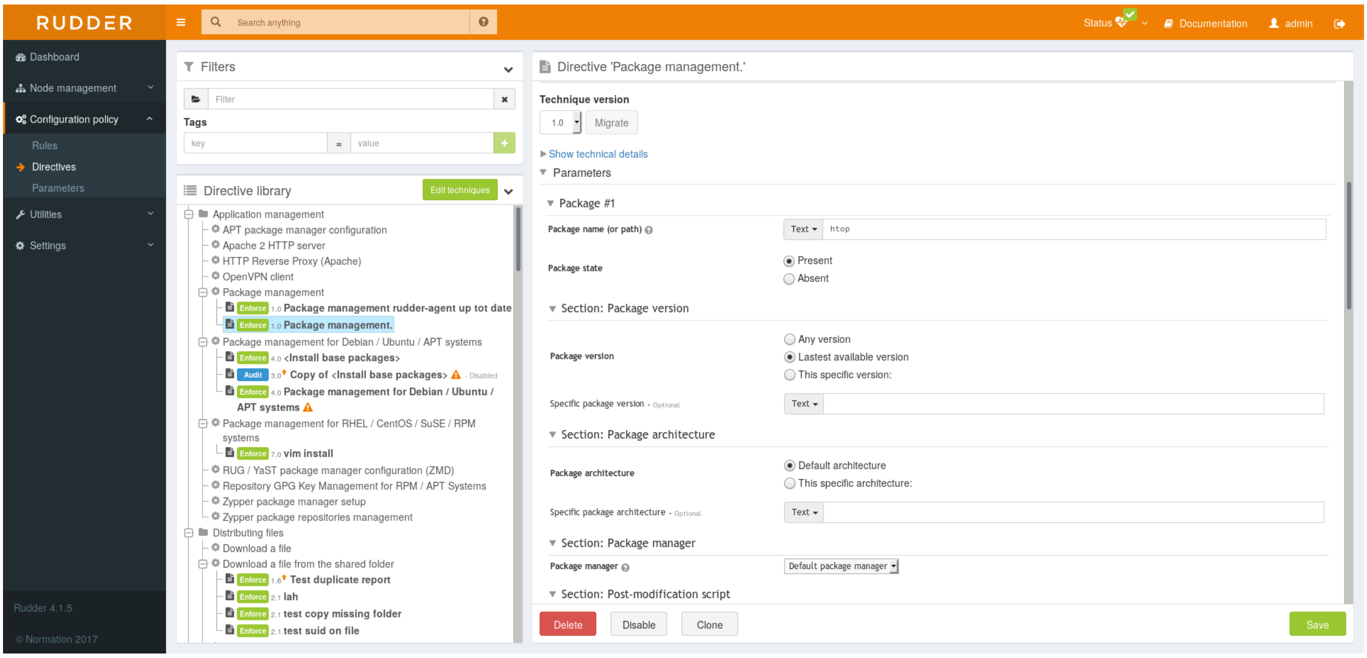
Directive This is an instance of a *Technique*, which allows to set values for the parameters of the latter. Each *Directive* can have a unique name. A *Directive* should be completed with a short and a long description, and a collection of parameters for the variables defined by the *Technique*.

The screen is divided in three parts:

- on the left, The list of *Directives*, grouped by *Technique*
- on the right, The selected *Directive* form.

Click on the name of a *Technique* to show its description, and how to Create a *Directive* base on it.

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*.



Directive library:

- Application management
  - APT package manager configuration
  - Apache 2 HTTP server
  - HTTP Reverse Proxy (Apache)
  - OpenVPN client
  - Package management
    - Enforce 1.0 **Package management rudder-agent up tot date**
    - Enforce 1.0 **Package management.**
  - Package management for Debian / Ubuntu / APT systems
    - Enforce 4.0 **<Install base packages>**
    - Audit 3.0 **Copy of <Install base packages>** ⚠ - Disabled
    - Enforce 4.0 **Package management for Debian / Ubuntu / APT systems** ⚠
  - Package management for RHEL / CentOS / SuSE / RPM systems
    - Enforce 7.0 **vim install**
  - RUG / YaST package manager configuration (ZMD)
  - Repository GPG Key Management for RPM / APT Systems
  - Zypper package manager setup
  - Zypper package repositories management
- Distributing files
  - Download a file
  - Download a file from the shared folder
    - Enforce 1.6 **Test duplicate report**
    - Enforce 2.1 **lah**
    - Enforce 2.1 **test copy missing folder**
    - Enforce 2.1 **test suid on file**

Directive 'Package management.'

Technique version
1.0 Migrate

▶ Show technical details

▼ Parameters

▼ Package #1

Package name (or path) Text htop

Package state ● Present ○ Absent

▼ Section: Package version

Package version ○ Any version ● Lastest available version ○ This specific version:

Specific package version - Optional Text

▼ Section: Package architecture

Package architecture ● Default architecture ○ This specific architecture:

Specific package architecture - Optional Text

▼ Section: Package manager

Package manager Default package manager

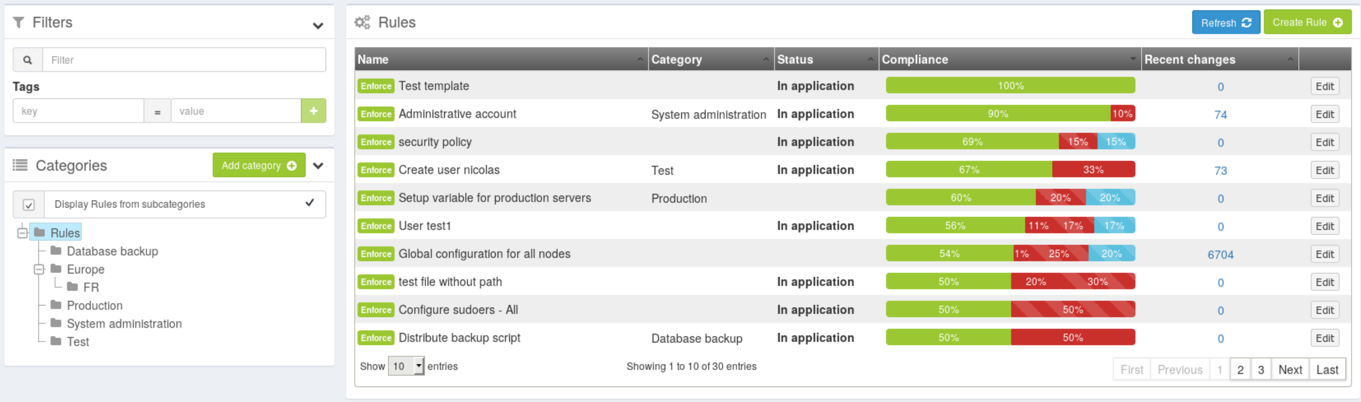
▼ Section: Post-modification script

Example 7.1 Create a Directive for Name resolution

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

7.3 Rules

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



☑ Display Rules from subcategories

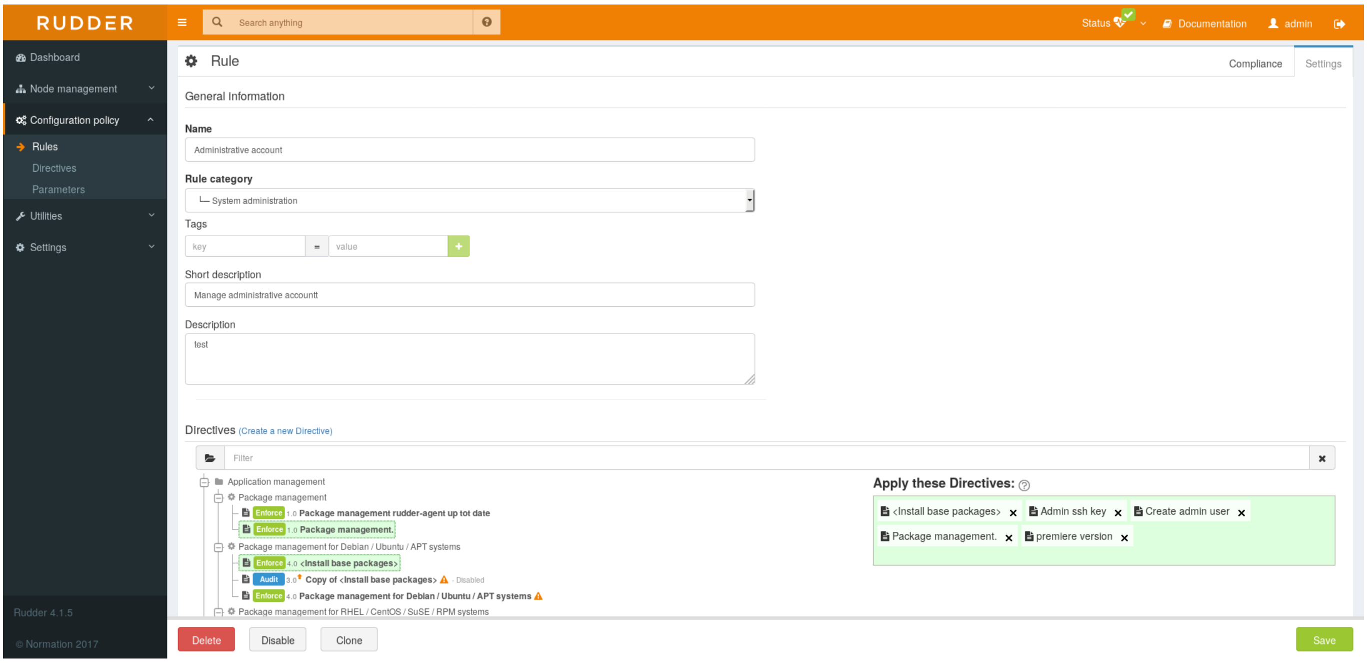
- Rules
  - Database backup
  - Europe
    - FR
  - Production
  - System administration
  - Test

Rules

Name	Category	Status	Compliance	Recent changes	
Enforce Test template		In application	100%	0	Edit
Enforce Administrative account	System administration	In application	90% 10%	74	Edit
Enforce security policy		In application	69% 15% 15%	0	Edit
Enforce Create user nicolas	Test	In application	67% 33%	73	Edit
Enforce Setup variable for production servers	Production		60% 20% 20%	0	Edit
Enforce User test1		In application	56% 11% 17% 17%	0	Edit
Enforce Global configuration for all nodes			54% 1% 25% 20%	6704	Edit
Enforce test file without path		In application	50% 20% 30%	0	Edit
Enforce Configure sudoers - All		In application	50% 50%	0	Edit
Enforce Distribute backup script	Database backup	In application	50% 50%	0	Edit

Show 10 entries Showing 1 to 10 of 30 entries First Previous 1 2 3 Next Last

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 status icon on the top of the page turns to grey, with moving arrows. if you feel the generated promises should be modified (for instance, if you changed the configuration of *Rudder*), you can click on the status menu in the top bar and click on "Regenerate policies"



Rule

General information

Name: Administrative account

Rule category: └ System administration

Tags: key = value

Short description: Manage administrative accountt

Description: test

Directives (Create a new Directive)

- Application management
  - Package management
    - Enforce 1.0 Package management rudder-agent up tot date
    - Enforce 1.0 Package management.
  - Package management for Debian / Ubuntu / APT systems
    - Enforce 4.0 <Install base packages>
    - Audit 3.0 Copy of <Install base packages> ⚠ - Disabled
    - Enforce 4.0 Package management for Debian / Ubuntu / APT systems ⚠
  - Package management for RHEL / CentOS / SuSE / RPM systems

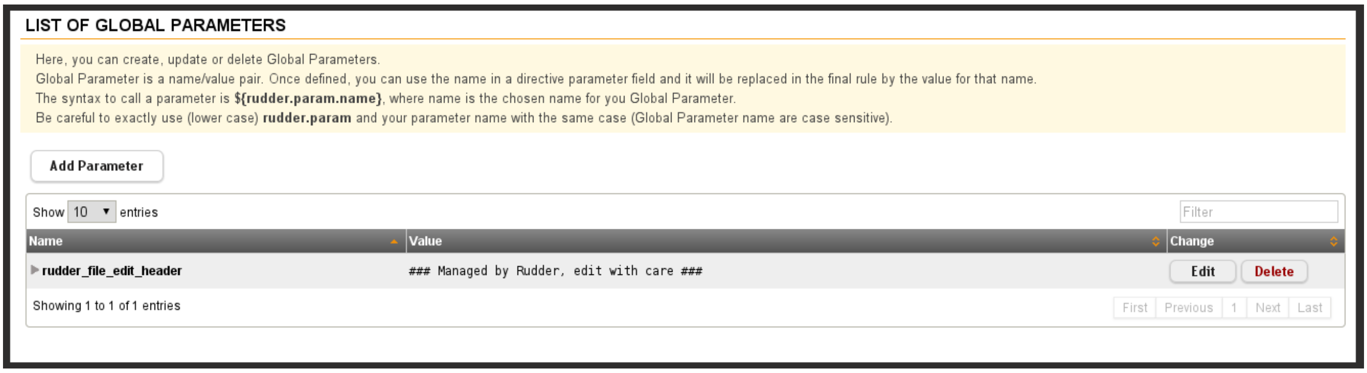
Apply these Directives:

<Install base packages> ✕ Admin ssh key ✕ Create admin user ✕ Package management. ✕ premiere version ✕

7.4 Variables

7.4.1 User defined parameters

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



LIST OF GLOBAL PARAMETERS

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

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

Showing 1 to 1 of 1 entries

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

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

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

7.4.2 System variables

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

The information about a *Node*:

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

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

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

7.5 Compliance and Drift Assessment

7.5.1 Overview in Rudder

Rudder is built to continuously assess drift compared to defined policies, with or without auto-healing.

By auto-healing, we mean that optionally, *Rudder* can continuously enforce correct configuration over time, correcting the assessed drift so that your configuration converges towards desired states. This behavior is optionnal, and *Rudder* can only report drift without changing configuration. That policy enforce or audit mode can be configured by node, rule or directive (see Section 7.7 for more details).

Rudder is able to adapt to complex process and only do the minimal required work so that the server converges to the desired state, and so whatever was the starting state point. *Rudder* works as a GPS would, adapting the path to your destination depending of the path you actually took. This process is much more resilient to changes than a step by step, procedural description of the commands to execute.

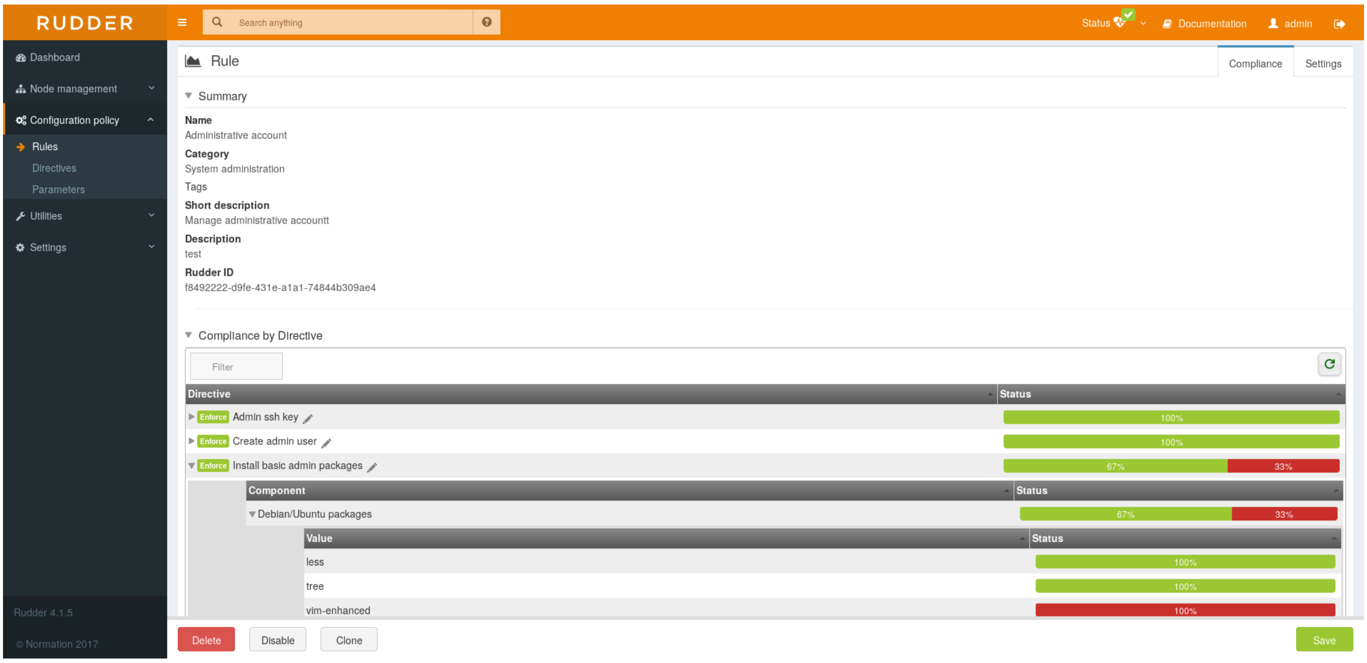
Compliance and drift from expected configurations are then reported with possibility to drill down in non-compliance issues to identify the root problem.

Of course, one can always correct a drift error by hand by updating coniguration target and changing policy mode from "audit" to "enforce" mode.

7.5.1.1 Compliance and drift reporting

Compliance drifts (non compliances, enforcement errors, repaires) are reported in *Rudder* by several means:

- Compliance are reported in aggregated format globally in the dashboard, and by rules or nodes (example for *Rule* below)
- they are stored in *Rudder* compliance database, and each *Rule* displays an history of changes as depicted in "Changes history on a *Rule*" below.
- each drifts fires an event which is logged in file /var/log/rudder/compliance/non-compliant-reports.log and can be used to integrates with log aggregation engine like Logstash, or hooks (typically to send notification to IRC or Slack, send email, etc)
- see for example the Slack connector here: https://github.com/*Normation*/rudder-tools/blob/master/scripts/rudder-notification/forward-non-compliance-to-slack.sh
- compliance and drift are also available from *Rudder* API to provide deeper integration with your IT Infrastructure.



Rule

Summary

Name
Administrative account

Category
System administration

Tags

Short description
Manage administrative accountt

Description
test

Rudder ID
f8492222-d9fe-431e-a1a1-74844b309ae4

Compliance by Directive

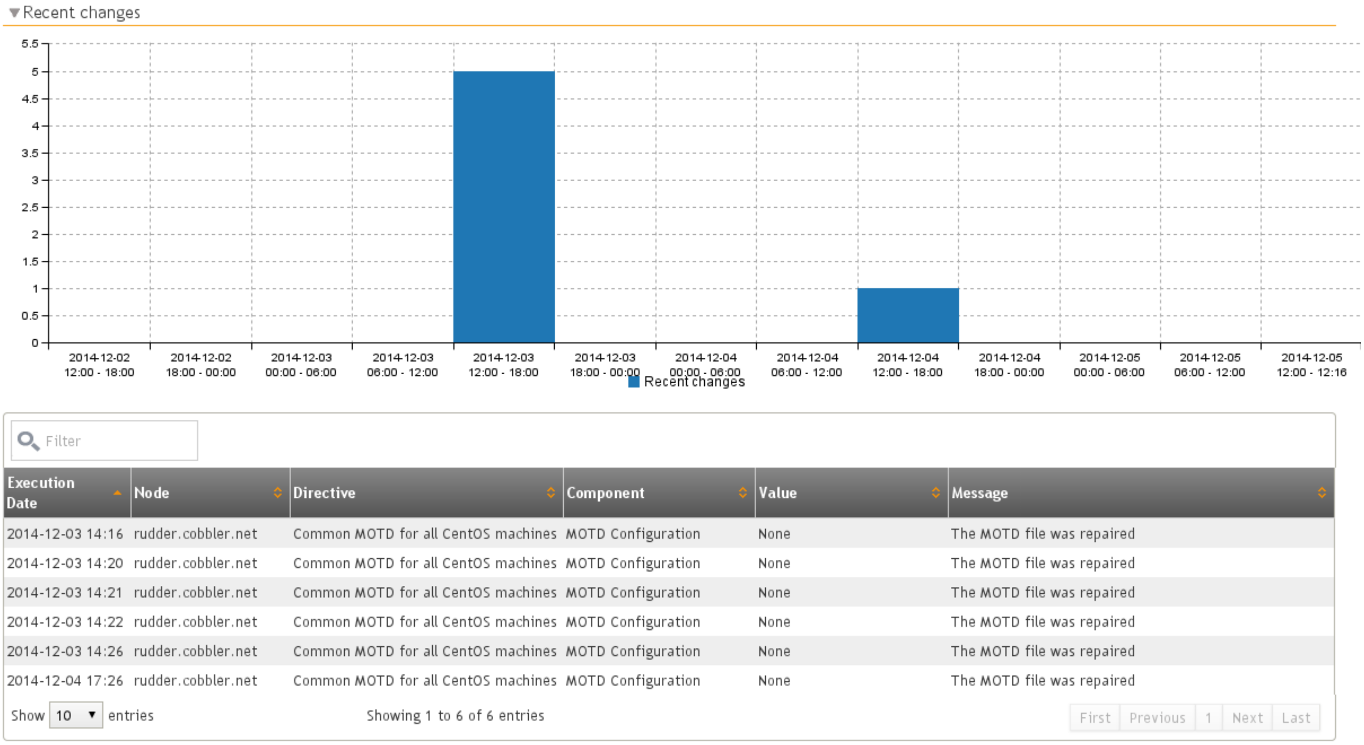
Directive	Status
▶ Enforce Admin ssh key	100%
▶ Enforce Create admin user	100%
▼ Enforce Install basic admin packages	67% 33%

Component	Status
▼ Debian/Ubuntu packages	67% 33%

Value	Status
less	100%
tree	100%
vim-enhanced	100%

Figure 7.2: Compliance on a Rule

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



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

Figure 7.3: Changes history on a Rule

7.5.2 How compliance is calculated ?

As previously seen, in *Rudder* you define *Rules* which target groups of *Nodes*, and are composed of configuration *Directives*.

A *Directive* contains one or multiple sub-configuration elements which generates reports. For example, for a Sudoers *Directive*, each user can be such an element.

Reports have states explaining what is the drift between the expected configuration and the actual configuration. Some states depends if the user choose to auto-matically enforce drift correction or if he chose to only reports on drift).

Finaly, a node can get a global state if reports don't come at expected frequency or for expected policy configuration version.

Below you will find all details about the possible states and their meaning with the actual compliance calculus method.

Checking that the node is correctly reporting, at correct frequency

At the node level, we are checking that the node is sending reports according to the expected frequency, and for the currently defined version of the configuration for it.

Based on this information, we get a

Applying When a new set of policies are defined for a node (or any update to existing one), *Rudder* waits during a grace period for reports so that the node has time to apply the new policies. During this period, the configuration is said *Applying*.

No report The system didn't send any reports since a time incompatible with the agent frequency run interval. Most likely, the node is not online or there is an ongoing network issue between the node and *Rudder* server.

At directive level: checking for drift and auto-healing

Success or Compliant The system is already in the desired state. No change is needed. Conformity is reached.

Repaired When a configuration policy is "enforced", that state means that 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.

Error When configuration is enforced, it means that the system is not in the desired state and *Rudder* wasn't able to repair the system.

Non compliant When configuration is not enforced, it means that the systemn is not in the desired state. A drift is reported.

Not applicable A specific configuration may not be applicable on a given node because some precondition are not met. For example, the specified configuration is only relevant for Linux nodes, and thus is Not applicable on a *Windows* server.

Unexpected We have a special kind of report for unexpected states (both for enforce and audit mode). These reports generally mean that the node is sending reports for unexpected configuration components. It may be due to bad parameters for the configuration, or an error in the *Technique*.

Compliance calculus

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

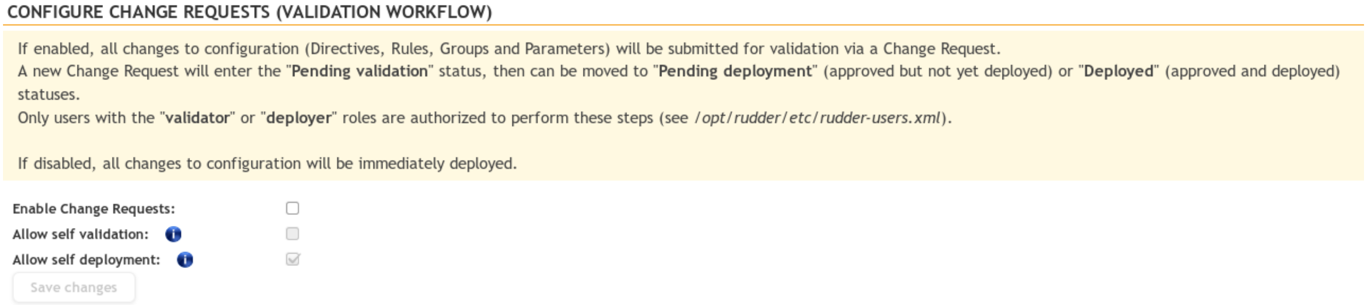
Number of *Nodes* for which conformity is reached for every *Directive* of the *Rule* / Total number of *Nodes* on which the *Rule* has been applied

7.6 Validation workflow in Rudder

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

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

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



CONFIGURE CHANGE REQUESTS (VALIDATION WORKFLOW)

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

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

Enable Change Requests: □
Allow self validation: □
Allow self deployment: ☑

7.6.1 What is a Change request ?

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

A Change request has:

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

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

7.6.1.1 Change request status

There is 4 Change request status:

Pending validation

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

Pending deployment

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

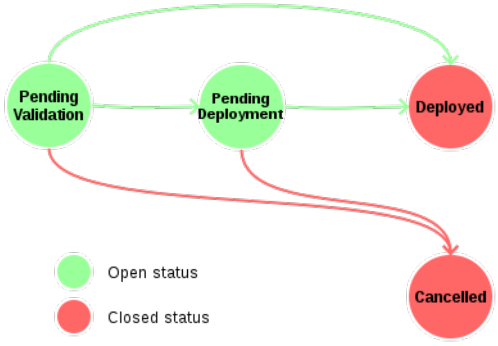
Deployed

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

Cancelled

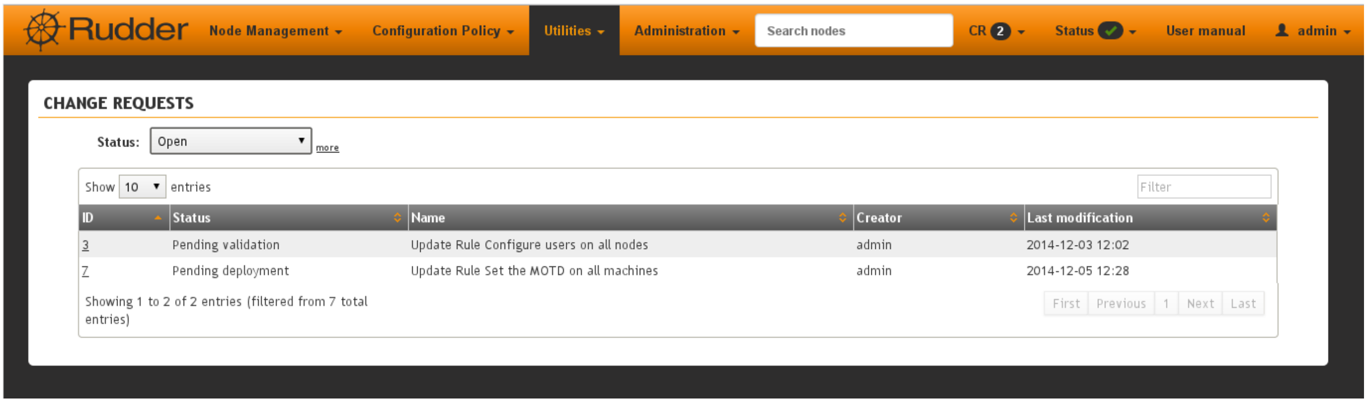
- The change was not approved.
- This is a final state, it can’t be moved anymore.

Here is a diagram about all those states and transitions:



7.6.2 Change request management page

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



CHANGE REQUESTS

Status: Open more

Show 10 entries

Filter

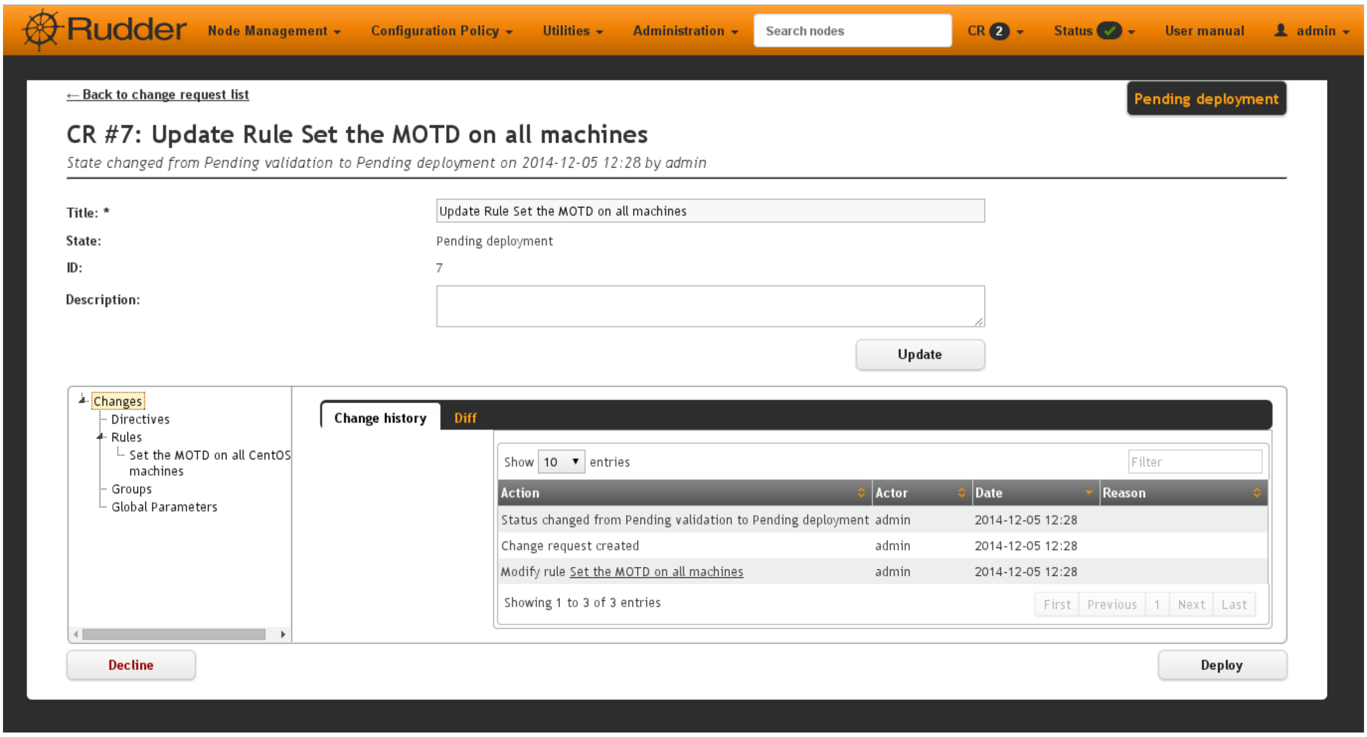
ID	Status	Name	Creator	Last modification
3	Pending validation	Update Rule Configure users on all nodes	admin	2014-12-03 12:02
7	Pending deployment	Update Rule Set the MOTD on all machines	admin	2014-12-05 12:28

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

First Previous 1 Next Last

7.6.2.1 Change request detail page

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



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

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

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

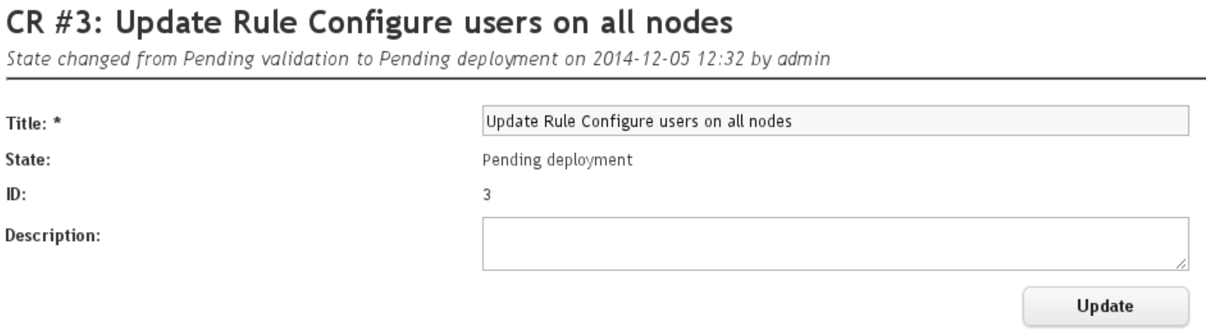
Show 10 entries Filter

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

Showing 1 to 3 of 3 entries First Previous 1 Next Last

The page is divided into two sections:

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



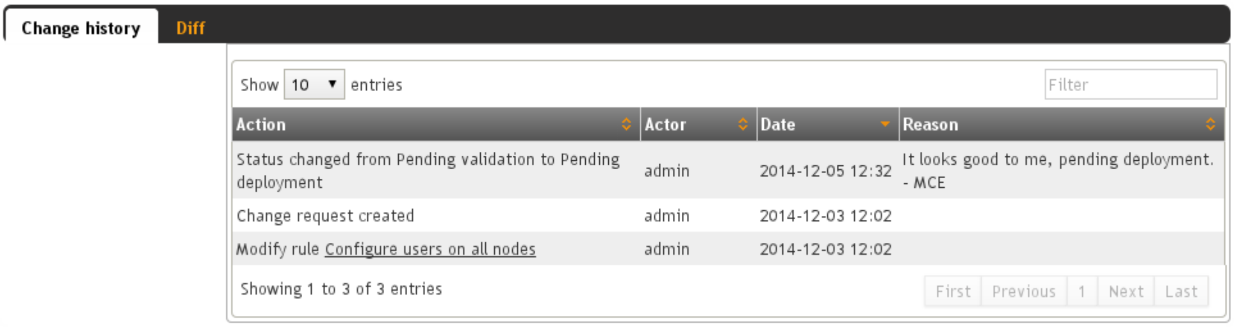
CR #3: Update Rule Configure users on all nodes

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

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

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

- History about that change request

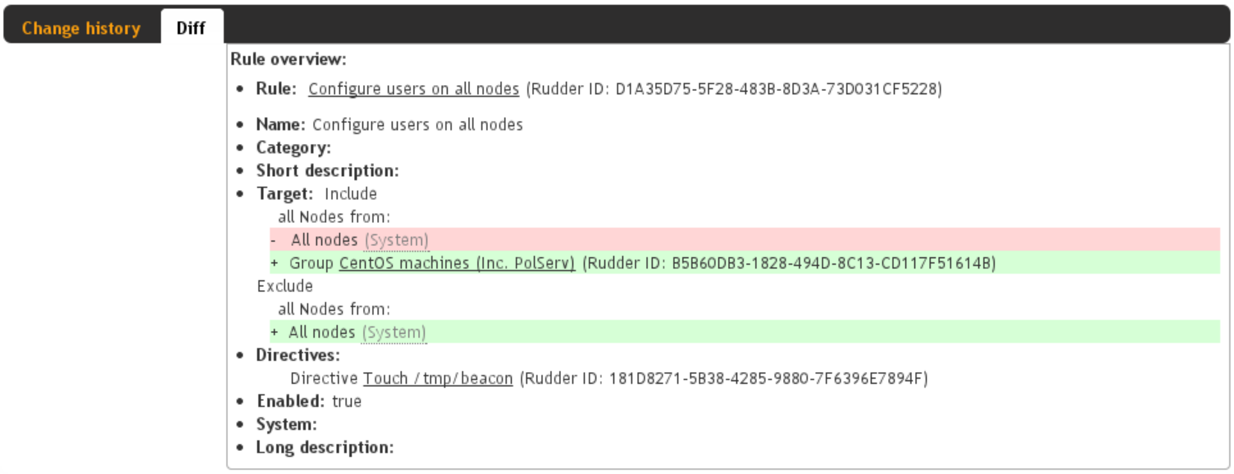


Show 10 entries Filter

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

Showing 1 to 3 of 3 entries First Previous 1 Next Last

- Display the change proposed



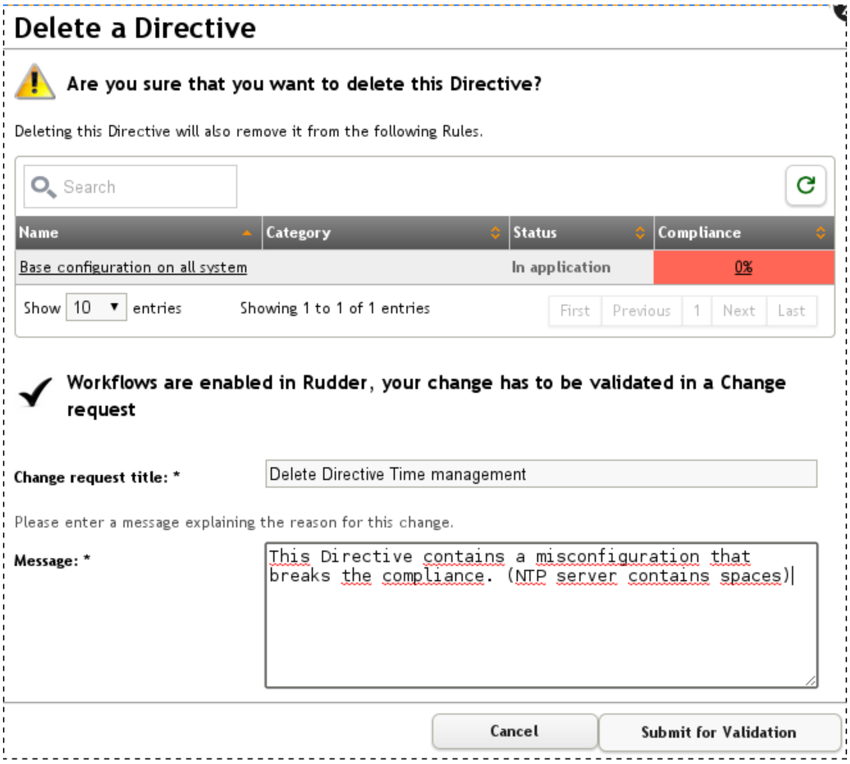
Rule overview:

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

7.6.3 How to create a Change request ?

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

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



Delete a Directive

Are you sure that you want to delete this Directive?

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

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

Show 10 entries Showing 1 to 1 of 1 entries

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

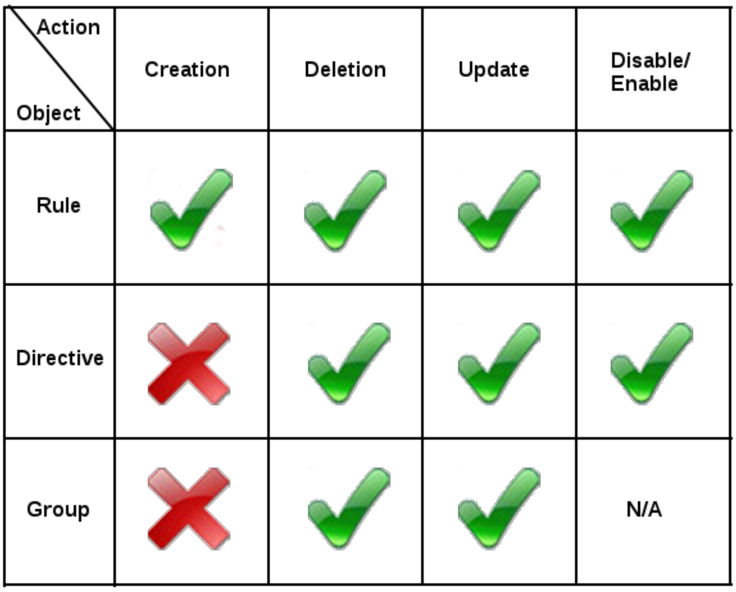
Change request title: * Delete Directive Time management

Please enter a message explaining the reason for this change.

Message: * This Directive contains a misconfiguration that breaks the compliance. (NTP server contains spaces)

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

Here is a small table about all possibilities:



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

7.6.4 How to validate a Change request ?

7.6.4.1 Roles

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

Validator Can validate Change request

Deployer To deploy Change Request

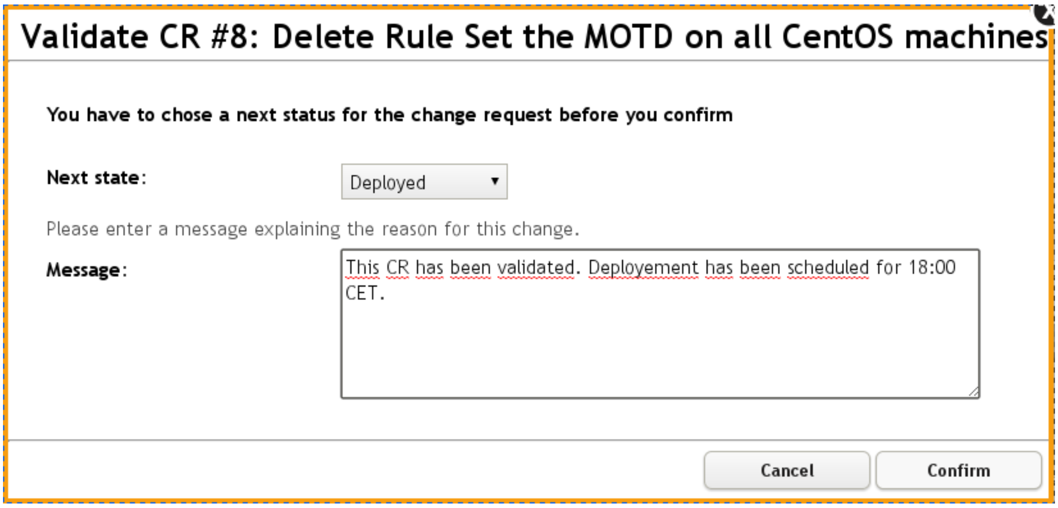
Both of those roles:

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

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

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

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



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

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

Next state: Deployed

Please enter a message explaining the reason for this change.

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

7.6.4.2 Self Validations

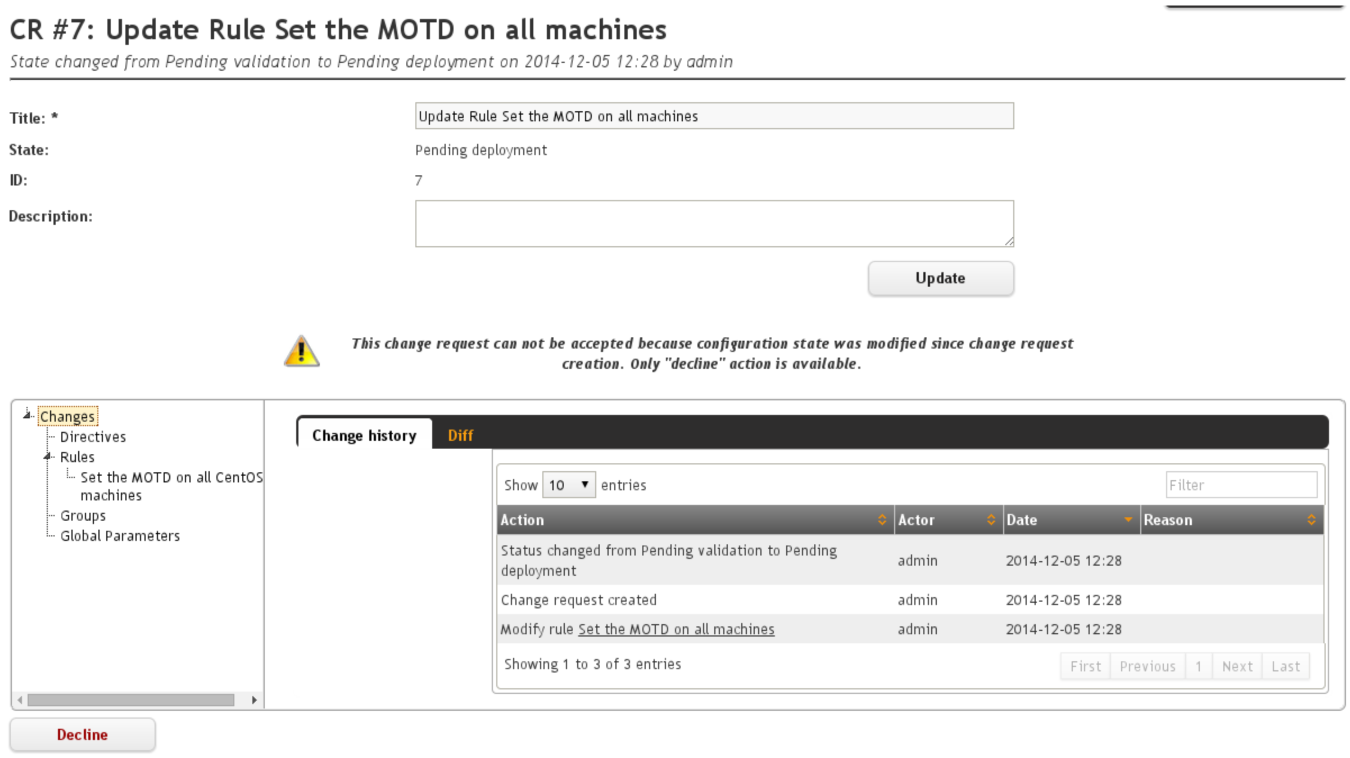
Using Change requests means that you want your team to share knowledge, and validate each other change. So by default:

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

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

7.6.5 Change request and conflicts

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



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

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

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

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

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

Showing 1 to 3 of 3 entries

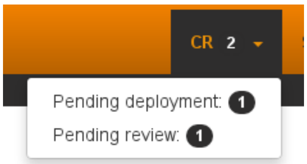
For now, we decided to reduce to the possibility of an error or inconsistency when there are concurrent changes. In a future version of *Rudder*, there will be a system to handle those conflicts, and make sure actual changes are not overwritten.

7.6.6 Notifications:

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

7.6.6.1 Pending change requests

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

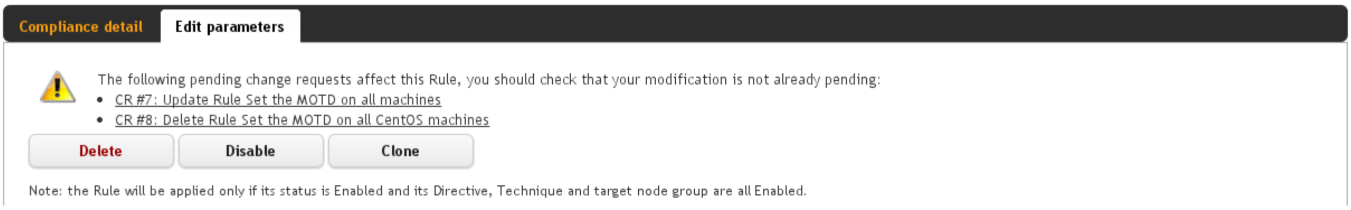


Pending deployment: 1

Pending review: 1

7.6.6.2 Change already proposed on Rule/Directive/Group

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



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

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

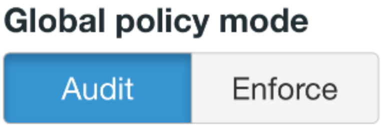
Note: the Rule will be applied only if its status is Enabled and its Directive, Technique and target node group are all Enabled.

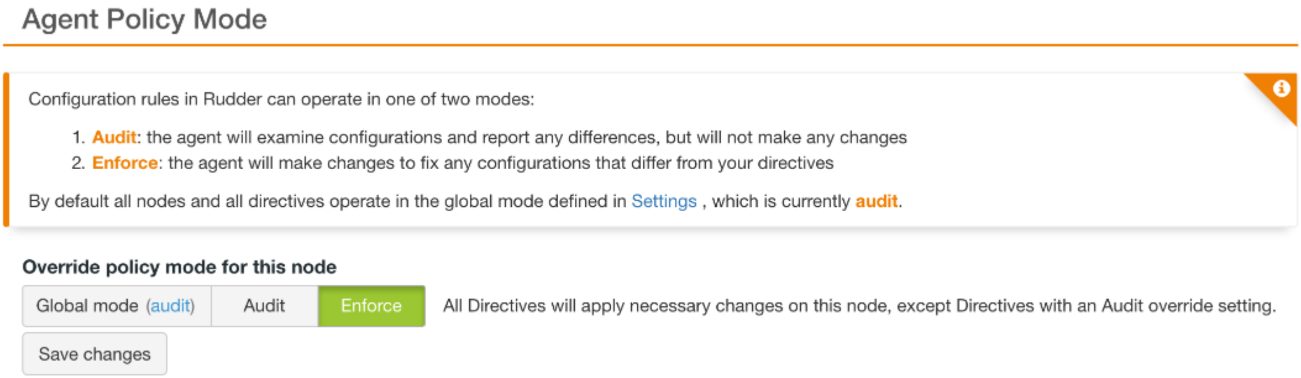
7.7 Policy Mode (Audit/Enforce)

Rudder 4.0 includes a policy mode setting, that allows two distinct behaviors:

- **Audit**: Test if the system is in the desired state, and report about it
- **Enforce**: Test if the system is in the desired state, if not, try to act to get to this state, and report about actions taken and final state

This allows for example to use Rudder as an audit tool or <<, _using_audit_mode_to_validate_a_policy_before_applying_it, to test a policy before enforcing it>>.





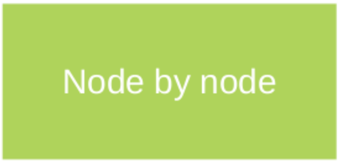
Configuration rules in Rudder can operate in one of two modes:

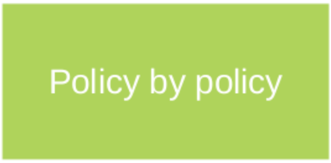
1. Audit: the agent will examine configurations and report any differences, but will not make any changes
2. Enforce: the agent will make changes to fix any configurations that differ from your directives

By default all nodes and all directives operate in the global mode defined in Settings , which is currently audit.

Override policy mode for this node

All Directives will apply necessary changes on this node, except Directives with an Audit override setting.





This mode can be set:

- Globally on the *Rudder* root server. In this can case there are two options: allow to override this mode on specific items, or use the global configuration everywhere.
- On a directive.
- On a node.

A lot of attention and several safeguards have been put in place to ensure that if you choose to use "Audit" for a target, nothing will be changed on the node for that target (except *Rudder*'s own configuration under `/var/rudder`), and only some harmless commands will be run (like listing installed packages or refreshing package lists).

Nodes are fully aware of exactly what directives need to be executed in Audit or in Enforce mode, and the "rudder agent" command line has been enhanced to let you see the result with a glimpse: the first column in `rudder agent run` output is now the mode (**A** for **Audit** and **E** for **Enforce**), and the compliance summary is split by audit mode. In addition to pre-existing technical reports, new ones have been added to report on "audit-compliant" (the check was OK), "audit-non-compliant" (the check was done, but the result is not the one expected), "audit-not-applicable" (the check is not applicable for that node, for example because of a limitation on the OS type), "audit-error" (the check wasn't able to finish correctly) status.

7.7.1 How is the effective mode computed?

We will here explain what is the computation made during generation to decide which mode to apply to a directive on a node, based on the current settings.

The short rule is: **Override wins, then Audit wins**

For a given directive on a given node at a given time, we have three different policy mode settings:

- The global mode, called **G**, which can be **Audit** or **Enforce**
- The node mode called **N**, which can be **Global** (if not overridden), **Audit, or *Enforce**
- The directive mode, called **D**, which can be **Global** (if not overridden), **Audit, or *Enforce**

The result is:

- If override is not allowed, the policy mode is **always** the global mode **G**.
- If override is allowed:
  - If **N** and **D** are set to use the **Global** default value (i.e. no override), the policy mode is the global mode **G**.
  - If **N** uses the **global** value and **D** is overriden to **Audit** or **Enforce**, the **D** value is used.
  - If **D** uses the **global** value and **N** is overriden to **Audit** or **Enforce**, the **N** value is used.
  - If **N** and **D** are overriden to **Audit** or **Enforce**, the value is **Audit** if at least one of **N** or **D** is **Audit**, **Enforce** if both are in **Enforce** mode

7.8 Technique editor

7.8.1 Introduction

7.8.1.1 First, what is a Technique ?

A technique is a description in code form of what the agent has to do on the node. This code is actually composed of a series of Generic method calls. These different Generic method calls are conditional.

7.8.1.2 What is a Generic method?

A generic method is a description of an elementary state independent of the operating system (ex: a package is installed, a file contains such line, etc…). Generic methods are independent of the operating system (It has to work on any operating system). Generic methods calls are conditioned by class expressions, which are boolean expression combining basic conditions with classic boolean operators (ex : operating system is *Debian*, such generic method produced a modification, did not produce any modification, produced an error, etc…)

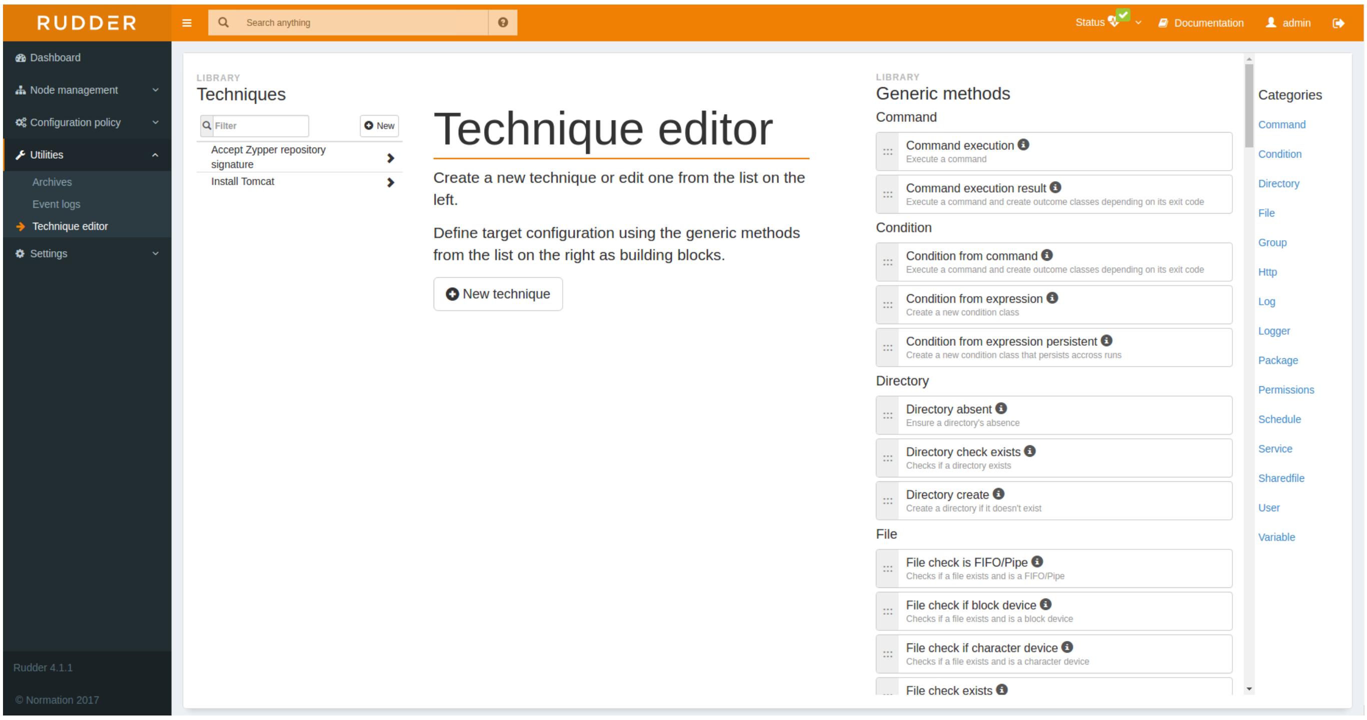
7.8.2 Technique Editor

7.8.2.1 Utility

Rudder provides a set of pre-defined *Techniques* that cover some basic configuration and system administration needs. Of course,this set of techniques cannot responds to all of the specific needs of each client. That's why *Rudder* integrate the ***Technique editor***, a tool to create advanced *Techniques*. Directly accessible from Ruder menu (*Utilities* > Technique *editor*), this tool has an easy-to-use interface, which doesn't require any programming skills but nevertheless allows to create complex *Techniques*.

7.8.2.2 Interface

Here is an overview of its interface :


RUDDER
Search anything
Status
Documentation
admin
Dashboard
Node management
Configuration policy
Utilities
Archives
Event logs
Technique editor
Settings
Rudder 4.1.1
© Normation 2017
LIBRARY
Techniques
Filter
New
Accept Zypper repository signature
Install Tomcat
New technique
LIBRARY
Generic methods
Command
Command execution
Execute a command
Command execution result
Execute a command and create outcome classes depending on its exit code
Condition
Condition from command
Execute a command and create outcome classes depending on its exit code
Condition from expression
Create a new condition class
Condition from expression persistent
Create a new condition class that persists accross runs
Directory
Directory absent
Ensure a directory's absence
Directory check exists
Checks if a directory exists
Directory create
Create a directory if it doesn't exist
File
File check is FIFO/Pipe
Checks if a file exists and is a FIFO/Pipe
File check if block device
Checks if a file exists and is a block device
File check if character device
Checks if a file exists and is a character device
File check exists
Categories
Command
Condition
Directory
File
Group
Http
Log
Logger
Package
Permissions
Schedule
Service
Sharedfile
User
Variable

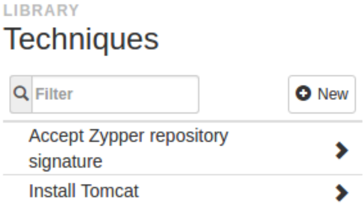
Technique editor

Create a new technique or edit one from the list on the left.

Define target configuration using the generic methods from the list on the right as building blocks.

The interface is divided into 3 columns:

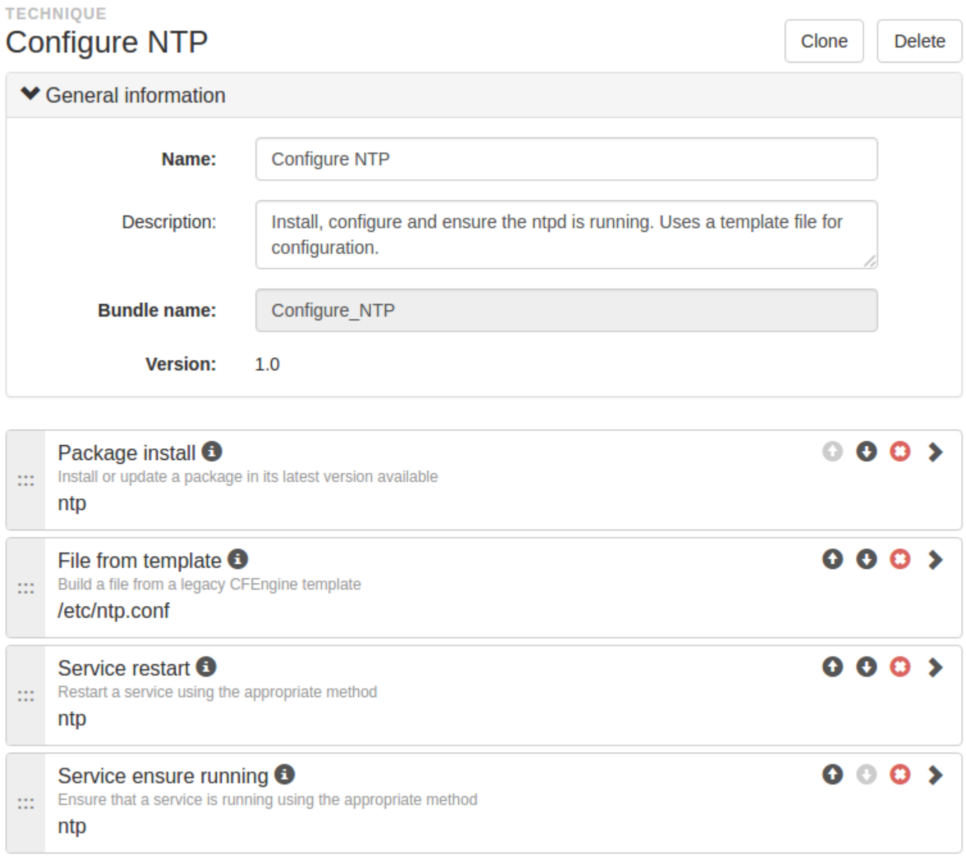
- A column listing custom *Techniques*



Here, we can see our previously created *Techniques*. We can click on them to see their details/edit them, or create a new one by clicking on the "New" button. Theses *Techniques* are visible in the **ncf techniques** category in the ***Directives tree***, so can be used to create new *Directives*.

- A column with the *Technique* content

When we create a new *Technique*, or when we edit an existing one, the configuration form appears at the center of the interface, instead of the title and the description of the tool.



TECHNIQUE

Configure NTP

General information

Name: Configure NTP

Description: Install, configure and ensure the ntpd is running. Uses a template file for configuration.

Bundle name: Configure_NTP

Version: 1.0

Package install
Install or update a package in its latest version available
ntp

File from template
Build a file from a legacy CFEngine template
/etc/ntp.conf

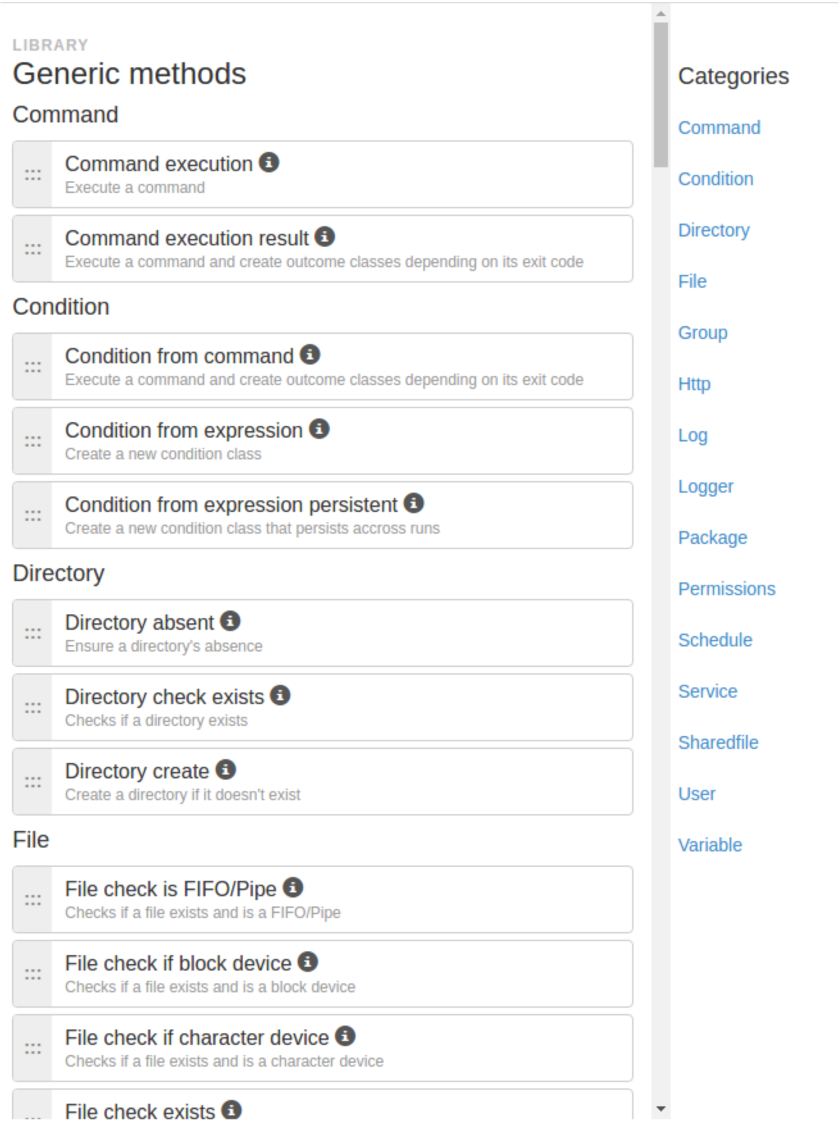
Service restart
Restart a service using the appropriate method
ntp

Service ensure running
Ensure that a service is running using the appropriate method
ntp

Then we can see the name, the description, the Bundle name, the version and the Generic methods list of the current *Technique*. Only the name and the description are editable, the Bundle name and the version are automatically defined during the *Technique* creation.

- A column listing Generic methods / displaying generic method details

To the right of the interface is the list of Generic methods available for *Technique* configuration. This list is made up of about a hundred Generic methods, grouped according to their category to make them easier to use. (An exhaustive list of them available at any time in the online product documentation can be found on the following link: http://www.rudder-project.org/doc/_generic_methods.html)



LIBRARY

Generic methods

Command

Command execution
Execute a command

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

Condition

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

Condition from expression
Create a new condition class

Condition from expression persistent
Create a new condition class that persists accross runs

Directory

Directory absent
Ensure a directory's absence

Directory check exists
Checks if a directory exists

Directory create
Create a directory if it doesn't exist

File

File check is FIFO/Pipe
Checks if a file exists and is a FIFO/Pipe

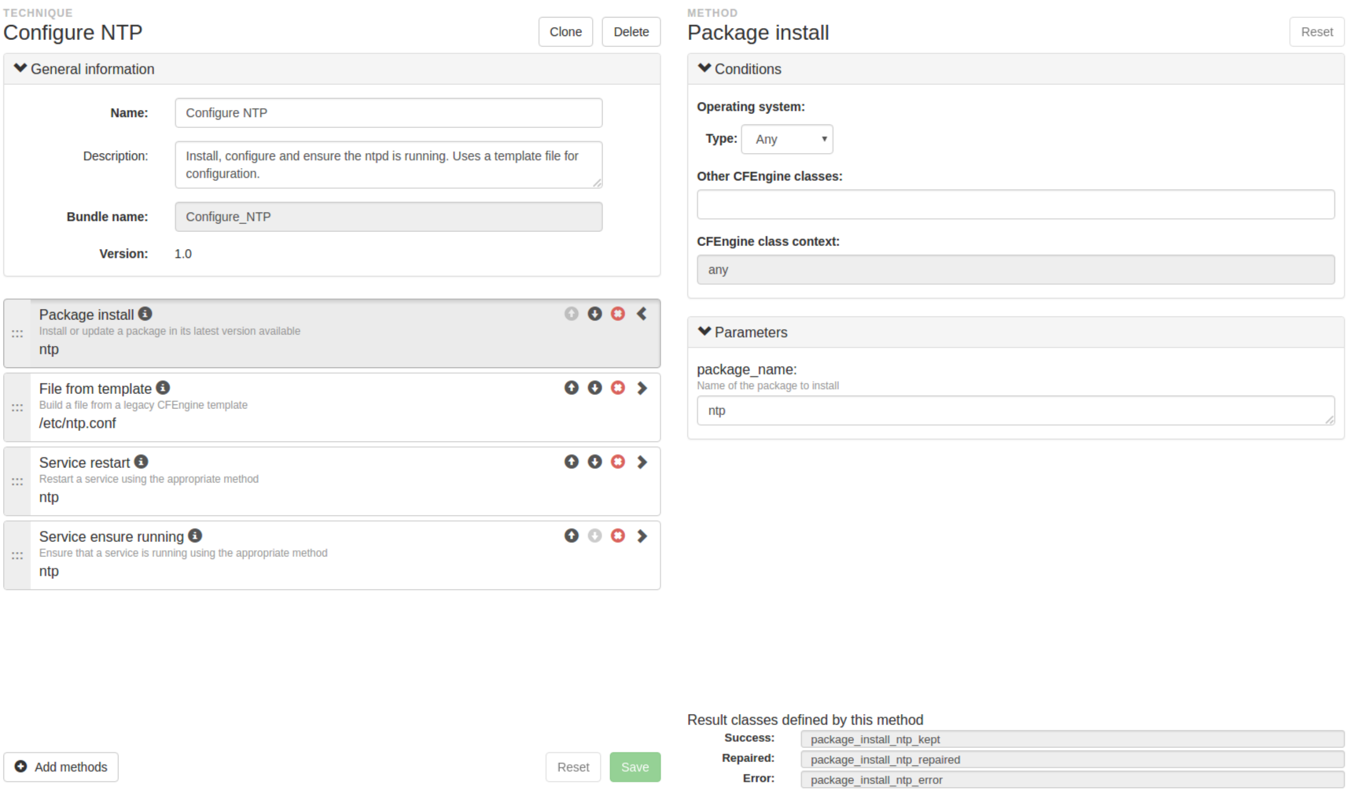
File check if block device
Checks if a file exists and is a block device

File check if character device
Checks if a file exists and is a character device

File check exists

Categories

You just need to click on a Generic method or drag'n drop it in the area provided for such purpose to add it to the current *Technique*. Once it's done, you can configure it by clicking on it. Then a new display containing the method details appears instead of the Generic methods list:



Configure NTP

General information

Name: Configure NTP

Description: Install, configure and ensure the ntpd is running. Uses a template file for configuration.

Bundle name: Configure_NTP

Version: 1.0

Package install
Install or update a package in its latest version available
ntp

File from template
Build a file from a legacy CFEngine template
/etc/ntp.conf

Service restart
Restart a service using the appropriate method
ntp

Service ensure running
Ensure that a service is running using the appropriate method
ntp

Package install

Conditions

Operating system:

Type: Any

Other CFEngine classes:

CFEngine class context: any

Parameters

package_name:
Name of the package to install
ntp

Result classes defined by this method

Success: package_install_ntp_kept

Repaired: package_install_ntp_repaired

Error: package_install_ntp_error

The Generic method details are divided into 3 blocks :

1. Conditions
   - Conditions allow user to restrict the execution of the method.
2. *Parameters*
   - *Parameters* are in mono or multi line text format. They can contains variables which will be extended at the time of the execution.
3. Result classes
   - One result class of three will be defined following the execution of a generic method:
     - Success, when the configuration is correct and no action are needed
     - Repaired, when the configuration is wrong and actions to fix it were executed with success
     - Error, when the configuration is wrong but actions to fix it failed

Theses classes can be used in another Generic methods conditions. ie, you can execute a command if a previous one failed or was repaired.

7.8.3 Create your first Technique

Now we are going to see how to create a simple technique to configure a ntp server, step by step.

7.8.3.1 1. General information

Let's start from the beginning. Click on the "*New Technique*" button and start filling in the General information fields (only name is required).

In our case:

- **Name**: *Configure NTP*
- **Description**: *Install, configure and ensure the ntpd is running. Uses a template file to configuration.*

7.8.3.2 2. Add and configure generic methods

Now, we have to find and add the generic methods which correspond to the actions we want to execute. In our case, we want to add the following methods:

- Package install (You can find it in the **Package category**)
  - This method only take one parameter, the name of the package to install. So here, fill in the **package_name** field with the value *ntp*.
- File from template (You can find it in the **File category**)
  - This method take two parameters. The first one corresponds to the absolute path of the source file containing a template to be expanded. We are going to use a *Rudder* variable here to get the correct path. Fill in the **source_template** field with the value *${path_technique}/templates/ntp.conf*.
  - The second corresponds to the absolute path of the destination file. Fill in with the value */etc/ntp.conf*.
- Service restart (You can find it in the **Service category**)
  - This method only take one parameter, the name of the service we want to restart. So here, fill in the **service_name** field with the value *ntp*.
  - Also, we want to restart the service only if it has just been installed, so only if the result classes defined following the execution of **Package install** method is **Repaired** (package_install_ntp_repaired). So here, fill in the **Other *CFEngine* classes** field in the Conditions panel with the value *package_install_ntp_repaired*.
- Service ensure running (You can find it in the **Service category**)
  - This method only take one parameter, the name of the service we want to check. Again, here, fill in the **service_name** field with the value *ntp*.

7.8.3.3 3. Save and apply your technique

And… It’s already done. Rather fast, right? Don’t forget to save. Now you can see it in the ***Directives tree***, and use it to create a *Directive* that will be applied on your Nodes thanks to a Rule.

Chapter 8

Configuration policies

8.1 How to

8.1.1 Enforce a line is present in a file only once

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

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

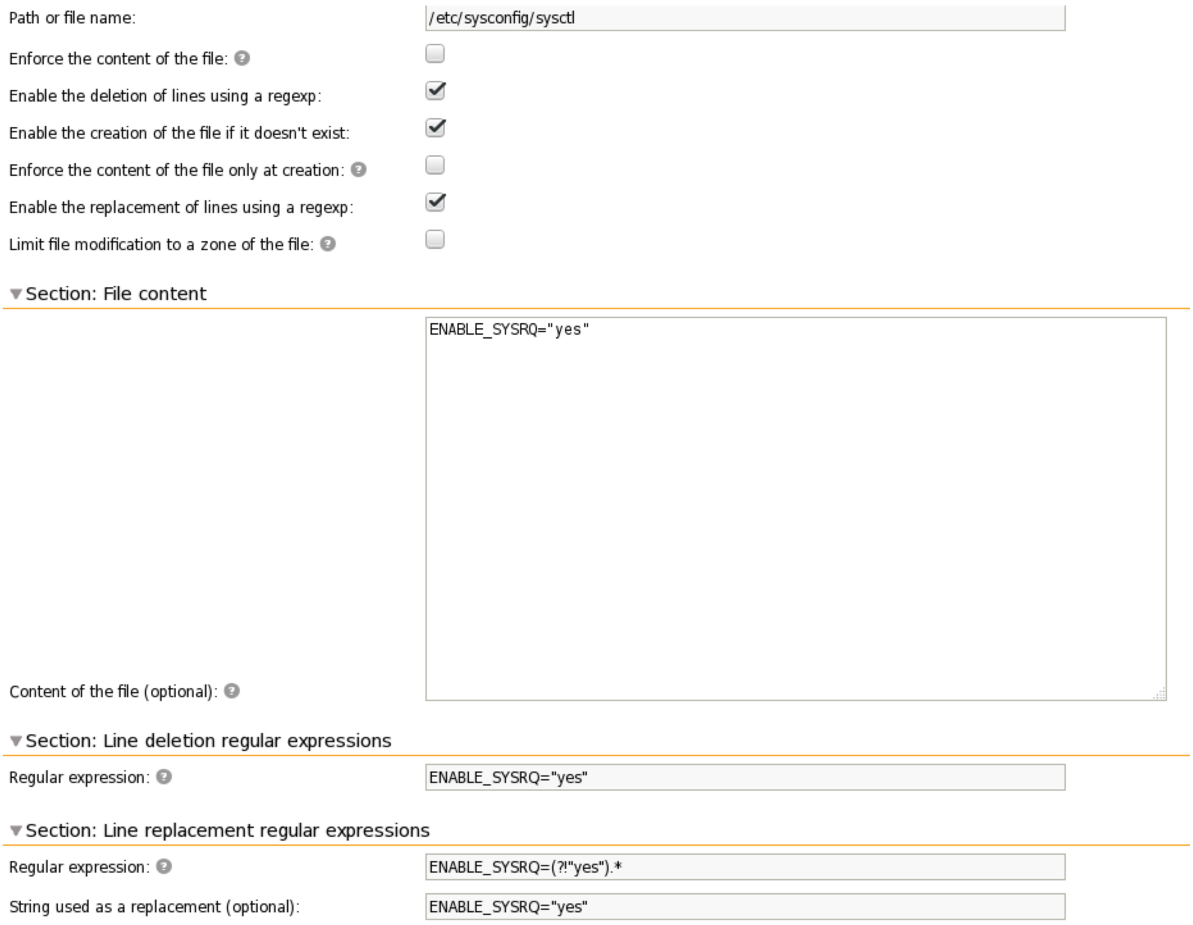
The whole logic to edit a file so it contain only one occurence of a line is:

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

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

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

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



Path or file name: /etc/sysconfig/sysctl

Enforce the content of the file: □

Enable the deletion of lines using a regexp: ☑

Enable the creation of the file if it doesn't exist: ☑

Enforce the content of the file only at creation: □

Enable the replacement of lines using a regexp: ☑

Limit file modification to a zone of the file: □

▼Section: File content

Content of the file (optional): ENABLE_SYSRQ="yes"

▼Section: Line deletion regular expressions

Regular expression: ENABLE_SYSRQ="yes"

▼Section: Line replacement regular expressions

Regular expression: ENABLE_SYSRQ=(?!"yes").*

String used as a replacement (optional): ENABLE_SYSRQ="yes"

8.1.2 Share files between nodes

Rudder 4.1 introduced a way to share files from one node to another. It allows a node to send a file to its relay, which will make it available for another target node, that has to to specifically download it.

This file sharing method is secured by:

- The control of uploaded file signature by the server, to check it matches the source node's private key.
- The same mechanism as standard file copy in *Rudder* to download the shared file from the server.

It also includes a ttl mechanism that allows sharing a file for a limited amount of time.

To use this feature, two generic methods are available in the technique editor:

- sharedfile_from_node: To download a file shared from another node.
- sharedfile_to_node: To make a file available to another node.

See the documentation of these methods for details about the required parameters, and especially sharedfile_to_node for a complete usage example.

8.2 Security considerations

8.2.1 Data confidentiality

Rudder is designed to strictly separate policies between nodes, and to only let a node access its own policies.

This section will give details about how the policies are secured, and which content is node-specific or global.

8.2.1.1 Private data

All confidential information should be stored in private data, namely:

- the directives, groups, rules, and their parameters
- the techniques parameters in the *Technique* Editor
- the shared-files directory

There are:

- always transfered encrypted between nodes (using agent copy protocol or https for the interface and the API)
- only available to the nodes that need it
- only accessible locally by the users that need it

More precisely:

- root server:
  - all the data is present on it
  - files are readable and writable only by the root user and (for some of them) the rudder group
  - some data is also accessible from our backends (PostgreSQL, OpenLDAP), but only locally (the services are listenning on loopback) and from *Rudder*-specific users, whith passwords only accessible to the root user
  - accessible remotely by the Web interface (needs an authorized user account) or the API (needs a token)
- relay: only the data needed for the served nodes and the relay itself are available and stored locally, only accessible to the root user
- node: only the data needed to configure the node is available and stored locally, only accessible to the root user

8.2.1.2 Common data

This refers to content available from all nodes in the authorized networks, readable from all users on the nodes (and that can be transfered withtout encryption when using initial promises of a pre-4.0 node).

These unprotected contents are:

- the tools (`/var/rudder/tools`)
- the common ncf part (`/var/rudder/ncf/common`), which includes all the content distibuted in the `ncf` package
- the *Rudder* techniques sources (without parameters), which includes all the content distibuted in the `rudder-techniques` package

8.2.2 Node-Server communication security

This section gives more details about the different flows between nodes and servers.

8.2.2.1 File copy

File copy is used to get policies and files copied during policy execution (named **shared-files**).

In 4.0 servers, two protocols are available:

- The old protocol (*CFEngine* "1" protocol), which is plain text by default with the ability to encrypt certain file transfers, kept for compatibility with older *Rudder* releases. It is deprecated and will disappear in future releases.
- The new protocol (*CFEngine* "2" protocol), which is TLS based. Everything is encrypted, still using the *CFEngine* keys.

Note: The new protocol was already available on the server since *Rudder* 2.11, but never used by the nodes.

The access policy is:

- Peer to peer key exchange, without central authority
- TOFU (Trust On First Use): keys are sent and accepted at first connection (from the policy server on nodes and from nodes with an IP in the Allowed Networks on policy servers).
- *Node*-specific files have an ACL containg the public key of the node, as found in the inventory
- Common files have an IP-based ACL based on the Allowed Networks

The old hostame and IP ACL are still generated for node-specific files to ensure compatibility with older nodes, but will be removed in the future.

8.2.2.2 Inventory

Nodes send an inventory to the server after installation or upgrade, and once a day.

This inventory contains various information, including:

- The node’s public key
- The node’s policy server

The inventory security policy is:

- Inventories are sent by the node to its configured policy_server over HTTPS, currently whithout certificate validation.
- Inventories are signed by the node using its private key, which allows the server to check this signature using the public key coming from previous inventory and to ensure it really comes from the right node.It avoids treating a malicious (or bogus) inventory coming from another node, and you should check the public key when accepting a new node.
- Once a node has sent a signed inventory, no unsigned inventory will be accepted for this node.

8.3 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…

8.3.1 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.

8.3.2 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.

8.3.3 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.

8.3.4 Using Rudder as an Audit tool

Using *Rudder* as an Audit tool is useful if you do not want to make any changes on the system, temporarily (freeze period, etc.) or permanently.

To use *Rudder* as an Audit tool without modifying any configuration on your systems, set the Policy Mode to **Audit** in the Settings, and do not allow overriding.

8.3.5 Using Audit mode to validate a policy before applying it

Before applying a configuration policy to some systems (a new policy or a new system), you can switch the policy mode of the directive defining this policy or of the nodes it is applied to to **Audit**.

This is particularly useful when adding rules to enforce policies that are supposed to be already applied: you can measure the gap between expected and actual state, and check what changes would be made before applying them.

Chapter 9

Basic administration

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

9.1 Archives

9.1.1 Archive usecases

The archive feature of *Rudder* allows to:

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

9.1.1.1 Changes testing

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

9.1.1.2 Changes qualification

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

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

9.1.2 Concepts

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

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

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

9.1.3 Archiving

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

To perform a global archive, the steps are:

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

9.1.4 Importing configuration

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

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

1. Uncompress the zip archive in /var/rudder/configuration-repository
2. If necessary, correct all files permissions: `chown -R root:rudder directives groups parameters ruleCategories rules techniques` and `chown -R ncf-api-venv:rudder ncf/50_techniques techniques/ncf_techniques`
3. Add all files in the git repository: `git add . && git commit -am "Importing configuration"`
4. Finally, in the Web interface, go to the *Administration* > *Archives* section, and select *Latest Git commit* in the drop down list in the Global archive section, and click on *Restore everything* to restore the configuration.

Tip

You can also perform the synchronisation from on environment to another by using git, through a unique git repository referenced on both environment.

For instance, using one unique git repository you can follow this workflow:

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

9.1.5 Deploy a preconfigured instance

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

9.2 Event Logs

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

Event Categories

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

9.3 Policy Server

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

9.3.1 Configure allowed networks

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

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

9.3.2 Clear caches

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

9.3.3 Reload dynamic groups

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

9.4 Plugins

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

A plugin is an `.rpkg` file (for "*Rudder* package").

9.4.1 Install a plugin

To install a plugin, copy the `.rpkg` file on your server, and run:

```
/opt/rudder/bin/rudder-pkg install-file <package.rpkg>
```

You can list currently installed plugins using:

```
/opt/rudder/bin/rudder-pkg list
```

You can also enable or disable, or remove a plugin with:

```
/opt/rudder/bin/rudder-pkg plugin enable <plugin>
/opt/rudder/bin/rudder-pkg plugin disable <plugin>
/opt/rudder/bin/rudder-pkg remove <package>
```

See all available commands with:

```
/opt/rudder/bin/rudder-pkg --help
```

9.5 Basic administration of Rudder services

9.5.1 Restart the agent of the node

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

```
service rudder-agent restart
```

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

9.5.2 Restart the root rudder service

9.5.2.1 Restart everything

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

```
service rudder restart
```

9.5.2.2 Restart only one component

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

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

```
service rudder-agent restart
```

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

```
service rudder-jetty restart
```

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

```
service apache2 restart
```

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

```
service rudder-slapd restart
```

SQL server Store the received reports from the nodes.

```
service postgresql* restart
```

9.6 REST API

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

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

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

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

9.6.1 Default setup

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

9.6.1.1 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
```

9.6.1.2 Apache access rules

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

Example 9.1 Example usage of non authenticated REST API

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

9.6.1.3 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 authenticated, the authenticated user name will be used in the logs.

9.6.2 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.

9.6.3 Promises regeneration

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

9.6.4 Dynamic groups regeneration

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

9.6.5 Technique library reload

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

9.6.6 Archives manipulation

Various methods are available to import and export items:

9.6.6.1 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

9.6.6.2 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).

9.6.6.3 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.

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

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

9.6.6.5 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
```

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

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

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

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

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

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

9.7 User management

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

9.7.1 Configuration of the users using a XML file

9.7.1.1 Generality

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

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

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

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

```
service rudder-jetty restart
```

9.7.1.2 Passwords

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

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

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

Here is an example of authentication file with hashed password:

```
<authentication hash="sha256">

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

</authentication>
```

Value	Algorithm	Linux command to hash the password
"md5"	MD5	`read mypass; echo -n $mypass | md5sum`
"sha" or "sha1"	SHA1	`read mypass; echo -n $mypass | shasum`
"sha256" or "sha-256"	SHA256	`read mypass; echo -n $mypass | sha256sum`
"sha512" or "sha-512"	SHA512	`read mypass; echo -n $mypass | sha512sum`

Table 9.1: Hashed passwords algorithms list

9.7.2 Configuring an LDAP authentication provider for Rudder

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

9.7.2.1 LDAP is only for authentication

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

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

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

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

```
service rudder-jetty restart
```

9.7.2.2 Enable LDAP authentication

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

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

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

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

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

Here are some usage examples,

on standard *LDAP*:

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

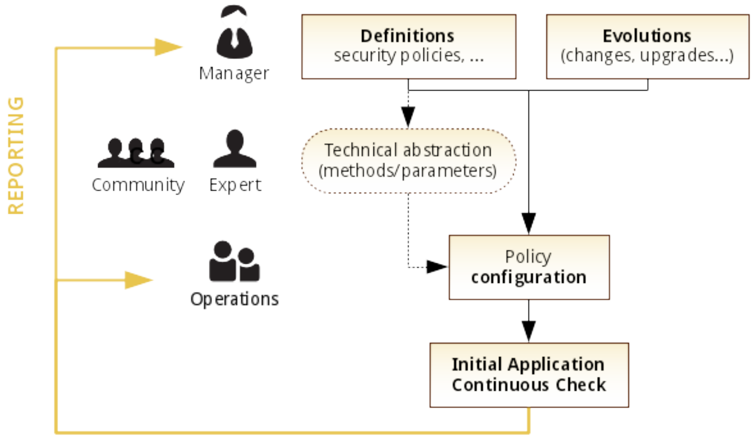
on *Active Directory*:

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

9.7.3 Authorization management

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

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



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

9.7.3.1 Pre-defined roles

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

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

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

9.7.3.2 Custom roles

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

A permission is composed of a type and a level:

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

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

Usage example:

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

9.7.4 Going further

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

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

Chapter 10

Advanced Node management

10.1 Node management

10.1.1 Reinitialize policies for a Node

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

```
rudder agent reset
```

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

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

10.1.2 Completely reinitialize a Node

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

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

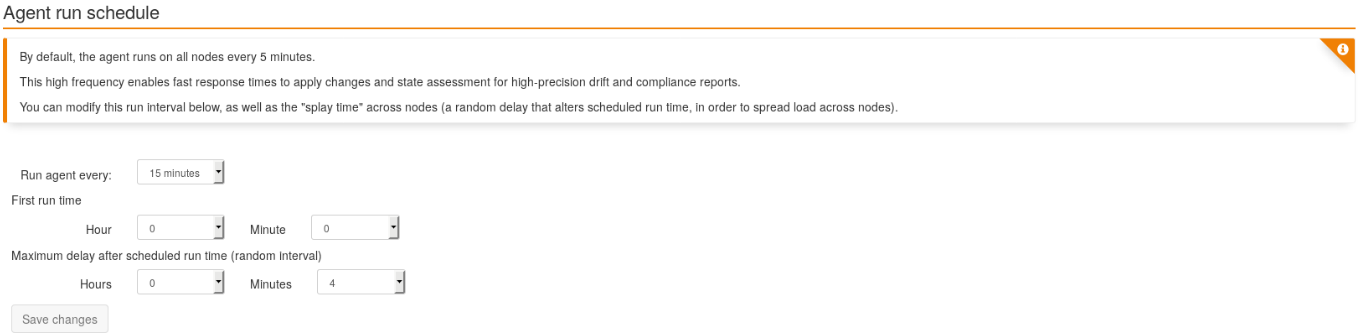
The command to reinitialize a *Node* is:

```
rudder agent reinit
```

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

10.1.3 Change the agent run schedule

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



Agent run schedule

By default, the agent runs on all nodes every 5 minutes.

This high frequency enables fast response times to apply changes and state assessment for high-precision drift and compliance reports.

You can modify this run interval below, as well as the "splay time" across nodes (a random delay that alters scheduled run time, in order to spread load across nodes).

Run agent every: 15 minutes

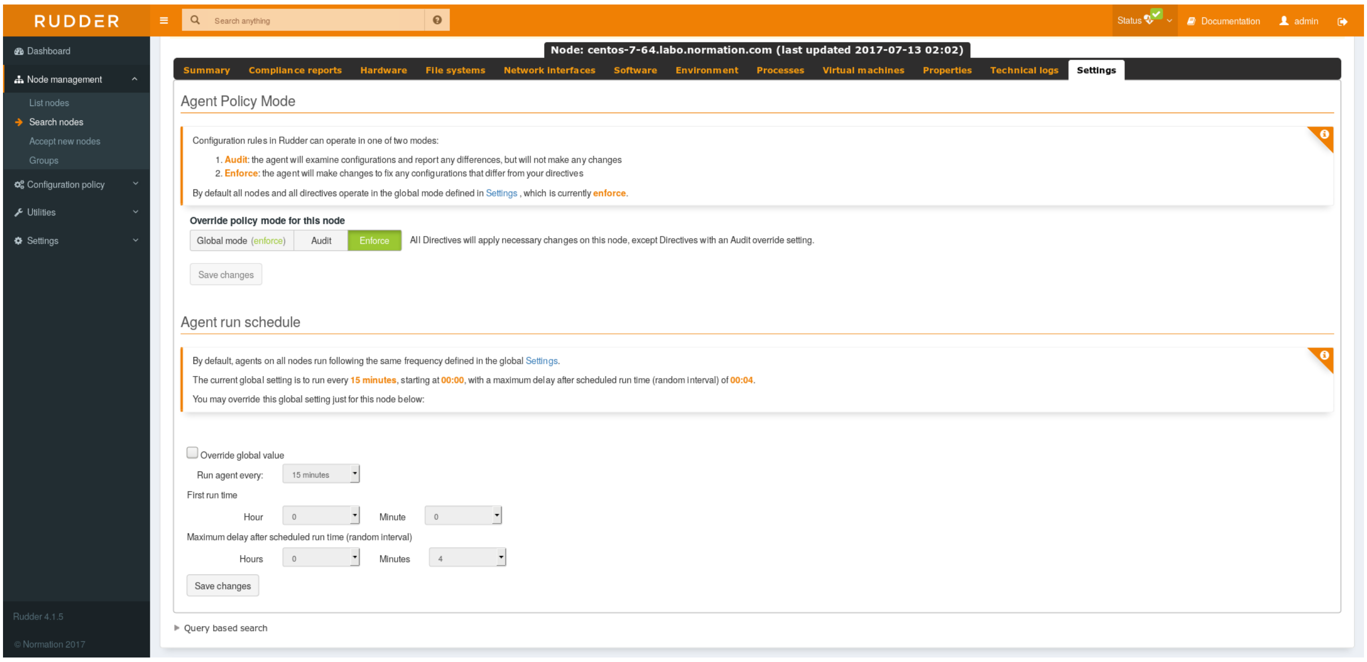
First run time

Hour 0 Minute 0

Maximum delay after scheduled run time (random interval)

Hours 0 Minutes 4

This settings can also be modified *Node* by *Node*, allowing you to customize the agent behavior (*Node* with little ressource like a Raspberry Pi or with limited bandwith). To do that, go into the *Node* details in the **Settings** tab



Node: centos-7-64.labo.normation.com (last updated 2017-07-13 02:02)

Agent Policy Mode

Configuration rules in Rudder can operate in one of two modes:

1. Audit: the agent will examine configurations and report any differences, but will not make any changes
2. Enforce: the agent will make changes to fix any configurations that differ from your directives

By default all nodes and all directives operate in the global mode defined in Settings , which is currently enforce.

Override policy mode for this node

Global mode (enforce) | Audit | Enforce — All Directives will apply necessary changes on this node, except Directives with an Audit override setting.

Agent run schedule

By default, agents on all nodes run following the same frequency defined in the global Settings.

The current global setting is to run every 15 minutes, starting at 00:00, with a maximum delay after scheduled run time (random interval) of 00:04.

You may override this global setting just for this node below:

☐ Override global value

Run agent every: 15 minutes

First run time

Hour 0 Minute 0

Maximum delay after scheduled run time (random interval)

Hours 0 Minutes 4

Warning

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

10.1.4 Installation of the Rudder Agent

10.1.4.1 Static files

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

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

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

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

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

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

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

10.1.4.2 Generated files

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

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

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

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

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

10.1.4.3 Services

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

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

cf-serverd This *CFEngine Community* daemon is listening on the network on *Rudder Root* and Relay servers, serving policies and files to *Rudder Nodes*.

10.1.4.4 Configuration

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

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

10.1.5 Rudder Agent interactive

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

```
rudder agent run
```

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

```
rudder agent run

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

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

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

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

```
# rudder agent run
 !! Error connecting to server (timeout)
 !!! System error for connect: "Operation now in progress"
 !! No server is responding on this port
Unable to establish connection with rudder-root
```

Restart the *CFEngine* service:

```
service rudder-agent restart
```

10.1.6 Processing new inventories on the server

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

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

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

10.1.6.2 Process incoming inventories

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

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

```
rudder agent run
```

10.1.6.3 Validate new Nodes

User interaction is required to validate new *Nodes*.

10.1.6.4 Prepare policies for the Node

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

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

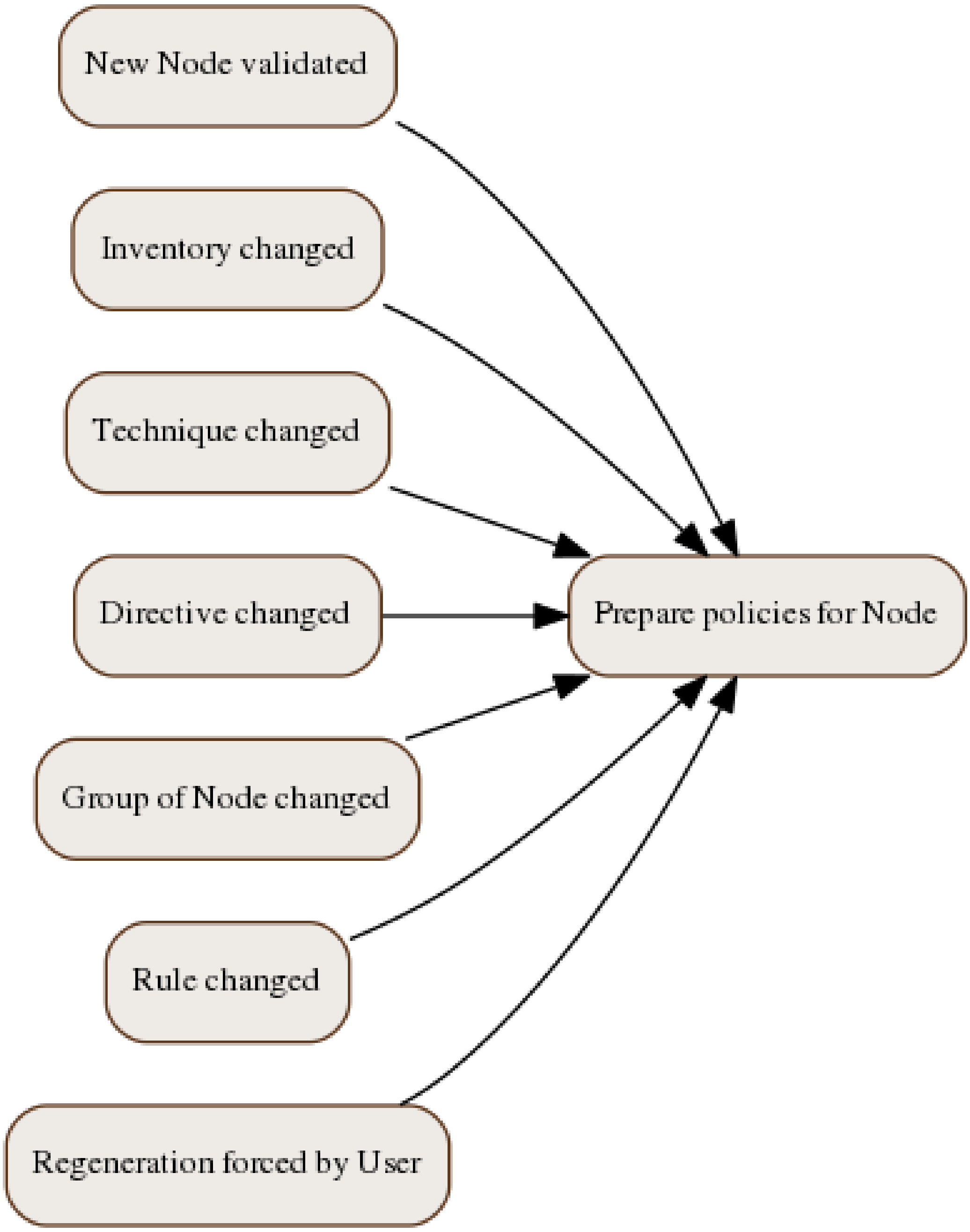


Figure 10.1: Generate policy workflow

10.1.7 Agent execution frequency on nodes

10.1.7.1 Checking configuration (CFEngine)

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

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

See Section 10.1.3 Section to see how to configure it

10.1.7.2 Inventory (FusionInventory)

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

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

Chapter 11

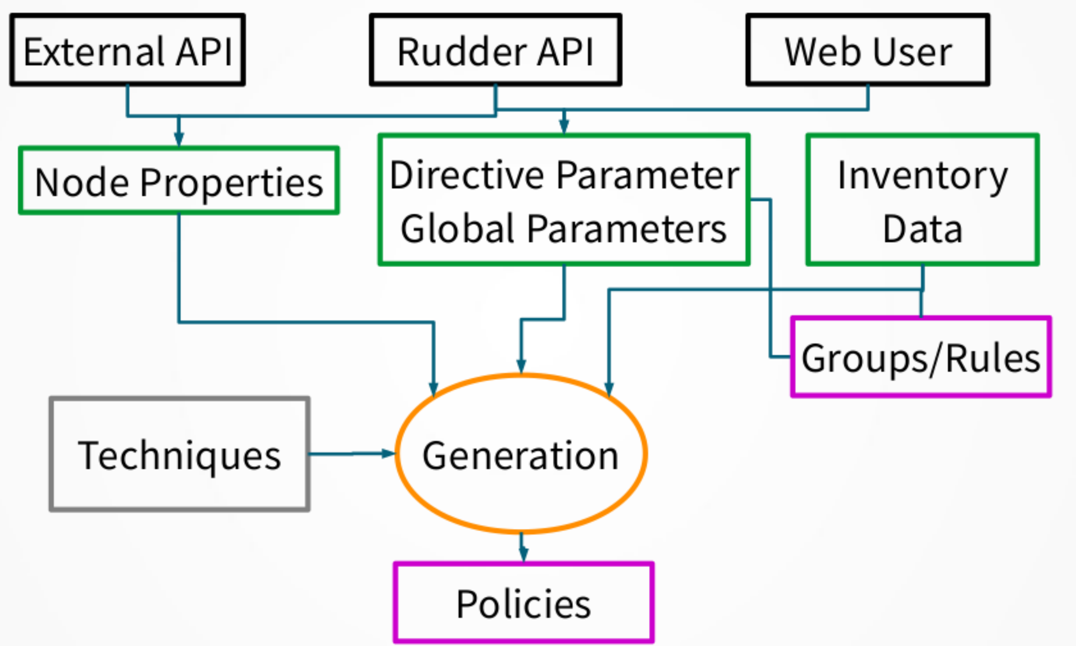
Advanced configuration

11.1 Policy generation

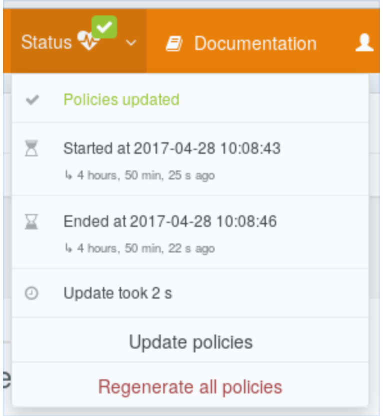
Each time a change occurs in the *Rudder* interface, having an impact on the policy needed by a node, it is necessary to regenerate the modified promises for every impacted node. By default this process is launched after each change.

The process of policy generation:

- Use configured policies and information about the nodes to generate the files defining the policy that reflects the desired state
- Compute and store expected reports that will be produced when executing these policies
- Check the validity of the generated policies
- Replace the old version of the policies by the new one for impacted node
- Restart the policy server on the *Rudder* central server is authorizations have changed



You can customize some of these actions and add new ones using the Server Event Hooks.



Started at 2017-04-28 10:08:43
↳ 4 hours, 50 min, 25 s ago

Ended at 2017-04-28 10:08:46
↳ 4 hours, 50 min, 22 s ago

Update took 2 s

11.1.1 `Update policies` button

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

11.1.2 `Regenerate all policies` button

The button `Regenerate all policies` on the top right of the screen, in the `Status` menu, allows you to force the regeneration of all policies. It will clear all internal caches, and force a complete computation of the policies. This is generally useful to make sure everything is correct after a problem on the central server.

11.2 Technique creation

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

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

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

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

11.2.1 Recommended solution: Technique Editor

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

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

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

11.2.1.1 Using the Technique Editor

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

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

11.2.1.2 Logs

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

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

11.2.2 Understanding how Technique Editor works

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

11.2.2.1 Directory layout

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

ncf directory structure exists in two root folders:

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

11.2.3 Sharing ncf code with nodes

To share those folders to all nodes, *Rudder* makes a copy of these folders in two places:

- `/var/rudder/ncf`, for part common to all nodes - so NOT techniques,
  - `/var/rudder/ncf/local` is a copy of node-independant directories from `/var/rudder/configuration-repository/ncf`, so almost everything **BUT** `/var/rudder/configuration-repository/ncf/50_techniques`.
  - `/var/rudder/ncf/common` is a copy `/usr/share/ncf/tree`
- `/var/rudder/share/xxxx-yyyy-node-id-zzzz/rules/cfengine-community/Technique_Name/1.0/Technique_Name.cf` for techniques, with one directory for each technique applied to the node.
- `/var/rudder/share/xxxx-yyyy-node-id-zzzz/rules/cfengine-community/rudder_expected_reports.csv` contains information about report expected for all ncf techniques applied to that node.

Files in `/var/rudder/ncf` are synchronized automatically by the "rudder agent update" command when the agent runs on the server. So any modification done in files in these directories will be lost at the next synchronization.

Files under `/var/rudder/share/` are updated during policy generation.

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

11.2.3.1 From ncf Technique Editor to Rudder Techniques and back

Here we will explain how the *Technique* Editor integration to *Rudder* is done to transform ncf techniques into full fledge *Rudder* one. We will also get the big picture of the web flow and the resulting events triggered on *Rudder* servier side.

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

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

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

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

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

https://your.rudder.server/ncf-builder/#!?path=/var/rudder/configuration-repository/ncf

Get requests are triggered when accessing *Technique* editor.

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

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

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

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

They are triggered when clicking on save/delete button.

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

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

11.2.3.2 Hooks

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

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

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

- `post.write_technique.commit.sh`
  - It commits the *Technique* newly created into *Rudder* Git configuration repository located in `/var/rudder/configuration-repository`.
- `post.write_technique.rudderify.sh`

- It generates a valid *Rudder Technique* from a the newly created *Technique* and reloads *Rudder Technique* Library so that updates are taken into account.

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

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

11.2.4 Create Technique manually

11.2.4.1 Prerequisite

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

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

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

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

11.2.4.2 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*.

11.2.4.3 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
```

Will output:

```
<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
```

11.3 Node properties

Node properties can be found in the "properties" tab of each node in *Rudder*.

Node properties can be modified using *Rudder*'s API, see http://www.rudder-project.org/rudder-api-doc/#api-*Nodes*-update*Node*Properti

Properties can also be defined on the node itself, to override locally properties.

Each property is a key=value pair. The value can be a string or a well-formatted JSON data structure.

Some examples: `datacenter=Paris` `datacenter= { "id": "FRA1", "name": "Colo 1, Paris", "location": "Paris, France", "dns_suffix": "paris.example.com" }`

11.3.1 Using properties

You can use node properties almost everywhere in *Rudder*:

- in directive parameters
- in the technique editor
- in your own techniques and generic methods

To use a property, simply use the variable node.properties with the variable call syntax.

Example with a property named *datacenter*:

```
${node.properties[datacenter]}
```

Warning
Before *Rudder* 3.1.14 and 3.2.7, node properties could not be used in JavaScript expressions (see following section), since they are evaluated during policy generation and node properties were only made available to agents at runtime. Since *Rudder* 3.1.14, 3.2.7 and 4.0.0 and later, you can enable a feature switch in "Administration/Settings" to enable node properties expansion in directive parameters. More details are available at Section 11.4.

In a mustache template, use:

```
{{{vars.node.properties.datacenter}}}
```

11.3.2 Local override

The agent searches for optionnal properties files `/var/rudder/local/properties.d/*.json`, and will override existing properties.

As a result, if you have node properties defined server side as `"sysctls_postgresql":{"kernel.shmall":"903330","kernel.shmmax":"3700041320"}` and `"vm":{"vm.dirty_ratio":"10"}`

and a local property file `/var/rudder/local/properties.d/postgresql_config.json` as

```
{
  "properties":
  {
    "sysctls_postgresql": {
      "kernel.shmmax":"5368709120"
    }
  }

}
```

The resulting properties will be:

`"sysctls_postgresql":{"kernel.shmmax":"5368709120"}` and `"vm":{"vm.dirty_ratio":"10"}`

`sysctls_postgresql` has been overriden by local property, and `vm` has been left untouched. Note that it is an override, as the semantic of merging is not deterministic with literal values, and it does not allow to unset values. If you need to merge, please refer to the next paragraph.

11.3.3 Merging properties

If you want to merge server defined properties with local defined properties, rather than override them, you will need to use the generic method variable_dict_merge_tolerant to define which variables you need to merge, and define the local variables in a different namespace than properties.

For instance, if you have defined in the node properties the following properties

`"sysctls_postgresql":{"kernel.shmall":"903330","kernel.shmmax":"3700041320"}`

and you wish to merge these values on a node with locally defined variable, to change the value of kernel.shmmax and set the value of kernel.shmmni, you can define the file /var/rudder/local/properties.d/postgresql_config.json with the following content

```
{
    "local_properties":
    {
        "sysctls_postgresql": {
            "kernel.shmmax":"5368709120",
            "kernel.shmmni":"4096"
        }
    }

}
```

and use the generic method `variable_dict_merge_tolerant` to merge `node.properties[sysctls_postgresql]` and `node.local_properties[sysctls_postgresql]`, and set the result in merged_properties.sysctls_postgresql (for instance): `variable_dict_merge_tolerant("merged_properties", "sysctls_postgresql", "node.properties[sysctls_postgresql]", "node.local_properties[sysctls_postgresql]")`

As a result, merged_properties.sysctls_postgresql will contain

---

"sysctls_postgresql": { "kernel.shmall":"903330", "kernel.shmmax":"5368709120", "kernel.shmmni":"4096" }

11.3.4 Under the hood

On the server, one or more properties files are written for each node in the `/var/rudder/share/<uuid>/rules/cfengine-community/properties.d/` directory. This directory is then copied to each node by the agent with all other promise files.

In the agent, properties are made available in the `node.<namespace>` container that contains the values. Those values are read from `/var/rudder/cfengine-community/inputs/properties/*.json`. All files are taken in order and override the previous ones - the last one wins.

The agent searches for optionnal properties files `/var/rudder/local/properties.d/*.json`, and will define variables or override existing properties.

Each file must contain at least 2 levels of JSON content, the first level is the namespace level and the second level is the key level.

The namespace name must be an ASCII name that doesn't start with _ and must match the following regex: `[a-zA-Z0-9][a-zA-Z0-9_]*`

For example:

```
{
  "properties":
  {
    "datacenter": "Paris",
    "environment": "production",
    "customer": "Normation"
  }
}
```

The merge is a first level merge done at the namespace level. This means that:

- a key in a namespace is fully overridden by the same key in the same namespace in a later file.
- a key in a namespace is never overriden by the same key in a different namespace
- a key that is overriden never retains original data even if it is a data container itself

The result key is available in the `node.<namespace>` data variable. A usage example:

```
${node.properties[datacenter]}
```

To get the original data (for debug only) there is the `properties.property_<fileid>` variable. A usage example:

```
${properties.property__var_rudder_cfengine_community_inputs_properties_d_properties_json[ ↩
    properties][datacenter]}
```

11.4 Node properties expansion in directives

It is possible to use properties defined on nodes to build *Directive* values. The resulting values will be computed during policy generation, and can therefore provide unique values for each node or be used in JavaScript expressions.

Properties on nodes are defined using *Rudder*'s *REST API*, with the *Update* Node *properties* API call. More details in our API documentation.

Properties can also be defined directly on the nodes, by creating properties files `/var/rudder/local/properties.d/*.json/`

11.4.1 Feature availability

This feature was introduced in *Rudder* 3.1.14, *Rudder* 3.2.7 and *Rudder* 4.0.0.

If you upgraded to 3.1.14 (or a later 3.1.x version) or 3.2.7 (or a later 3.2.x version) from a previous *Rudder* version, this feature is disabled by default in order to mitigate any risk of undesired side effects on existing installations. You can enable it in the Administration/Settings page, using the **Enable node properties expansion in *Directives*** switch.

Rudder installations from 4.0.0 onwards have this feature enabled by default.

11.4.2 Usage

In any directive text field, you can access properties defined on nodes using the following syntax:

```
${node.properties[property_name][key_one][key_two]}
```

where:

- `property_name` is the name of the property defined via the API
- `key_one` and `key_two` are keys in the JSON structure
- the value obtained is the string representation, in compact mode, of the entire node property or sub-structure of the JSON value
- if the key is not found, an error will be raised that will stop policy generation
- spaces are authorized around separators ([,],|,}..)

11.4.2.1 Providing a default value

Most of the time, you will need to provide a default value to node properties expansion to avoid a policy generation error due to missing node properties. This is also a good case to allow a simple override mechanism for a parameter where only some nodes have a specific value.

You can also use other node properties, or other *Rudder* parameters as defaults, using the same syntax as above.

Some examples:

```
${node.properties[datacenter][id] | default = "LON2" }
${node.properties[datacenter][name] | default = """Co-location with "Hosting Company" in ↩
    Paris (allows quotes)""" }
${node.properties[datacenter][id] | default = ${rudder.param.default_datacenter} }
${node.properties[netbios_name] | default = ${rudder.node.hostname} }
${node.properties[dns_suffix] | default = ${node.properties[datacenter][dns_suffix] | ↩
    default = "${rudder.node.hostname}.example.com" }

#or even use cfengine variables in the default
${node.properties[my_override] | default = "${cfengine.key}"}
```

11.4.2.2 Forcing expansion on the node

In some cases, you will want to use a `${node.properties[key]}` in a directive parameter, but you don't want to expand it during policy generation on the *Rudder* server, but instead let the value be expanded during the agent run on the node. Typically if the value is to be used by a templating tool, or if the value is known only on the node.

For these cases, you can add the "node" option to the property expression:

```
${node.properties[datacenter][id] | node }
```

This will be rewritten during policy generation into:

```
${node.properties[datacenter][id]}
```

Which will be considered as a standard variable by the agent, which will replaced this expression by its value if it's defined, or kept as is if it's unknown.

The variable content is read from `/var/rudder/cfengine-community/inputs/properties.d/properties.json`, and from the optionally defined `/var/rudder/local/properties.d/*.json` files. You can find more information on node properties in Section 11.3.

11.5 JavaScript evaluation in Directives

It is possible to use JavaScript expressions to build *Directive* values. The resulting values will be computed during policy generation, and can therefore provide unique values for each node.

11.5.1 Feature availability

This feature was introduced in *Rudder* 3.1.12, *Rudder* 3.2.5 for password fields only, and generalized for all fields in *Rudder* 3.1.14, *Rudder* 3.2.7 and *Rudder* 4.0.

If you upgraded to 3.1.12 (or a later 3.1.x version) or 3.2.5 (or a later 3.2.x version) from a previous *Rudder* version, this feature is disabled by default in order to mitigate any risk of undesired side effects on existing installations. You can enable it in the Administration/Settings page, using the **Enable script evaluation in *Directives*** parameter.

Rudder installations from 4.0 onwards have this feature enabled by default.

11.5.2 Usage

All standard JavaScript methods are available, and a *Rudder*-specific library, prefixed with `rudder.` also provides some extra utilities. This library is documented below.

For example, to get the first 3 letters of each node's hostname, you can write:

```
"${rudder.node.hostname}".substring(0,3)
```

Limitation of the scripting language
JavaScript expressions are evaluated in a sandboxed JavaScript environment. It has some limitations, such as:

- It cannot write on the filesystem
- Scripts are killed after 5 seconds of execution, to prevent overloading the system

11.5.3 Rudder utility library

11.5.3.1 Standard hash methods

The following methods allow to simply hash a value using standard algorithms:

- `rudder.hash.md5(string)`
- `rudder.hash.sha256(string)`
- `rudder.hash.sha512(string)`

These methods do not use a salt for hashing, and as such are not suitable for distributing passwords for user accounts on UNIX systems. See below for a preferable approach for this.

11.5.3.2 UNIX password-compatible hash methods

The following methods are specially designed to provided hashes that can be used as user passwords on UNIX systems (in `/etc/shadow`, for example). Use these if you want to distribute hashes of unique passwords for each of your nodes, for example.

Two different cases exist: support for generic Unix-like systems (Linux, BSD, . . .) and support for AIX systems (which use a different hash algorithm).

Available methods are:

- `rudder.password.auto(algorithm, password [, salt])`
- `rudder.password.unix(algorithm, password [, salt])`
- `rudder.password.aix(algorithm, password [, salt])`

The parameters are:

- `algorithm` can be "MD5", "SHA-512", "SHA512", "SHA-256", "SHA256" (case insensitive)
- `password` is the plain text password to hash
- `salt` is the optional salt to use in the password (we **strongly** recommend providing this value - see warning below)

The `unix` method generates Unix crypt password compatible hashes (for use on Linux, BSD, etc), while the `aix` method generates AIX password compatible hashes. The `auto` method automatically uses the appropriate algorithm for each node type (AIX nodes will have a AIX compatible hash, others will have a Unix compatible hash). We recommend always using `auto` for simplicity.

For example, to use the first 8 letters of each node's hostname as a password, you could write:

```
rudder.password.auto("SHA-256", "${rudder.node.hostname}".substring(0,8), "abcdefg")
```

Providing a salt
It is strongly recommended to provide a **salt** to the methods above. If no salt is provided, a random salt is created, and will be recreated at each policy generation, causing the resulting hashes to change each time. This, in turn, will generate an unnecessary "repaired" status for the password component on all nodes at each policy generation.

JVM requirements
This features is tested only on HotSpot 1.7 and 1.8, OpenJDK 1.7 and 1.8, IBM JVM 1.7 and 1.8.

JVM requirements for AIX password hashes
AIX password generation depends on the availability of **PBKDF2WithHmacSHA256** and **PBKDF2WithHmacSHA512** in the JVM. These algorithms are included by default on HotSpot 1.8 and OpenJDK 1.8 and upward. In the case where your JVM does not support these algorithms, typically on an IBM JDK or a JVM 1.7 version of HotSpot and OpenJDK, the hashing algorithm falls back to **SHA1** with **PBKDF2WithHmacSHA1**, and an error message will be logged. You can also check your JVM editor manual to add support for these algorithms.

11.5.4 Status and future support

In a future version of *Rudder*, JavaScript evaluation will be supported in all fields in *Directives*.

In the meantime, you can already test this functionality out by entering a JavaScript expression in any *Directive* field, prefixed by "evaljs:". Please be aware that this is unsupported and untested, so do this at your own risk.

If you do encounter any issues, please get in touch or open a ticket - we'd love to hear about them!

There is currently no plan to extend this support to the fields in the *Technique* editor.

11.6 Server Event Hooks

Rudder 4.1 introduces the possibility to execute files (hooks), typically scripts, when some predefined event occurs on *Rudder*.

11.6.1 Generalities about hooks

Hooks are organized in subdirectories. The root of the sub-directories is `/opt/rudder/etc/hooks.d/`

Each sub-directory has a name related to the event that will trigger the hooks execution. By default, a hook directory contains a template example and a Readme.txt file explaining the generalities about hooks and the specificities of that hook (parameters, etc).

Hooks must be executable. All executable files will be used as hooks EXCEPT if they end with one of the extensions listed in **/opt/rudder/etc/rudder-web.properties for property *rudder.hooks.ignore-suffixes**. A common convention is to use the **.disabled** suffixe to do so.

Non executable files will be ignored (which allows to put other files in these directories, like a readme, for example).

Hooks parameter are passed by environment variable. *Rudder* will fill dedicated environment variable for each hooks.

Hooks are executed sequentially, in lexical order. We encourage you to use the patter "NN-hookname", with NN a number like "05", "20", etc.

Hooks have normalized returned code. Return codes on hooks are interpreted as follow:

- 0 : success, no log (appart if debug one) , continue to next hook
- 1-31 : error , error log in /var/log/rudder/webapp/, stop processing
- 32-63 : warning, warning log in /var/log/rudder/webapp/, continue to next hook
- 64-255 : reserved for future use case. Behavior may change without notice.

Available Hooked events: for now, all hooks are related to different steps of the policy generation process. In the future, more hooks will be supported like node acceptation.

11.6.2 node-post-acceptance

11.6.2.1 When/What ?

This directory contains hooks executed after a node was successfully accepted.

Typically, these hooks triggers action on other system, like registering the node into a monitoring system or into an external CMDB, or to send notification.

11.6.2.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_NODE_ID : the nodeId
- RUDDER_NODE_HOSTNAME : the node fully qualified hostname
- RUDDER_NODE_POLICY_SERVER_ID: the node policy server id
- RUDDER_AGENT_TYPE : agent type ("cfengine-nova" or "cfengine-community")

11.6.3 node-post-deletion

11.6.3.1 When/What ?

This directory contains hooks executed after a node was successfully deleted.

Typically, these hooks clean resources related to that node and notify external services that the node was deleted from *Rudder*.

11.6.3.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_NODE_ID: the nodeId
- RUDDER_NODE_HOSTNAME: the node fully qualified hostname
- RUDDER_NODE_POLICY_SERVER_ID: the node policy server id
- RUDDER_AGENT_TYPE : agent type ("cfengine-nova" or "cfengine-community")

- RUDDER_POLICIES_DIRECTORY_CURRENT: the full path of the base directory containing policies for that node
- RUDDER_POLICIES_DIRECTORY_NEW: the full path of the base directory containing next policies for that node (during a generation)
- RUDDER_POLICIES_DIRECTORY_ARCHIVE: the full path of the base directory containing previous policies for that node
- RUDDER_NODE_ROLES: a comma separated list of node's server role name

11.6.4 node-pre-deletion

11.6.4.1 When/What ?

This directory contains hooks executed before a node is deleted.

Typically, these hooks interact with external services using knowledge to validate if the node should actually be deleted.

11.6.4.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_NODE_ID: the nodeId
- RUDDER_NODE_HOSTNAME: the node fully qualified hostname
- RUDDER_NODE_POLICY_SERVER_ID: the node policy server id
- RUDDER_AGENT_TYPE : agent type ("cfengine-nova" or "cfengine-community")
- RUDDER_POLICIES_DIRECTORY_CURRENT: the full path of the base directory containing policies for that node
- RUDDER_POLICIES_DIRECTORY_NEW: the full path of the base directory containing next policies for that node (during a generation)
- RUDDER_POLICIES_DIRECTORY_ARCHIVE: the full path of the base directory containing previous policies for that node
- RUDDER_NODE_ROLES: a comma separated list of node's server role name

11.6.5 policy-generation-finished

11.6.5.1 When/What ?

This directory contains hooks executed after policies are fully generated for all nodes, and these new policies are available for download for the node.

Typically, these hooks are used to log information about the generation which just happened or notify third parties that new policies are available (for ex: `cf-serverd` SIGHUP)

11.6.5.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_GENERATION_DATETIME : ISO-8601 YYYY-MM-ddTHH:mm:ss.sssZ date/time that identifies that policy generation.
- RUDDER_NODE_IDS : space separated list of node id updated during the process, or the empty string if no nodes were updated.

- RUDDER_END_GENERATION_DATETIME : ISO-8601 YYYY-MM-ddTHH:mm:ss.sssZ date/time when the generation ended (minus these hooks)
- RUDDER_NUMBER_NODES_UPDATED : integer >= 0; number of nodes updated (could be found by counting $RUDDER_NODE_IDS)
- RUDDER_ROOT_POLICY_SERVER_UPDATED: 0 if root was updated, anything else if not

11.6.6 policy-generation-node-finished

11.6.6.1 When/What ?

This directory contains hooks executed after policies are fully generated for node and made available for the node to download.

Typically, these hooks interact with external services using knowledge from the generated policies (ex: send node-properties JSON file to a third party service).

11.6.6.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_GENERATION_DATETIME : generation datetime: ISO-8601 YYYY-MM-ddTHH:mm:ss.sssZ date/time that identify that policy generation start
- RUDDER_NODE_ID : the nodeId
- RUDDER_NODE_HOSTNAME : the node fully qualified hostname
- RUDDER_NODE_POLICY_SERVER_ID : the node policy server id
- RUDDER_AGENT_TYPE : agent type ("cfengine-nova" or "cfengine-community")
- RUDDER_POLICIES_DIRECTORY_CURRENT : new policies directory (for ex for nodes under root: /var/rudder/share/$RUDDER_

Technically, you could infer RUDDER_POLICIES_DIRECTORY_NEW, from RUDDER_NODE_ID, but it's tedious for nodes behind a relay, and it is just simpler not to have to track what are the *Rudder* internal names, which may change without notice.

11.6.7 policy-generation-node-ready

11.6.7.1 When/What ?

This directory contains hooks executed after policies are fully generated for node, but before these new policies replace the old ones (technically, before we move /var/rudder/share/$NODEID/rules.new to /var/rudder/share/$NODEID/rules).

Typically, these hooks proceed to sanity checks on the new policies (ex: cf-promises), update permission on files, or interact with external services using information from the generated policies (ex: send node-properties JSON file to a third party service).

11.6.7.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_GENERATION_DATETIME : generation datetime: ISO-8601 YYYY-MM-ddTHH:mm:ss.sssZ date/time that identify that policy generation start
- RUDDER_NODE_ID : the nodeId
- RUDDER_NODE_HOSTNAME : the node fully qualified hostname

- RUDDER_NODE_POLICY_SERVER_ID : the node policy server id
- RUDDER_AGENT_TYPE : agent type ("cfengine-nova" or "cfengine-community")
- RUDDER_POLICIES_DIRECTORY_NEW: new policies directory (for ex for nodes under root: /var/rudder/share/$RUDDER_NODE

Technically, you could infer RUDDER_POLICIES_DIRECTORY_NEW, from RUDDER_NODE_ID, but it's tedious for nodes behind a relay, and it is just simpler not to have to track what are the *Rudder* internal names, which may change without notice.

11.6.8 policy-generation-started

11.6.8.1 When/What ?

This directory contains hooks executed when a policy generation starts.

Typically, these hooks are used to log information about the generation which just started or notify third parties that shared information to node should be updated (shared-folder, etc).

11.6.8.2 Parameters

Hooks parameters are passed by environment variable:

- RUDDER_GENERATION_DATETIME: generation datetime: ISO-8601 YYYY-MM-ddTHH:mm:ss.sssZ date/time that identify that policy generation.

11.7 New directives default naming scheme

When a new directive is created, by default the *Name* field is filled with the *Technique* name. For example, if you create a new *Directive* from the *Users Technique*, the Name field will get the value: "Users".

This not always what you want, especially for your custom *Techniques*. So you have the possibility to define new default values for Name, at *Technique* or at *Technique* and Version granularity.

This is done by adding or updating the file: `/var/rudder/configuration-repository/techniques/default-directive-names.conf`.

That file need to be commited in git, and the *Technique* library reloaded to take effect:

--- cd /var/rudder/configuration-repository/techniques/ vi default-directive-names.conf …. git add default-directive-names.conf git commit -m "Change default names for new directives" rudder server reload-techniques ---

The file format is a simple `techniqueId[/optionalVersion]: default name to use` format. The *Technique* ID is the name of the directory containing the *Technique* version directory in `/var/rudder/configuration-repository/techniques`.

For example, if we imagine that in your company, you have the internal convention to create one directive by user role with the login in the name, you would prefer to have a default value to:

--- Role <user-role>: <matching-login> ---

And then, for Users *Technique* version 7, you changed your mind and now use the scheme:

--- Role: [user-role] (with login [login]) ---

Then the file will look like:

--- # Default pattern for new directive from "userManagement" technique: userManagement= Role <user-role>: <matching-login>

# For userManagement version 2.0, prefer that pattern in new *Directives*: userManagement/7.0: Role: [user-role] (with login [login]) ---

Chapter 12

Advanced administration

This chapter covers more advanced administration task of *Rudder* services.

12.1 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.

12.1.1 Automatic PostgreSQL table maintenance

Rudder uses an automatic mechanism to automate the archival and pruning of the reports database.

By default, this system will:

- Archive reports older that 3 days (30 in *Rudder* 2.6)
- Remove reports older than 90 days

It thus reduces the work overhead by only making *Rudder* handle relevant reports (fresh enough) and putting aside old ones.

This is obviously configurable in /opt/rudder/etc/rudder-web.properties, by altering the following configuration elements:

- rudder.batch.reportscleaner.archive.TTL: Set the maximum report age before archival
- rudder.batch.reportscleaner.delete.TTL: Set the maximum report age before deletion

The default values are OK for systems under moderate load, and should be adjusted in case of excessive database bloating.

The estimated disk space consumption, with a 5 minute agent run frequency, is 150 to 400 kB per *Directive*, per day and per node, which is roughly 5 to 10 MB per *Directive* per month and per node.

Thus, 25 directives on 100 nodes, with a 7 day log retention policy, would take 2.5 to 10 GB, and 25 directives on 1000 nodes with a 1 hour agent execution period and a 30 day log retention policy would take 9 to 35 GB.

12.1.2 PostgreSQL database vacuum

In some cases, like a large report archiving or deletion, the *Rudder* interface will still display the old database size. This is because even if the database has been cleaned as requested, the physical storage backend did not reclaim space on the hard drive, resulting in a "fragmented" database. This is not an issue, as PostgreSQL handles this automatically, and new reports sent by the nodes to *Rudder* will fill the blanks in the database, resulting in a steady growth of the database. This task is handled by the autovacuum process, which periodically cleans the storage regularly to prevent database bloating.

However, to force this operation to free storage immediately, you can trigger a "vacuum full" operation by yourself, however keep in mind that this operation is very disk and memory intensive, and will lock both the *Rudder* interface and the reporting system for quite a long time with a big database.

Manual vacuuming using the psql binary

```
#~You can either use sudo to change owner to the postgres user, or use the rudder  ↩
    connection credentials.

#~With sudo:
sudo -u postgres psql -d rudder

#~With rudder credentials, it will ask the password in this case:
psql -u rudder -d rudder -W

# And then, when you are connected to the rudder database in the psql shell, trigger a  ↩
    vacuum:
rudder=# VACUUM FULL;

# And take a coffee.
```

12.1.3 LDAP database reindexing

In some very rare case, you will encounter some *LDAP* database entries that are not indexed and used during searches. In that case, OpenLDAP will output warnings to notify you that they should be.

LDAP database reindexing

```
# Stop OpenLDAP
service rudder-slapd stop

# Reindex the databases
service rudder-slapd reindex

# Restart OpenLDAP
service rudder-slapd restart
```

12.2 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.

12.2.1 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
```

12.2.2 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

# Disable Rudder agent
rudder agent disable

# Stop Rudder services
service rudder stop

# Drop the OpenLDAP database
rm -rf /var/rudder/ldap/openldap-data/*.mdb

# 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

# Start PostgreSQL
service postgresql start

#~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

# Enable Rudder agent
rudder agent enable

#~And restart the machine or just Rudder:
service rudder restart
```

12.2.3 Migration

To migrate a *Rudder* installation, just backup and restore your *Rudder* installation from one machine to another.

If your server address changed, you will also have to do the following on every node that is directly connected to it (managed nodes or relays):

- Remove the server public key `rm /var/rudder/cfengine-community/ppkeys/root-MD5=*.pub`
- Modify `/var/rudder/cfengine-community/policy_server.dat` with the new address, then you can force your nodes to send their inventory by running `rudder agent inventory`

12.3 Performance tuning

Rudder and some applications used by *Rudder* (like the *Apache* web server, or Jetty) can be tuned to your needs.

12.3.1 Reports retention

To lower *Rudder* server's disk usage, you can configure the retention duration for node's execution reports in `/opt/rudder/etc/rudder-web.properties` file with the options:

`rudder.batch.reportscleaner.archive.TTL=30`

`rudder.batch.reportscleaner.delete.TTL=90`

12.3.2 Apache web server

The *Apache* web server is used by *Rudder* as a proxy, to connect to the Jetty application server, and to receive inventories using the WebDAV protocol.

There are tons of documentation about *Apache* performance tuning available on the Internet, but the defaults should be enough for most setups.

12.3.3 Jetty

The Jetty application server is the service that runs *Rudder* web application and inventory endpoint. It uses the Java runtime environment (JRE).

The default settings fit the basic recommendations for minimal *Rudder* hardware requirements, but there are some configuration switches that you might need to tune to obtain better performance with *Rudder*, or correct e.g. timezone issues.

To look at the available optimization knobs, please take a look at `/etc/default/rudder-jetty` on your *Rudder* server.

12.3.4 Java "Out Of Memory Error"

It may happen that you get java.lang.OutOfMemoryError. They can be of several types, but the most common is: "java.lang.OutOfMemor
Java heap space".

This error means that the web application needs more RAM than what was given. It may be linked to a bug where some process consumed much more memory than needed, but most of the time, it simply means that your system has grown and needs more memory.

You can follow the configuration steps described in the following paragraph.

12.3.5 Configure RAM allocated to Jetty

To change the RAM given to Jetty, you have to:

```
# edit +/etc/default/rudder-jetty+ with your preferred text editor, for example vim:
vim /etc/default/rudder-jetty

Notice: that file is alike to +/opt/rudder/etc/rudder-jetty.conf+, which is the file with
default values. +/opt/rudder/etc/rudder-jetty.conf+ should never be modified directly  ↩
    because
modification would be erased by packaging in the following Rudder versuib update.

# modify JAVA_XMX to set the value to your need.
# The value is given in MB by default, but you can also use the "G" unit to specify a size  ↩
    in GB.

JAVA_XMX=2G

# save your changes, and restart Jetty:
service restart rudder-jetty
```

The amount of memory should be the half of the RAM of the server, rounded up to the nearest GB. For example, if the server has 5GB of RAM, 3GB should be allocated to Jetty.

12.3.6 Optimize PostgreSQL server

The default out-of-the-box configuration of PostgreSQL server is really not compliant for high end (or normal) servers. It uses a really small amount of memory.

The location of the PostgreSQL server configuration file is usually:

```
/etc/postgresql/9.x/main/postgresql.conf
```

On a *SuSE* system:

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

12.3.6.1 Suggested values on an high end server

```
#
# Amount of System V shared memory
# --------------------------------
#
# A reasonable starting value for shared_buffers is 1/4 of the memory in your
# system:

shared_buffers = 1GB
```

```
# You may need to set the proper amount of shared memory on the system.
#
#   $ sysctl -w kernel.shmmax=1073741824
#
# Reference:
# http://www.postgresql.org/docs/8.4/interactive/kernel-resources.html#SYSVIPC
#
# Memory for complex operations
# -----------------------------
#
# Complex query:

work_mem = 24MB
max_stack_depth = 4MB

# Complex maintenance: index, vacuum:

maintenance_work_mem = 240MB

# Write ahead log
# ---------------
#
# Size of the write ahead log:

wal_buffers = 4MB

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

effective_cache_size = 1024MB
```

12.3.6.2 Suggested values on a low end server

```
shared_buffers = 128MB
work_mem = 8MB
max_stack_depth = 3MB
maintenance_work_mem = 64MB
wal_buffers = 1MB
effective_cache_size = 128MB
```

12.3.7 CFEngine

If you are using *Rudder* on a highly stressed machine, which has especially slow or busy I/O's, you might experience a sluggish *CFEngine* agent run everytime the machine tries to comply with your *Rules*.

This is because the *CFEngine* agent tries to update its internal databases everytime the agent executes a promise (the .lmdb files 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.

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 a drawback: you should backup and restore the content of this directory manually in case of a machine reboot because all the persistent states are stored here, so in case you are using, for example the jobScheduler *Technique*, you might encounter an unwanted job execution because *CFEngine* will have "forgotten" the job state.

Also, note that the mode=0700 is important as *CFEngine* will refuse to run correctly if the state directory is world readable, with an error like:

```
error: UNTRUSTED: State directory /var/rudder/cfengine-community (mode 770) was not private ↩
    !
```

Here is the command line to use:

How to mount a RAMdisk on CFEngine state directory

```
# How to mount the RAMdisk manually, for a "one shot" test:
mount -t tmpfs -o size=128M,nr_inodes=2k,mode=0700,noexec,nosuid,noatime,nodiratime tmpfs / ↩
    var/rudder/cfengine-community/state

# How to put this entry in the fstab, to make the modification permanent
echo "tmpfs /var/rudder/cfengine-community/state tmpfs defaults,size=128M,nr_inodes=2k,mode ↩
    =0700,noexec,nosuid,noatime,nodiratime 0 0" >> /etc/fstab
mount /var/rudder/cfengine-community/state
```

12.3.8 Rsyslog

If you are using syslog over TCP as reporting protocol (it is set in **Administration** → **Settings** → **Protocol**), you can experience issues with rsyslog on *Rudder* policy servers (root or relay) when managing a large number of nodes. This happens because using TCP implies the system has to keep track of the connections. It can lead to reach some limits, especially:

- max number of open files for the user running rsyslog
- size of network backlogs
- size of the conntrack table

You have two options in this situation:

- Switch to UDP (in **Administration** → **Settings** → **Protocol**). It is less reliable than TCP and you can lose reports in case of networking or load issues, but it will prevent breaking your server, and allow to manage more *Nodes*.
- Stay on TCP. Do this only if you need to be sure you will get all your reports to the server. You will should follow the instructions below to tune your system to handle more connections.

All settings needing to modify */etc/sysctl.conf* require to run *sysctl -p* to be applied.

12.3.8.1 Maximum number of TCP sessions in rsyslog

You may need to increase the maximum number of TCP sessions that rsyslog will accept. Add to your */etc/rsyslog.conf*:

```
$ModLoad imtcp
# 500 for example, depends on the number of nodes and the agent run frequency
$InputTCPMaxSessions 500
```

Note: You can use *MaxSessions* instead of *InputTCPMaxSessions* on rsyslog >= 7.

12.3.8.2 Maximum number of file descriptors

If you plan to manage hundreds of *Nodes* behind a relay or a root server, you should increase the open file limit (10k is a good starting point, you might have to get to 100k with thousands of *Nodes*).

You can change the system-wide maximum number of file descriptors in */etc/sysctl.conf* if necessary:

```
fs.file-max = 100000
```

Then you have to get the user running rsyslog enough file descriptors. To do so, you have to:

- Have a high enough hard limit for rsyslog
- Set the limit used by rsyslog

The first one can be set in */etc/security/limits.conf*:

```
username hard nofile 8192
```

For the second one, you have two options:

- Set the soft limit (which will be used by default) in */etc/security/limits.conf* (with *username soft nofile 8192*)
- If you want to avoid changing soft limit (particularly if rsyslog is running as root), you can configure rsyslog to change its limit to a higher value (but not higher than the hard limit) with the *$MaxOpenFiles* configuration directive in */etc/rsyslog.conf*

You have to restart rsyslog for these settings to take effect.

You can check current soft and hard limits by running the following commands as the user you want to check:

```
ulimit -Sn
ulimit -Hn
```

12.3.8.3 Network backlog

You can also have issues with the network queues (which may for example lead to sending SYN cookies):

- You can increase the maximum number of connection requests awaiting acknowledgment by changing *net.ipv4.tcp_max_syn_backlog = 4096* (for example, the default is 1024) in */etc/sysctl.conf*.
- You may also have to increase the socket listen() backlog in case of bursts, by changing *net.core.somaxconn = 1024* (for example, default is 128) in */etc/sysctl.conf*.

12.3.8.4 Conntrack table

You may reach the size of the conntrack table, especially if you have other applications running on the same server. You can increase its size in */etc/sysctl.conf*, see the Netfilter FAQ for details.

12.4 Password management

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

12.4.1 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
```

12.4.2 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
```

12.4.3 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)

12.5 Password upgrade

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

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

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

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

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

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

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

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

12.6 Use a database on a separate server

This section allows installing a separate database only without splitting the rest of the server components like when using the rudder-multiserver-setup script. The setup is done in two places: on the database server and on the *Rudder* root server.

It also allows moving an existing database to another server.

Use different user and database names
It can be useful, for example if you want to share you database server between several *Rudder* root servers (see note below), to use a different database for your *Rudder* root server. To do so:

- Create the new database (replace `alternate_user_name`, `alternate_base_name` and specify a password):

```
su - postgres -c "psql -q -c \"CREATE USER alternate_user_name WITH PASSWORD ' ↩
    GENERATE_A_PASSWORD'\""
su - postgres -c "psql -q -c \"CREATE DATABASE alternate_base_name WITH OWNER =  ↩
    alternate_user_name\""
```

- Initialize it. First copy the initialization script:

```
cp /opt/rudder/etc/postgresql/reportsSchema.sql /opt/rudder/etc/postgresql/reportsSchema ↩
    -alternate.sql
```

- In the copied file, change the:

```
ALTER database rudder SET standard_conforming_strings=true;
```

To:

```
ALTER database alternate_base_name SET standard_conforming_strings=true;
```

- Then apply the script:

```
su - postgres -c "psql -q -U alternate_user_name -h localhost -d alternate_base_name \
    -f /opt/rudder/etc/postgresql/reportsSchema-alternate.sql"
```

- Follow the standard instructions of this section, with two differences:
  - You need to adjust the line added to `pg_hba.conf` to match your user and database name.
  - You need to also change the database name and user in `rudder-web.properties`.

Use the same database server for several Rudder root servers
It is possible to share the same database server between several *Rudder* instances, by following the preceding tip to use a different database than the default one. However, there are some important points to know:

- This database server can only be used with the rudder-db role in case of multiserver setup.
- This database server can only be a node for one of the *Rudder* servers. This also means that this root server will have indirect access to the content of the other databases.

12.6.1 On the database server

- Install and configure the agent on the node, and install the **rudder-reports** package.
- Change the `postgresql.conf` file (usually in `/var/lib/pgsql` or `/etc/postgresql`), to listen on the right interface to communicate with the server:

```
# you can use '*' to listen on all interfaces
listen_addresses = 'IP_TO_USE'
```

- Also ensure that network policies (i.e. the firewall settings) allow PostgreSQL flows from the root server to the database server.
- Add an authorization line for the server (in `pg_hba.conf`, in the same directory):

```
host    rudder          rudder          ROOT_SERVER_IP/32       md5
```

- Restart postgresql to apply the new settings:

```
service postgresql restart
```

- Execute the following command to configure the password (that should be the same as RUDDER_PSQL_PASSWORD in `/opt/rudder/etc/rudder-passwords.conf` on the root server):

```
su - postgres -c "psql -c \"ALTER USER rudder WITH PASSWORD ’ ↩
    RUDDER_SERVER_DATABASE_PASSWORD’\""
```

- Run an inventory to the server:

```
rudder agent inventory
```

12.6.2 On the root server

In the following section, DATABASE_HOST refers to the hostname of the new database server, and SERVER_HOST to the hostname of the root server.

- Remove the `rudder-server-root` and rudder-reports packages if installed. For example, you can run on *Debian*:

```
service rudder restart
apt-mark manual rudder-webapp rudder-inventory-endpoint
apt-get remove --purge rudder-reports
```

- You can also remove the postgresql package and database from the server if installed, but keep in mind you will lose all existing data. You can follow the backup and restore procedure to migrate the data to the new database.
- Change the hostname in `/opt/rudder/etc/rudder-web.properties`:

```
rudder.jdbc.url=jdbc:postgresql://DATABASE_HOST:5432/rudder
```

- Edit `/var/rudder/cfengine-community/inputs/rudder-server-roles.conf` and set the following line:

```
rudder-db:DATABASE_HOST
```

- Edit the /etc/rsyslog.d/rudder.conf file and change the hostname in:

```
:ompgsql:DATABASE_HOST,rudder,rudder,...
```

- Run an inventory:

```
rudder agent inventory
```

- Restart rudder services:

```
service rsyslog restart
service rudder restart
```

- Clear the cache (in Administration → Settings)

You should now have finished configuring the database server. You can check the technical logs to see if reports are correctly written into the database and read by the web application.

12.7 Multiserver Rudder

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

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

12.7.1 Preliminary steps

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

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

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

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

Add firewall rules:

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

12.7.2 Install rudder-relay-top

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

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

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

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

12.7.3 Install rudder-db

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

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

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

12.7.4 Install rudder-ldap

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

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

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

12.7.5 Install rudder-web

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

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

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

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

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

12.8 Mirroring Rudder repositories

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

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

To synchronize with the APT repository just type:

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

To synchronize with the RPM repository just type:

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

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

12.9 Monitoring

This section will give recommendations for:

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

12.9.1 Monitoring Rudder itself

12.9.1.1 Monitoring a Node

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

You can use the *rudder agent health* command to check for communication errors. It will check the agent configuration and look for connection errors in the last run logs. By default it will output detailed results, but you can start it with the *-n* option to enable "nrpe" mode (like Nagios plugins, but it can be used with other monitoring tools as well). In this mode, it will display a single line result and exit with:

- 0 for a success
- 1 for a warning
- 2 for an error

If you are using nrpe, you can put this line in your *nrpe.cfg* file:

```
command[check_rudder]=/opt/rudder/bin/rudder agent health -n
```

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

12.9.1.2 Monitoring a Server

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

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

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

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

12.9.2 Monitoring your configuration management

There are two interesting types of information:

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

12.9.2.1 Monitor compliance

You can use the *Rudder* API to get the current compliance state of your infrastructure. It can be used to simply check for configuration errors, or be integrated in other tools.

Here is an very simple example of API call to check for errors (exits with 1 when there is an error):

```
curl -s -H "X-API-Token: yourToken" -X GET 'https:/your.rudder.server/rudder/api/latest/ ↩
    compliance/rules' | grep -qv '"status": "error"'
```

See the API documentation for more information about general API usage, and the compliance API documentation for a list of available calls.

12.9.2.2 Monitor events

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

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

The lines have the following format:

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

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

You can use the following regex to match the different fields:

```
^\[(?P<Date>[^\]]+)\] N: (?P<NodeUUID>[^ ]+) \[(?P<NodeFQDN>[^\]]+)\] S: \[(?P<Result ↩
    >[^\]]+)\] R: (?P<RuleUUID>[^ ]+) \[(?P<RuleName>[^\]]+)\] D: (?P<DirectiveUUID>[^ ]+)  ↩
    \[(?P<DirectiveName>[^\]]+)\] T: (?P<TechniqueName>[^/]+)/(?P<TechniqueVersion>[^ ]+) C: ↩
     \[(?P<ComponentName>[^\]]+)\] V: \[(?P<ComponentKey>[^\]]+)\] (?P<Message>.+)$
```

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

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

filter {
   grok {
      match => { "message" => "^\[%{DATA:date}\] N: %{DATA:node_uuid} \[%{DATA:node}\] S:  ↩
          \[%{DATA:result}\] R: %{DATA:rule_uuid} \[%{DATA:rule}\] D: %{DATA:directive_uuid} ↩
           \[%{DATA:directive}\] T: %{DATA:technique}/%{DATA:technique_version} C: \[%{DATA: ↩
          component}\] V: \[%{DATA:key}\] %{DATA:message}$" }
   }
   # Replace the space in the date by a "T" to make it parseable by Logstash
   mutate {
      gsub => [ "date", " ", "T" ]
   }
   # Parse the event date
   date {
      match => [ "date" , "ISO8601" ]
```

```
    }
    # Remove the date field
    mutate { remove => "date" }
    # Remove the key field if it has the "None" value
    if [key] == "None" {
       mutate { remove => "key" }
    }
}

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

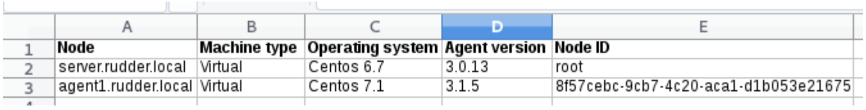
12.10 Use Rudder inventory in other tools

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

12.10.1 Export to a spreadsheet

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

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



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

You can easily modify the script to add other information.

12.10.2 Rundeck and Ansible

There are plugins for Rundeck and Ansible that can be used with each tool to make them aware of inventory data from *Rudder*. For more information, see details in the Third party integration with Rudder paragraph.

12.11 Directives ordering

Configuration in *Rudder* are based on desired states, describing the expected state of the system. However, there are cases where having order is desirable (like ensuring that a JVM is present before deploying an Application server, or ensuring a user is present before setting it sudoers), even if it will converge over the course of several agent runs.

In *Rudder*, there is two separated ways to order things, depending the type of *Technique*". So, before that, we need to explain how Policies are generated on the agent from *Directives* based on the same *Technique*.

12.11.1 Policy generation and Directive merge

In *Rudder*, Policies are generated from *Directives*, but several *Directives* based on the same *Technique* always lead to **one** Policy on the agent. For unique (non multi-instance) *Technique*, the one with the highest priority is selected. For multi-instance *Technique*, the different *Directive* values are **merged** into one Policy after having been sorted.

Separated Policy Generation in Rudder 4.3

```
In Rudder 4.3, that limitation is lifted and Technique can be made to generate  ↩
    ONE Policy for each Directive. That capacity is controled by the
`POLICYGENERATION` tag, where the value `merged` is the pre-4.3 default behavior ↩
    , and values `separated` or `separated-with-param` lead to one Policy per  ↩
    Directive.

See https://www.rudder-project.org/redmine/issues/10625[Don't merge directive  ↩
    from same technique on generation] for more information.
```

12.11.2 Sorting Directives based on the same Technique

For *Directive* based on the same *Technique*, the sort order is based on the **Priority** value of the *Directive*. Between two *Directive*, the one with the highest **Priority** is the first:

- for a **non** multi-instance *Technique*, it means that it is there is only one that is chosen in the resulting Policies (the others are discared),
- for a multi-instance *Technique*, it means that the variables in the Policy will be declared and check in sorting order of *Directives* (so the first *Directive*'s variables will be declared in first position and check first during an agent run).

If several ***Directives*** have the same **Priority**, the ***Rule*** **name**, and then the ***Directive*** **name** are used for sorting in alphanumeric order.

Priority field value and meaning
The **Priority** field of a *Directive* used to be a number, from 0 to 10, where 0 means "highest priority". This changed with https://www.rudder-project.org/redmine/issues/11725 but if you knew *Rudder* before that change, please use "0" whenever the documentation says "highest priority".

12.11.2.1 Special use case: overriding generic_variable_definition

You can use the merging of *Directive* to define variable override with the "Generic Variable Definition" *Technique*.

For example, let say you want to define a **DNS** variable with default value **[default dns]** and on some node case, a value **[overrided dns]**:

- Create a *Directive* [1] with **high** priority: it will be your **default** case, so set **DNS** to **[default dns]**.
- Create an other *Directive* [2] with **lower** priority: it will be you specialized case, so set **DNS** to **[overrided dns]**.

Then, a node with only *Directive* [1] will have the default value defined, and a node with both *Directives* will have the overriding one.

It works because on the agent, you can redeclare a variable name and reassign to it a new value: the last one wins (so in our case, the **less** prioritary).

12.11.3 Sorting Policies

Rudder uses a best-effort method for ordering Policies, based on alphanumeric ordering of the corresponding *Rule*, then *Directive* name.

When several *Directive* were merged, *Rudder* choose the first (*Rule* name, *Directive* name) as the ordering value to use for the resulting Policy.

Best practice
You should always start *Rules* and *Directives* name by 2 (or 3) digits to be able to easily reorder Policy evaluation if the need happen:
Do not use: "My general security rule" and "Check ssh configuration"
But use: "05. My general security rule" and "40. Check ssh configuration"

12.11.4 Example

- given three *Techniques* A, B and C
- directives A1 and A2 based on *Technique* A, directives B1 and B2 based on B, directives C1 and C2 based on C
- all *Directives* have the same priority,
- rule R0 having [C1], R1 having [A1, B2] and rule R2 having [A2, B1, C2], all applied on a same node,
- merging (R0, C1) and (R2, C2) ⇒ [C1, C2] and keep (R0, C1) as Policy order
- merging (R1, A1) and (R2, A2) ⇒ [A1, A2] and keep (R1, A1) as Policy order,
- merging (R1, B2) and (R2, B1) ⇒ [B2, B1] (because R1 < R2) and keep (R1, B2) for policy order,
- so policies are sort: (R0, C1) then (R1, A1) then (R1, B2)
- resulting ordering of directive's values will be: [C1, C2] then [A1, A2] then [B1, B2]

Chapter 13

Troubleshooting and common issues

All technical and general answers are on http://faq.rudder-project.org/.

Chapter 14

Rudder extension and integration with third party software

Rudder was thought from the begining to be a good citizen in you infrastructure. Part of that good will intent is translated into the fact that everything is done so that *Rudder* is able to inter-operate well with the other parts of your infrastructure.

For that, there is two mains way of integrating *Rudder* with your existing infrastructure components: you can either extend *Rudder* with plugins, or you can extend your existing tools or process to take advantage of *Rudder*.

14.1 Extending Rudder with plugins

Rudder can be extended with Plugins so that new features or API endpoints are available in *Rudder* web application.

14.1.1 Rudder Plugin

A plugin is an archive in the `.rpkg` file format that can be manipulated with the `rudder-pkg` command (see Plugins Administration)

A *Rudder* plugin has full access to all *Rudder* internal APIs, datas, and process. Its power is very large, but some care must be taken to ensure that the plugin does not break *Rudder* main use cases. That is why we prefer to build smaller plugin, adding only one feature, and doing it in the least impacting way.

Here come a list of some plugins so that one can grasp the kind of feature that a plugin can bring to *Rudder*:

14.1.1.1 Extending API: rudder-plugin-itop

Link: https://github.com/normation/rudder-plugin-itop

This plugin was used to add new API endpoint dedicated to the integration with iTop CMDB software.

As of *Rudder* 4.0, the plugin is superseeded by *Rudder* Compliance API.

14.1.1.2 Adding information to node details: rudder-plugin-external-node-information

Link: https://github.com/normation/rudder-plugin-external-node-information

This plugin allows to add new tabs in *Rudder* "node details" page and display node specific files, stored in node-dedicated places. It also use a self-managed and hot-reloading configuration of its properties.

14.1.1.3 Providing new authentication methods

Rudder plugins can be used to provide new authentication methods. There is no open source version of such module, but at least a Radius Authentication Plugin exists.

14.1.1.4 Providing a full new feature: rudder-plugin-datasources

Link: https://github.com/normation/rudder-plugin-datasources

As we said, *Rudder* plugins can be quite powerful. For example, the "Data sources Plugin" brings a completely new feature to *Rudder* by allowing to configure external API data sources from which *Rudder* will get node properties. The plugin set-up its own data base table, comes with its own UI (available in *Rudder* "administration" page), and interacts with node properties.

14.1.2 Building your own plugins

As of *Rudder* 4.1, there is no dedicated, frozen plugins API for plugins. A plugin is built in Scala, and the normal starting point is to clone and study the template plugin project, rudder-plugin-helloworld.

The project code source is documented to be didactic and provides:

- an example of the packaging resources needed to build a ".rpkg" package,
- example of configuration file for the plugin,
- plugin definition and plugin registration when *Rudder* starts,
- how to interact with *Rudder* internal services,
- how to define new APIs.

Of course, you can look to the other open source plugins listed above to get other, more involved example about how to do things.

You also can interact with *Rudder* developers through the community regular communication channels.

14.2 Rudder integration in your infrastructure

The other mains way to integrate *Rudder* into an existing infrastructure is by making existing process or software take advantage of *Rudder*.

14.2.1 Existing third party integration

14.2.1.1 Rundeck

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

With that plugin, you can execute commands on node registered in *Rudder*, taking advantage of the groups defined for you policies.

14.2.1.2 Ansible

There is an inventory plugin for Ansible that makes possible to use *Rudder* inventory (including groups, nodes, group ids, node ids, and node properties) as inventory for Ansible, for example for orchestration tasks on your platform. An inventory in Ansible is the list of managed nodes, their groups and some pre-defined variables. The *Rudder* plugin is part of Ansible as of version 2.0 (but also works with previous versions).

You need to download the rudder.py and rudder.ini files, then you have to edit `rudder.ini` to fill (at least):

- Your *Rudder* server URL
- A valid API token

Then you can start using it with the `-i rudder.py` option (to the Ansible command-line). The plugin defines:

- An Ansible group for each *Rudder* group, with a group variable named `rudder_group_id` that contains the uuid of the group
- An host variable named `rudder_node_id` that contains the uuid of the node
- Host variables containing the *Rudder* node properties

You can then use them in the configuration, for example:

```
ansible -i rudder.py All_nodes_managed_by_root_policy_server -a "echo {{rudder_node_id}} {{ ↩
    rudder_group_id}} {{node_property}} {{node_property.key}}"
```

Will try to connect over SSH to all nodes managed by your *Rudder* server and display the given information.

You can defined the `ansible_host`, `ansible_user` and `ansible_port` node properties to control the way Ansible connects to the nodes.

14.2.1.3 iTop

iTop is an Open Source CMDB solution. It allows to describe you IT services and analyse impact of problems.

There is a prototype integration of iTop and *Rudder* which allows iTop to *Rudder* as a source of information about the server content (inventory) and current compliance level. With that integration, you can see in real time in your CMDB when a server managed by *Rudder* is drifting away from its expecting configuration, and use iTop to understand the impact of such a drift for your IT services.

14.2.2 Integrate Rudder thanks to its APIs

All the above plugins are using *Rudder* APIs under the hood to operate or get data from *Rudder*s. *Rudder* APIs are as powerful as the UI, and anything that can be done through the main graphical interface can also be scripted away with the APIs.

The documentation provided on APIs is exhaustive, but here comes a summary of what can be done with them:

- accept, delete a node and manage its parameters,
- get information with a parametrable depth about node inventories,
- search for nodes,
- manage (create, update, delete) groups, directives, rules and parameters,
- interact with the *Techniques* library,
- get compliance details about a node or a rule, with a parameterized depth of information,
- manage change requests.

And of course, any plugin can provide new API endpoints, as is doing the data source plugin.

These API can also be used to automate *Rudder* action, like node acceptation or compliance export and archiving for nodes.

Chapter 15

Rudder Plugins

This chapter presents available plugins provides for *Rudder* and maintained by along with *Rudder*. They are available for each version of *Rudder* and updated if needed (for example in case of API change).

Plugins can be open-source or only available in binary format.

15.1 Rudder agent DSC

This plugins allows to manage *Windows* systems, using Microsoft Powershell DSC

15.1.1 Install Windows DSC plugin on the server

15.1.1.1 Prerequisite

The *Windows* DSC plugin requires **zip** on the *Rudder* server, you need to install it prior to installing the plugin.

15.1.1.2 Installing and Upgrading

The installation and upgrade processes are exactly the same. Download the **rpkg** file, and run, on the *Rudder* server:

```
/opt/rudder/bin/rudder-pkg install-file rudder-plugin-dsc-<Rudder version>-<plugin version ↩
    >.rpkg
```

It will add:

- The ability to generate policies for *Windows Nodes*
- New generic methods in the technique editor
- New techniques

15.1.2 Install Windows DSC agent

The installation and upgrade processes are exactly the same.

15.1.2.1 Supported version of Microsoft Windows

The *Rudder* agent needs **PowerShell 4** or later, which is built-in on:

- *Windows* Server 2012 R2 and later

PowerShell 4 may also be installed on the following platforms, following this procedure: https://social.technet.microsoft.com/-wiki/contents/articles/20623.step-by-step-upgrading-the-powershell-version-4-on-2008-r2.aspx

- *Windows* Server 2008 R2
- *Windows* Server 2012

15.1.2.2 Desktop version of Microsoft Windows

There is no official support of *Rudder* agent on desktop versions of Microsoft *Windows*. However, the agent can be installed on the following platform:

- *Windows* 7 (you will need to upgrade to PowerShell 4 first, and activate WinRM)
- *Windows* 8 (you will need to upgrade to PowerShell 4 first, and activate WinRM)
- *Windows* 8.1
- *Windows* 10

Plase note that prior to the installation on *Windows* 7 and 8, you will need to install PowerShell 4 and make sure WinRM is activated with the following command:

```
Set-WSManQuickConfig DSC
```

Moreover, the *Windows* DSC agent comes without digital signature, you need to allow the unsigned source code execution on the *Windows* node. In some environment, this policy change can lead to security issues, please read the Microsoft *Windows* doc associated. This can be done in powershell by executing the following command:

```
Set-ExecutionPolicy RemoteSigned
```

15.1.2.3 Installation procedure

Download the **exe** file, and run, on your node:

```
rudder-agent-dsc-<Rudder version>-<plugin version>.exe
```

The installer will ask the IP address or DNS name of the policy server to use. If a policy server is already configured (for example during upgrade or an unattended installation), you can leave this field empty.

The installer will install the agent files and create the scheduled tasks to run the agent and the inventory. *Rudder* does not come as a *Windows* Service but as a scheduled PowerShell task, managed by **schtasks.exe**.

15.1.2.4 Unattended installation

For an automated unattended installation, you can pre-configure the policy server in the file:

```
C:\Program Files\Rudder\etc\policy-server.conf
```

Then the installer need to be executed with the following command:

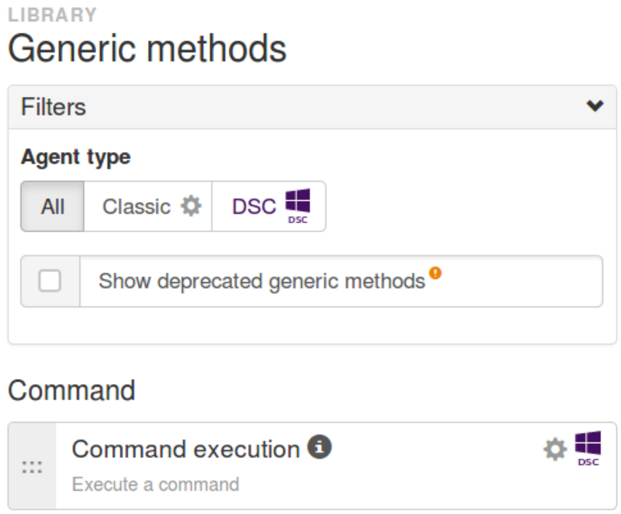
```
rudder-agent-dsc-<Rudder version>-<plugin version>.exe /S
```

This will install the agent in silent mode.

15.1.3 Technique editor with DSC

DSC Generic Methods are shipped with the *Rudder* dsc plugin. Some are specific for *Windows* managed systems (like the Registry management), and the others are the DSC version of existing generic methods.

A filter is available in the *Technique* Editor to select either all generic methods, generic methods available for classic agent, and generic method available for DSC agent, so that you can choose relevant methods for the type of nodes you need to manage



LIBRARY

Generic methods

Filters

Agent type

All | Classic | DSC

□ Show deprecated generic methods

Command

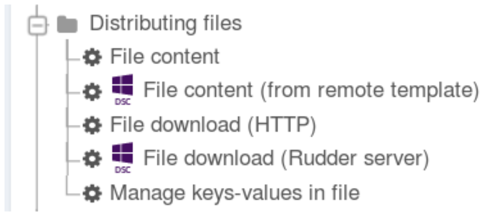
Command execution

Execute a command

15.1.4 DSC Techniques

Techniques compatible with DSC agent appear, in the *Directives* and *Techniques* trees, with a DSC symbol, as shown in the screenshot below.

Unfortunately, not all *Techniques* are compatible with DSC agent, as some are deprecated, or some will be completely rewritten, but the coverage is increasing regularly.



Distributing files
- File content
- File content (from remote template)
- File download (HTTP)
- File download (Rudder server)
- Manage keys-values in file

15.1.5 DSC Agent CLI

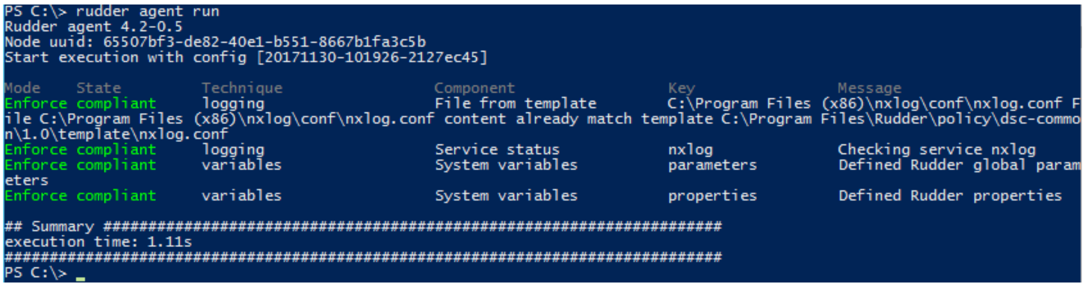
The *Rudder* agent CLI is available as a Powershell module, by running, in a Powershell terminal

```
rudder agent <action>
```

where action can be one of the following

- disable: Disable the agent, and prevent its execution
- enable: Enable the agent
- info: Show information about the agent and the node (hostname, *Rudder* ID, policy server, etc)
- inventory: Generate an inventory, and send it to the server
- run: Run the agent (see example output below)

- update: Update agent policy from the *Rudder Server*
- version: Show the version of the DSC *Rudder* agent



```
PS C:\> rudder agent run
Rudder agent 4.2-0.5
Node uuid: 65507bf3-de82-40e1-b551-8667b1fa3c5b
Start execution with config [20171130-101926-2127ec45]

Mode    State           Technique                  Component              Key                  Message
Enforce compliant       logging                    File from template     C:\Program Files (x86)\nxlog\conf\nxlog.conf F
ile C:\Program Files (x86)\nxlog\conf\nxlog.conf content already match template C:\Program Files\Rudder\policy\dsc-commo
n\1.0\template\nxlog.conf
Enforce compliant       logging                    Service status         nxlog                Checking service nxlog
Enforce compliant       variables                  System variables       parameters           Defined Rudder global param
eters
Enforce compliant       variables                  System variables       properties           Defined Rudder properties

## Summary #####################################################################
execution time: 1.11s
################################################################################
PS C:\> _
```

15.1.5.1 Agent logs

Rudder logs are visible in the output of the agent. You can get more details about what is done with the `-Verbose` option:

```
rudder agent run -v
```

You can also explore all agent logs (including those from unattended runs) in the *Windows* Event Viewer. Before *Windows* plugin version 4.2-1.6 *Rudder* used the windows system eventlog and was logging in the ***Windows* Logs** → **Application** view, with the ***Rudder*** source and the **101** Event ID.

Since the *Windows* plugin version 4.2-1.6 *Rudder* will report in a dedicated windows journal named *Rudder* and its logs are saved on different verbosity:

- **classic *Rudder* reports** will have the **Event ID 101**, they are the reports sent to the server.
- **Information logs** will have the **Event ID 102** and will only be local logs.

If you had an old plugin version installed *Rudder* will not try to install the new journal reference because it needs a complete reboot of the host system. See the last note on the Microsoft doc: https://msdn.microsoft.com/en-us/library/2awhba7a%28v=vs.110%29.as

If you want to change manually the *Rudder* eventlog use the following process, keep in mind that it will need a machine restart to avoid any reporting issues. First identify the current eventlog for *Rudder* by running in the powershell console

```
[System.Diagnostics.EventLog]::LogNameFromSourceName("Rudder", ".")
```

If it does not suit you, remove the *Rudder* source from it and create a new logger for *Rudder*

```
Remove-Eventlog -Source "Rudder"
New-Eventlog -Source "Rudder" -LogName "Rudder"
```

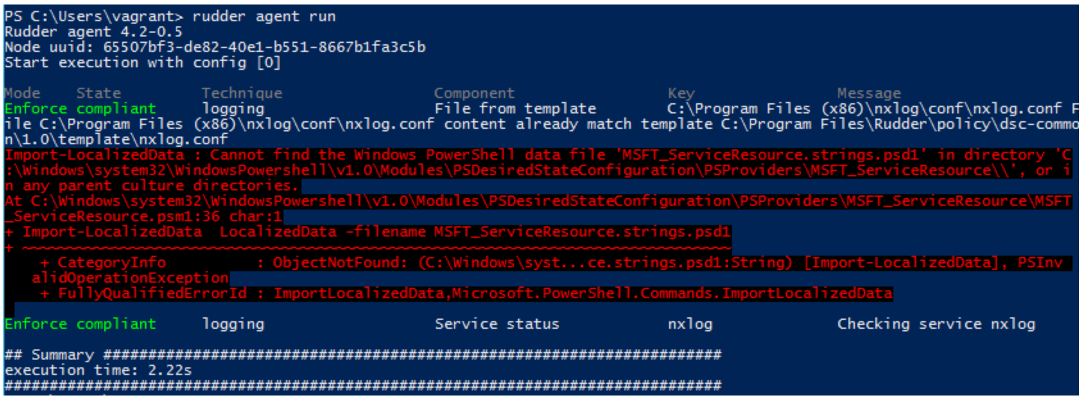
Then reboot the system.

15.1.6 Known issues

On the first run of the *Rudder* DSC agent CLI in a Powershell terminal, you may have the following error message:

```
Import-LocalizedData : Cannot find the Windows PowerShell data file 'MSFT_ServiceResource. ↩
    strings.psd1' in directory 'C:\Windows\system32\WindowsPowershell\v1.0\Modules\ ↩
    PSDesiredStateConfiguration\PSProviders\MSFT_ServiceResource\\', or in any parent  ↩
    culture directories.
```

This does not prevent the correct execution of the agent, and next runs in the same terminal will not exhibit the error



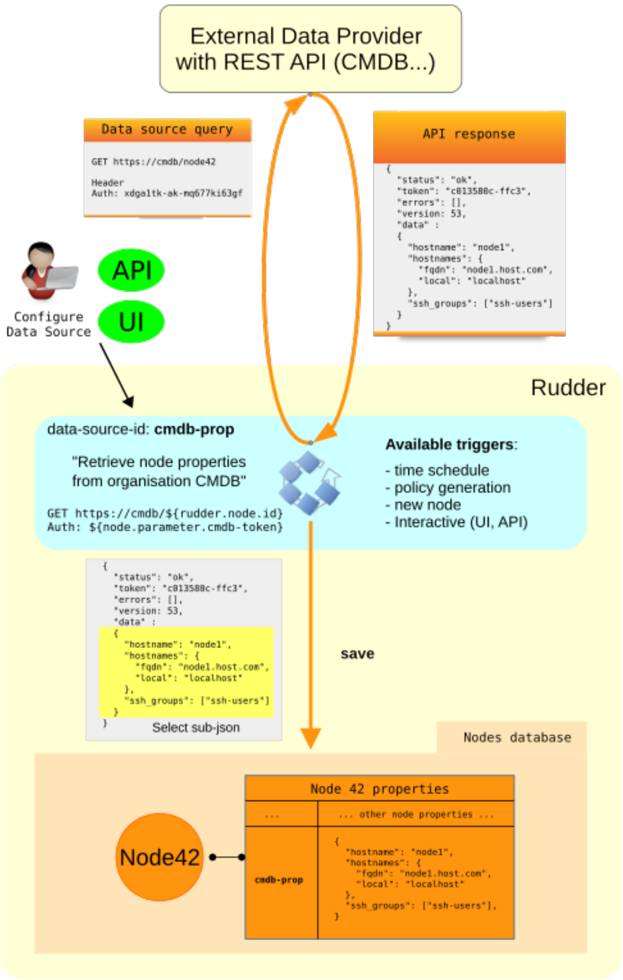
```
PS C:\Users\vagrant> rudder agent run
Rudder agent 4.2-0.5
Node uuid: 65507bf3-de82-40e1-b551-8667b1fa3c5b
Start execution with config [0]

Mode     State            Technique                  Component             Key                 Message
Enforce compliant         logging                    File from template    C:\Program Files (x86)\nxlog\conf\nxlog.conf F
ile C:\Program Files (x86)\nxlog\conf\nxlog.conf content already match template C:\Program Files\Rudder\policy\dsc-commo
n\1.0\template\nxlog.conf
Import-LocalizedData : Cannot find the Windows PowerShell data file 'MSFT_ServiceResource.strings.psd1' in directory 'C
:\Windows\system32\WindowsPowershell\v1.0\Modules\PSDesiredStateConfiguration\PSProviders\MSFT_ServiceResource\', or i
n any parent culture directories.
At C:\Windows\system32\WindowsPowershell\v1.0\Modules\PSDesiredStateConfiguration\PSProviders\MSFT_ServiceResource\MSFT
_ServiceResource.psm1:36 char:1
+ Import-LocalizedData  LocalizedData -filename MSFT_ServiceResource.strings.psd1
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (C:\Windows\syst...ce.strings.psd1:String) [Import-LocalizedData], PSInv
   alidOperationException
    + FullyQualifiedErrorId : ImportLocalizedData,Microsoft.PowerShell.Commands.ImportLocalizedData

Enforce compliant         logging                    Service status        nxlog               Checking service nxlog

## Summary #####################################################################
execution time: 2.22s
################################################################################
```

15.2 Node properties data sources

As explained in the chapter about node management, *Nodes* have properties that can be used to create groups or in techniques and directives parameters. These properties are key/value pairs, with values being a simple string or a well formed JSON document.

Rudder 4.1 introduces a new way to automatically import *Node* properties by defining data sources.

The following diagram explains the general working process of data source:



```
GET https://cmdb/node42

Header
Auth: xdga1tk-ak-mq677ki63gf
```

```
{
  "status": "ok",
  "token": "c813588c-ffc3",
  "errors": [],
  "version: 53,
  "data" :
  {
    "hostname": "node1",
    "hostnames": {
      "fqdn": "node1.host.com",
      "local": "localhost"
    },
    "ssh_groups": ["ssh-users"]
  }
}
```

```
GET https://cmdb/${rudder.node.id}
Auth: ${node.parameter.cmdb-token}
```

```
{
  "status": "ok",
  "token": "c813588c-ffc3",
  "errors": [],
  "version: 53,
  "data" :
  {
    "hostname": "node1",
    "hostnames": {
      "fqdn": "node1.host.com",
      "local": "localhost"
    },
    "ssh_groups": ["ssh-users"]
  }
}
```

```
{
  "hostname": "node1",
  "hostnames": {
    "fqdn": "node1.host.com",
    "local": "localhost"
  },
  "ssh_groups": ["ssh-users"],
}
```

As displayed, a data source provides a way for *Rudder* to query (when some conditions are met) a third party *REST API* to retrieve interesting properties for nodes and save them for a given *Node* property key.

More precisely, there are three main sets of properties to define (by UI or via *Rudder* API) to configure a data source:

15.2.1 First set: data source description

The first set of properties allows to define an unique identifier for the data source, which will be used as the property key in node, along with a human readable name and description.

15.2.2 Second set: query configuration

The second set of properties allows to define how the third party *REST API* will be queried and the returned JSON response processed.

For now, we only support one query mode which is to do one HTTP query for each node. In the future, a mode where only one query is done to retrieve information for all nodes will be added.

For the query, you will define the HTTP method to use (GET or POST), what is the remote URL, if there are specific headers or query parameters to add.

In case a 404 error is returned, the corresponding node property is deleted (on that node). In case of a timeout or any other HTTP errors, this is considered a temporary problem, and the node property is left as is.

When a JSON document is returned, you can define a JSON path expression (cf https://github.com/jayway/JsonPath/) to select only a sub-part of the document as the actual data to use as a node property.

Finally, the resulting data is assigned to the node, using the key name defined in the data source configuration.

You can use *Rudder* variables expansion (`${rudder.node.xxx}`, `${rudder.parameter.xxx}`, `${node.properties[xxx]}`) in most of these configuration option: URL, headers, query parameters, JSON path. They will be replaced by their values for each node at the time the HTTP query is run.

15.2.3 Third set: query triggers

The last set of options allows to define when the data source should be queried.

For now, there are 3 available triggers:

- a scheduled one, allowing to periodically do the update,
- a trigger on policy generation, which allows to get a refresh of node properties before possibly using them in techniques or directives,
- a trigger on node acceptation, so that a new node immediately get a working set of properties (for example to join the correct dynamic groups).

In addition to these configured triggers, data sources can be interactively refreshed with a call to a *Rudder REST API* or via the web interface.

15.3 Node external reports

This plugin allows to add external, static documents in a new tab in node details.

With the plugin, you configure directories on *Rudder* server where are located node specific documents. A new tab is created on node details page that allows to download documents for the corresponding node.

15.3.1 Documents naming convention

Documents must be stored in configured directories with a naming convention that allows *Rudder* to find back what document corresponds to a given node.

For that, the document name must contains the node `UUID` or `hostname` in **lower** case. The exact naming pattern if defined in the `fileformat` configuration parameter, and the value `@@node@@` is used to denotate the place where the node `UUID` or `hostname` will be. Both `UUID` and `hostname` are tested when looking for a matching file for a node.

15.3.2 Plugin configuration

This plugin is configured with a configuration file. Any modification in the file will be reloaded immediately without the need to restart *Rudder*.

15.3.3 Configuration file location

The default location for the configuration file is `/opt/rudder/share/plugins/node-external-reports/node-external-reports.properties`.

In case you need to change the configuration file location, you need to start *Rudder* with the JVM option parameter `-Drudder.plugin.externalNodeInformation.config=/full/path/to/config/file`.

15.3.4 Configuration file format

Plugin file format is as follow:

```
plugin.node-external-reports.reports {

  01_first_report_type= {
    title=title to display in node page
    description=a description which go below the title
    dirname=/full/path/to/base/directory/for/these/reports
    filename="file-name-pattern-for-@@node@@-report.pdf"
    content-type=application/pdf
  }

  02_second_report_type = {}
```

Note that quotes are mandatory only when the value contains `@@` (so most likely only for the `filename` parameter).

- "01_first_report_type" is a unique key, used internaly (in logs for example) and for sorting reports display order in node page;
- "title" is the name of section in the external document tab;
- "dirname" is the base directory on the local file system where documents are stored;
- "description" is a description for what the document is about;
- "filename": the file template name to lookup the document name. `@@node@@` will be replaced by the node `hostname` or `UUID`.
- "content-type": the HTTP content type to use for the new page. It allows to direct what the browser will do (open a PDF viewer, display an HTML page, etc).

For example, if you gather HTML "security" reports, text monitoring one, and PDF compliance KPI for your nodes, the configuration file will look like:

```
plugin.node-external-reports.reports {

  01_security= {
    title=Security Report
    description=This report display pen test results
    dirname=/var/reports/security
    filename="report-@@node@@-sec.html"
    content-type=text/html
  }

  02_monitoring {
    title=Monitoring Report
    description=Monitoring information about the node
    dirname=/var/reports/monitoring
    filename="monitor-@@node@@.txt"
    content-type=text/plain
  }

  03_compliance {
    title=Third party compliance report
    description=Compliance reports from CMDB
    dirname=/var/reports/compliance
    filename="compliance-@@node@@.pdf"
    content-type=application/pdf
  }
}
```

And the content of `/var/reports/` will looks like:

```
/tmp/reports
&#x251c;&#x2500;&#x2500; compliance
&#x2502;   &#x251c;&#x2500;&#x2500; compliance-node34.china1.bigcorp.com.html
&#x2502;   &#x2502;   .....
&#x2502;~~ &#x2514;&#x2500;&#x2500; compliance-00000068-55a2-4b97-8529-5154cbb63a18.pdf
&#x251c;&#x2500;&#x2500; monitoring
&#x2502;   &#x251c;&#x2500;&#x2500; monitor-compliance-node34.china1.bigcorp.com.txt
&#x2502;   &#x2502;   .....
&#x2502;~~ &#x2514;&#x2500;&#x2500; monitor-00000068-55a2-4b97-8529-5154cbb63a18.txt
&#x2514;&#x2500;&#x2500; security
    &#x251c;&#x2500;&#x2500; report-node34.china1.bigcorp-sec.com.html
    &#x2502;   .....
    &#x2514;&#x2500;&#x2500; report-00000068-55a2-4b97-8529-5154cbb63a18-sec.html
```

Chapter 16

Reference

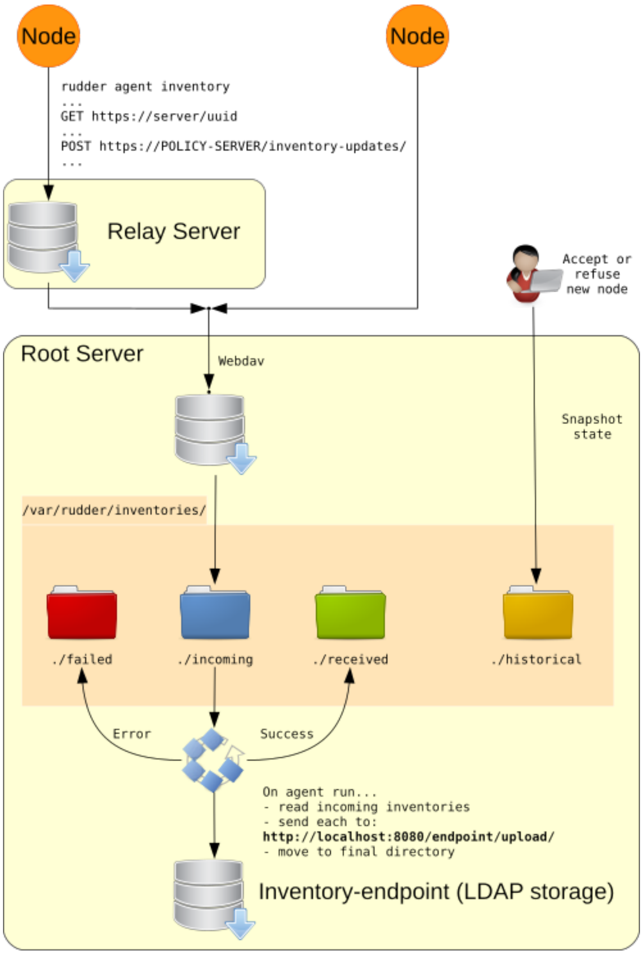
This chapter contains the reference *Rudder* configuration files

16.1 Inventory workflow, from nodes to Root server

One of the main information workflow in a *Rudder* managed system is the node's inventory one.

Node inventories are generated on nodes, are sent to the node policy server (be it a Relay or the Root server) up to the Root server, and stored in the *Rudder* database (technically an *LDAP* server), waiting for later use.

The goal of that section is to detail the different steps and explain how to spot and solve a problem on the inventory workflow. Following diagram sum up the whole process.



16.1.1 Processing inventories on node

Inventories are generated daily during an agent run in the 00:00-06:00 time frame window local to the node. The exact time is randomly spread on the time frame for a set of nodes, but each node will always keep the same time (modulo the exact time of the run).

User can request the generation and upload of inventory with the command:

```
$ rudder agent inventory
```

In details, generating inventory does:

- ask the node policy server for its UUID with an HTTP GET on `https://server/uuid`,
- generate an inventory by scanning the node hardware and software components,
- optionally make a digital signature of the generated inventory file,
- send file(s) to the node's policy server on `https://POLICY-SERVER/inventory-updates/`

The individual commands can be displayed with the `-i` option to `rudder agent inventory` command.

16.1.2 Processing inventories on relays

On the Relay server:

- the inventory is received by a `webdav` endpoint,
- the `webdav` service store the file in the folder `/var/rudder/inventories/incoming`
- on each agent runs, files in `/var/rudder/inventories/incoming` are forwarded to the Relay own policy server.

16.1.3 Processing inventories on root server

On the Root server, the start of the workflow is the same than on a relay:

- the inventory is received by a `webdav` endpoint,
- the `webdav` service store the file in the folder `/var/rudder/inventories/incoming`

Then, on each run, the agent:

- look for inventory / signature pairs:
  - inventories without a corresponding signature file are processed only if they are older than 2 minutes,
- POST the inventory or inventory+signature pair to the local API of "inventory-endpoint" application on `http://localhost:8080/endpoint/upload/`
- the API makes some quick checks on inventory (well formed, mandatory fields…) and :
  - if checks are OK, **ACCEPTS** (HTTP code `200`) the inventory,
  - if signature is configured to be mandatory and is missing, or if the signature is not valid, refuses with **UNAUTHORIZED** error (HTTP code `401`)
  - else fails with a **PRECONDITION FAILED** error (HTTP code `412`)
- on error, inventory file is moved to `/var/rudder/inventories/failed`,
- on success:
  - the inventory file is moved to `/var/rudder/inventories/received`,
  - in parallel, *inventory web* parses and updates *Rudder* database.

16.1.4 Queue of inventories waiting to be parsed

The *inventory endpoint* has a limited number of slot available for succesfully uploaded inventories to be queued waiting for parsing. That number can be configured in file `/opt/rudder/etc/inventory-web.properties`:

```
waiting.inventory.queue.size=50
```

Since *Rudder* 3.1.18 / 3.2.11 / 4.0.3, the number of currently waiting inventories can be obtained via a local *REST API* call to `http://localhost:8080/endpoint/api/info`:

```
$ curl http://localhost:8080/endpoint/api/info

{
  "queueMaxSize": 50,
  "queueFillCount": 50,
  "queueSaturated": true
}
```

16.2 Rudder Server data workflow

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

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

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

An extract of Rudder Techniques filesystem tree

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

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

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

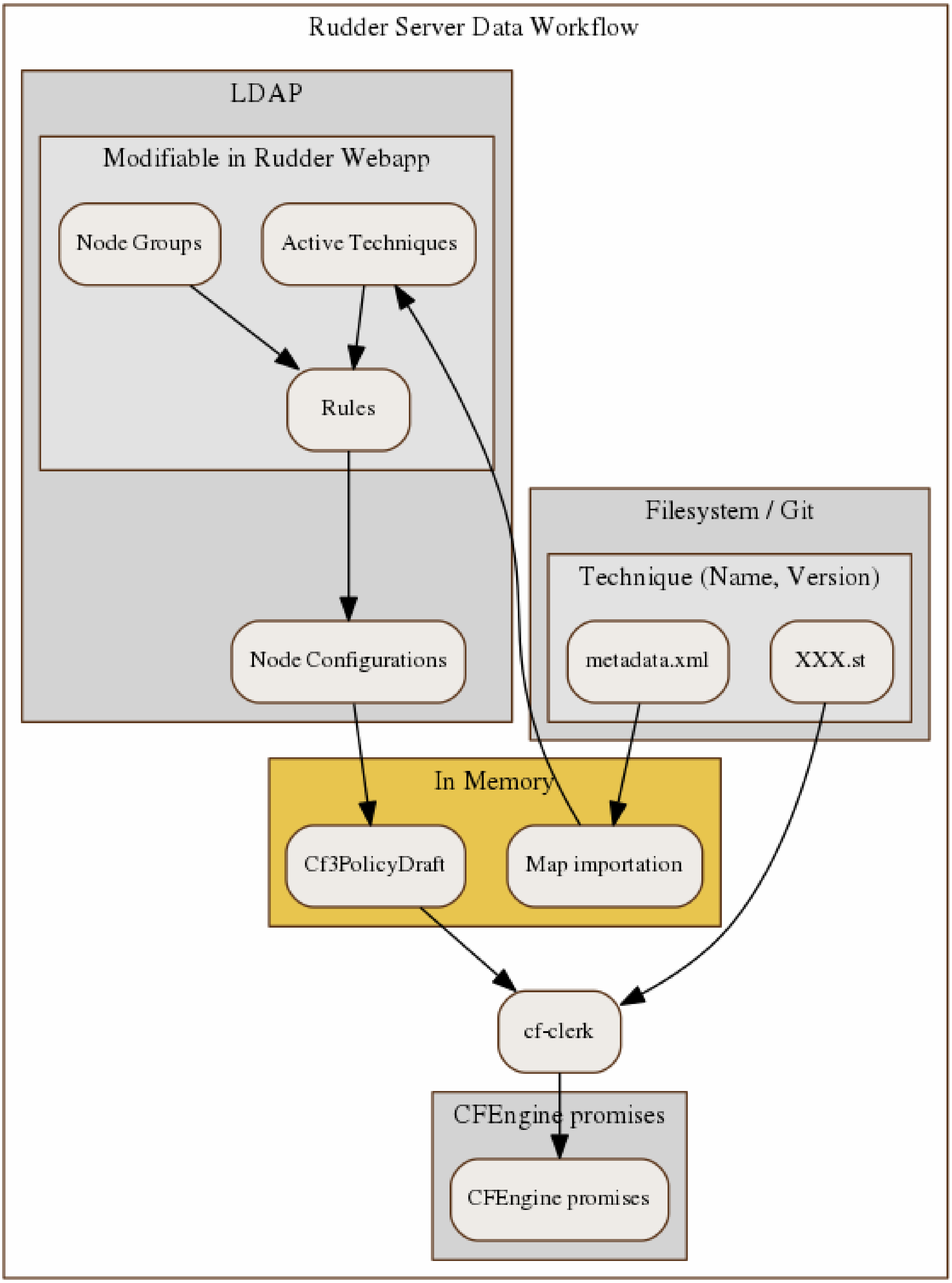


Figure 16.1: Rudder data workflow

16.3 Configuration files for Rudder Server

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

16.4 Rudder Agent workflow

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

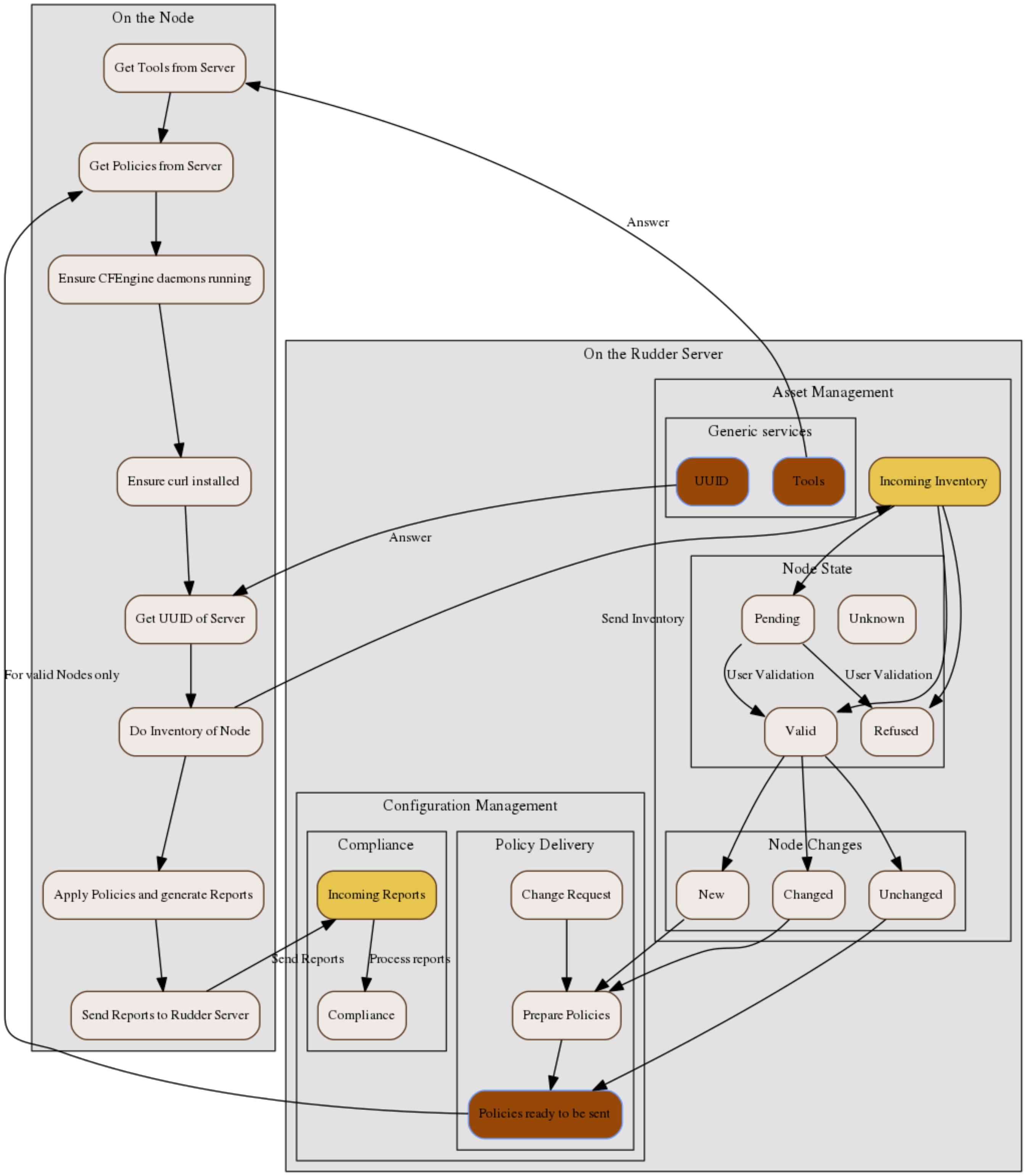


Figure 16.2: Rudder Agent workflow

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

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

16.4.1 Request data from Rudder Server

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

The agent then try to fetch new Applied 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.

16.4.2 Launch processes

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

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

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

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

16.4.3 Identify Rudder Root Server

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

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

16.4.4 Inventory

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

```
rudder agent inventory
```

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

16.4.5 Syslog

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

16.4.6 Apply Directives

Apply other policies and write reports locally.

16.5 Configuration files for a Node

- /etc/default/rudder-agent

16.6 Packages organization

16.6.1 Packages

Rudder components are distributed as a set of packages.

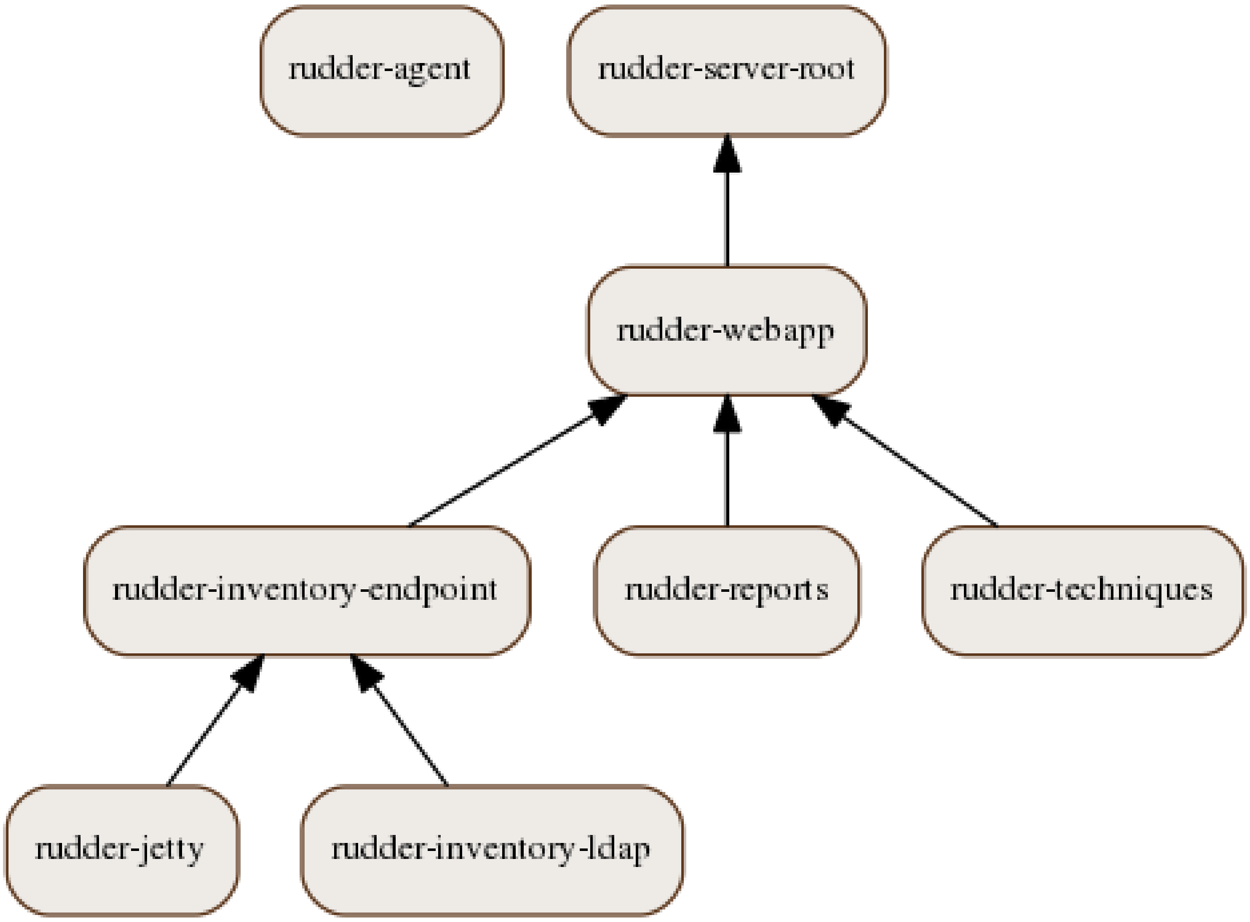


Figure 16.3: Rudder packages and their dependencies

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

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

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

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

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

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

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

- installs the *Rudder* configuration script:

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

- installs the initial promises for the Root Server in:

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

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

```
/etc/init.d/rudder
```

- 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
```

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

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

16.6.2 Software dependencies and third party components

The *Rudder* Web application requires the installation of Apache *2 httpd*, *JRE 7+*, 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*. *Rudder* 4.0 is tested for PostgreSQL 9.2 and higher. It still works with version 8.4 to 9.1, but not warranties are made that it will hold in the future. It is really recommanded to migrate to PostgreSQL 9.2 at least.

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

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

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

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

16.7 Building the Rudder Agent

16.7.1 Get source

Make sure you have network access and the git command.

Go to your build directory and checkout rudder-packages

```
cd /usr/src
git clone https://github.com/Normation/rudder-packages.git
cd rudder-packages
```

Choose the branch to build

```
# For branch 4.1 (branches before 4.1 are not supported)
git checkout branches/rudder/4.1
cd rudder-agent
```

Now choose one of the 3 next chapter, depending on your case: dpkg (debian-like package), rpm (redhat-like package) or other.

16.7.2 Build a dpkg package

Set the version to build:

- Update the debian/changelog file to make the first entry match the version you want to build.
- Edit the SOURCES/Makefile file and set the value of RUDDER_VERSION_TO_PACKAGE: see http://www.rudder-project.org/archives/ for a complete list of available versions.

Run the dpkg package builder:

```
dpkg-buildpackage
```

The package will be stored in the parent directory.

16.7.3 Build an rpm package

Set the version to build:

- Edit the SOURCES/Makefile file and set the value of RUDDER_VERSION_TO_PACKAGE: see http://www.rudder-project.org/archives/ for a complete list of available versions.

Run the rpm package builder:

```
# make sure you are in in rudder-agent, then
ln -s `pwd` /root/rpmbuild
rpmbuild -ba --define 'real_version 4.1.0' SPECS/*.spec
```

The package will be stored in RPMS/

16.7.4 Build an agent locally

Before building the agent, you must decide on some environment variables:

- RUDDER_VERSION_TO_PACKAGE: the version of the sources that will be used, see http://www.rudder-project.org/archives/- for a complete list. If a *rudder-sources* directory exists in SOURCES it will be used instead of downloading sources. The Variable still needs to be defined though.
- DESTDIR: where to put the installation, use / to install on the system and leave the default of ./target to prepare a package.
- USE_SYSTEM_OPENSSL: (default true), use system openssl (depends on libssl-dev) or build it with the agent.
- USE_SYSTEM_LMDB: (default false), use system lmdb (depends on liblmdb-dev) or build it with the agent.
- USE_SYSTEM_PCRE: (default true), use system pcre (depends on libpcre3-dev) or build it with the agent.
- USE_SYSTEM_PERL: (default false), use system perl (depends on perl) or build it with the agent.
- USE_SYSTEM_FUSION: (default false), use system fusion (depends on fusioninventory-agent), or build it with the agent. We advise you to use the *Rudder* version since it contains some patches.

```
# example
env="RUDDER_VERSION_TO_PACKAGE=4.1.0 DESTDIR=/ USE_SYSTEM_PERL=true"
make $env
make install $env
```

16.8 Generic methods

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

16.8.1 Command

16.8.1.1 command_execution

Execute a command

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **command_name**: Command name

Classes defined

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

16.8.1.2 command_execution_result

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.2 Condition

16.8.2.1 condition_from_command

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method executes a command, and defines a `${condition_prefix}_true` or a `${condition_prefix}_false` condition depending on the result of the command:

- If the exit code is in the `true_codes` list, this will produce a kept outcome class and a `${condition_prefix}_true` condition
- If the exit code is in the `false_codes` list, this will produce a repaired outcome class and a `${condition_prefix}_false` condition
- If the exit code is not in the list, or if the command is not present, this will produce an error outcome class and no classes with `${condition_prefix}`

The created condition is global to the agent.

16.8.2.2 Example

If you run a command `/bin/check_network_status` that output code 0, 1 or 2 in case of correct configuration, and 18 or 52 in case of invalid configuration, and you want to get this define a condition based on this command, you can use the following policy

```
condition_from_command("network_correctly_defined", "/bin/check_network_status",  ↩
    "0,1,2", "18,52")
```

- If the command exits 0, 1 or 2, then it will define the following conditions `network_correctly_defined_true`, `condition_from_command_network_correctly_defined_kept`, `condition_from_command_network_correctly_defined_reached`
- If the command exits 18, 52, then it will define the following conditions `network_correctly_defined_false`, `condition_from_command_network_correctly_defined_kept`, `condition_from_command_network_correctly_defined_reached`
- If the command exits any other code, then it will define the following conditions `condition_from_command_network_correctly_defined_error`, `condition_from_command_network_correctly_defined_reached`
- Finally, if the command is not present on the node, it will define `condition_from_command_network_correctly_defined_error`, `condition_from_command_network_correctly_defined_reached`

Parameters

- **condition_prefix**: The condition name
- **command**: The command to run
- **true_codes**: List of codes that produce a true status separated with commas (ex: 1,2,5)
- **false_codes**: List of codes that produce a false status separated with commas (ex: 3,4,6)

Classes defined

```
condition_from_command_${condition_prefix}_{kept, repaired, not_ok, reached}
```

16.8.2.3 condition_from_expression

Create a new condition class

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method evaluates an expression, and produces a `${condition_prefix}_true` or a `${condition_prefix}_false` condition depending on the result on the expression:

- If the expression results in a "defined" state, this will produce a kept outcome class and a `${condition_prefix}_true` condition
- If the expression results in an "undefined" state, this will produce a kept outcome class and a `${condition_prefix}_false` condition

Calling this method with a condition expression transforms a complex expression into a single class condition.

The created condition is global to the agent.

16.8.2.4 Example

If you want to check if a condition evaluates to true, like checking that you are on Monday, 2am, on RedHat systems, you can use the following policy

```
condition_from_expression("backup_time", "Monday.redhat.Hr02")
```

- If the system is a RedHat like system, on Monday, at 2am, then it will define the following conditions `backup_time_true`, `condition_from_expression_backup_time_kept`, `condition_from_expression_backup_time_reached`
- If the system not a RedHat like system, or it's not Monday, or it's not 2am, then it will define the following conditions `backup_time_false`, `condition_from_expression_backup_time_kept`, `condition_from_expression_backup_time_reached`
- If the condition is invalid (cannot be parsed), it will define only `condition_from_expression_backup_time_kept`, `condition_from_expression_backup_time_reached`

Parameters

- **condition_prefix**: The condition prefix
- **condition_expression**: The expression evaluated to create the condition (use *any* to always evaluate to true)

Classes defined

```
condition_from_expression_${condition_prefix}_{kept, repaired, not_ok, reached}
```

16.8.2.5 condition_from_expression_persistent

Create a new condition class that persists across runs

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method evaluates an expression, and produces a `${condition_prefix}_true` or a `${condition_prefix}_false` condition depending on the result on the expression, that lasts for the `duration` time:

- If the expression results in a "defined" state, this will produce a kept outcome class and a `${condition_prefix}_true` condition
- If the expression results in an "undefined" state, this will produce a kept outcome class and a `${condition_prefix}_false` condition

Calling this method with a condition expression transforms a complex expression into a single class condition.

The created condition is global to the agent and is persisted across runs. The persistence duration is controlled using `${duration}`; it defines for how long the resulting condition will be defined (in minutes). Note that there is no way to persist indefinitely.

16.8.2.6 Example

If you want to check if a condition evaluates to true, like checking that you are on Monday, 2am, on RedHat systems, and make it last one hour you can use the following policy

```
condition_from_expression_persistent_("backup_time", "Monday.redhat.Hr02", "60")
```

- If the system is a RedHat like system, on Monday, at 2am, then it will define the following conditions `backup_time_true`, `condition_from_expression_persistent_backup_time_kept`, `condition_from_expression_persistent_backup_time_reached`
- If the system not a RedHat like system, or it's not Monday, or it's not 2am, then it will define the following conditions `backup_time_false`, `condition_from_expression_persistent_backup_time_kept`, `condition_from_expression_persistent_backup_time_reached`
- If the condition is invalid (cannot be parsed), it will define only `condition_from_expression_persistent_backup_time_kept`, `condition_from_expression_persistent_backup_time_reached`

Parameters

- **condition_prefix**: The condition prefix
- **condition_expression**: The expression evaluated to create the condition (use *any* to always evaluate to true)
- **duration**: The persistence suffix in minutes

Classes defined

```
condition_from_expression_persistent_${condition_prefix}_{kept, repaired, not_ok, ↩
    reached}
```

16.8.2.7 condition_from_variable_match

Test the content of a string variable

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Test a variable content and create outcome classes depending on its value. This generic method will define a class `condition_from_variable_match_${condition_prefix}_{kept, repaired, error, reached}`.

If the variable is found and its content matches the given regex, this will produce a kept outcome condition and a `${condition_prefix}_true` condition If the variable can not be found or if its content does not match the given regex, this will produce an error outcome condition and a `${condition_prefix}_false` condition /! Regex for unix machine must be PCRE compatible and those for *Windows* agent must respect the .Net regex format.

Parameters

- **condition_prefix**: Prefix of the class (condition) generated
- **variable_name**: Complete name of the variable being tested, like my_prefix.my_variable
- **expected_match**: Regex to use to test if the variable content is compliant

Classes defined

```
condition_from_variable_match_${condition_prefix}_{kept, repaired, not_ok, reached ↵
    }
```

16.8.3 Directory

16.8.3.1 directory_absent

Ensure a directory’s absence

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

If `recursive` is false, only an empty directory can be deleted.

Parameters

- **target**: Directory to remove
- **recursive**: Should deletion be recursive, "true" or "false" (defaults to "false")

Classes defined

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

16.8.3.2 directory_check_exists

Checks if a directory exists

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.3.3 directory_create

Create a directory if it doesn't exist

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.4 Environment

16.8.4.1 environment_variable_present

Enforce an environment variable value. Caution, the new environment variable will not be usable by the agent until it is restarted

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **name**: Name of the environment variable
- **value**: Value of the environment variable

Classes defined

`environment_variable_present_${name}_{kept, repaired, not_ok, reached}`

16.8.5 File

16.8.5.1 file_check_FIFO_pipe

Checks if a file exists and is a FIFO/Pipe

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.2 file_check_block_device

Checks if a file exists and is a block device

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.3 file_check_character_device

Checks if a file exists and is a character device

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.4 file_check_exists

Checks if a file exists

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.5 file_check_hardlink

Checks if two files are the same (hard links)

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name_1**: File name #1 (absolute path on the target node)
- **file_name_2**: File name #2 (absolute path on the target node)

Classes defined

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

16.8.5.6 file_check_regular

Checks if a file exists and is a regular file

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.7 file_check_socket

Checks if a file exists and is a socket

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.8 file_check_symlink

Checks if a file exists and is a symlink

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file_name**: File name (absolute path on the target node)

Classes defined

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

16.8.5.9 file_check_symlinkto

Checks if first file is symlink to second file

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.5.10 file_copy_from_local_source

Ensure that a file or directory is copied from a local source

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **source**: Source file (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.11 file_copy_from_local_source_recursion

Ensure that a file or directory is copied from a local source

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.5.12 file_copy_from_local_source_with_check

Ensure that a file or directory is copied from a local source if a check command succeeds

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method is a conditionnal file copy.

It allows comparing the source and destination, and if they are different, call a command with the source file path as argument, and only update the destination if the commands succeeds (i.e. returns a code included in rc_ok).

16.8.5.13 Examples

```
# To copy a configuration file only if it passes a config test:
file_copy_from_local_source_with_check("/tmp/program.conf", "/etc/program.conf", " ↩
    program --config-test", "0");
```

This will:

- Compare `/tmp/program.conf` and `/etc/program.conf`, and return `kept` if files are the same
- If not, it will execute `program --config-test "/tmp/program.conf"` and check the return code
- If it is one of the `rc_ok` codes, it will copy `/tmp/program.conf` into `/etc/program.conf` and return a repaired
- If not, it will return an error

Parameters

- **source**: Source file (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)
- **check_command**: Command to run, it will get the source path as argument
- **rc_ok**: Return codes to be considered as valid, separated by a comma (default is 0)

Classes defined

```
file_copy_from_local_source_with_check_${destination}_{kept, repaired, not_ok,  ↩
    reached}
```

16.8.5.14 file_copy_from_remote_source

Ensure that a file or directory is copied from a policy server

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

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

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

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

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

Parameters

- **source**: Source file (absolute path on the policy server)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.15 file_copy_from_remote_source_recursion

Ensure that a file or directory is copied from a policy server

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

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

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

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

Parameters

- **source**: Source file (absolute path on the policy server)
- **destination**: Destination file (absolute path on the target node)
- **recursion**: Recursion depth to enforce for this path (0, 1, 2, . . . , inf)

Classes defined

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

16.8.5.16 file_create

Create a file if it doesn't exist

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **target**: File to create (absolute path on the target node)

Classes defined

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

16.8.5.17 file_create_symlink

Create a symlink at a destination path and pointing to a source target except if a file or directory already exists.

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **source**: Source file (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.18 file_create_symlink_enforce

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **source**: Source file (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)
- **enforce**: Force symlink if file already exist (true or false)

Classes defined

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

16.8.5.19 file_create_symlink_force

Create a symlink at a destination path and pointing to a source target even if a file or directory already exists.

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **source**: Source file (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.20 file_download

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

- **source**: URL to download from
- **destination**: File destination (absolute path on the target node)

Classes defined

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

16.8.5.21 file_enforce_content

Enforce the content of a file

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **file**: File name to edit (absolute path on the target node)
- **lines**: Line(s) to add in the file - if lines is a list, please use @{lines} to pass the iterator rather than iterating over each values
- **enforce**: Enforce the file to contain only line(s) defined (true or false)

Classes defined

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

16.8.5.22 file_ensure_block_in_section

Ensure that a section contains exactly a text block

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **file**: File name to edit (absolute path on the target node)
- **section_start**: Start of the section
- **section_end**: End of the section
- **block**: Block representing the content of the section

Classes defined

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

16.8.5.23 file_ensure_block_present

Ensure that a text block is present in a specific location

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **file**: File name to edit (absolute path on the target node)
- **block**: Block(s) to add in the file

Classes defined

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

16.8.5.24 file_ensure_key_value

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file**: File name to edit (absolute path on the target node)
- **key**: Key to define
- **value**: Value to define
- **separator**: Separator between key and value, for example "=" or " " (without the quotes)

Classes defined

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

16.8.5.25 file_ensure_key_value_option

Ensure that the file contains a pair of "key separator value", with options on the spacing around the separator

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **file**: File name to edit (absolute path on the target node)
- **key**: Key to define
- **value**: Value to define
- **option**: Option for the spacing around the separator: strict, which prevent spacings (space or tabs) around separators, or lax which accepts any number of spaces around separators
- **separator**: Separator between key and value, for example "=" or " " (without the quotes)

Classes defined

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

16.8.5.26 file_ensure_key_value_parameter_in_list

Ensure that one parameter exists in a list of parameters, on one single line, in the right hand side of a key→values line

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Edit the file, and ensure it contains the defined parameter in the list of values on the right hand side of a key→values line. If the parameter is not there, it will be added at the end, separated by parameter_separator. Optionnaly, you can define leading and closing character to enclose the parameters If the key does not exist in the file, it will be added in the file, along with the parameter

16.8.5.27 Example

If you have an initial file (`/etc/default/grub`) containing

```
GRUB_CMDLINE_XEN="dom0_mem=16G"
```

To add parameter `dom0_max_vcpus=32` in the right hand side of the line, you'll need the following policy

```
file_ensure_key_value_parameter_in_list("/etc/default/grub", "GRUB_CMDLINE", "=",  ↩
    "dom0_max_vcpus=32", " ", "\"", "\"");
```

Parameters

- **file**: File name to edit (absolute path on the target node)
- **key**: Full key name
- **key_value_separator**: character used to separate key and value in a key-value line
- **parameter**: String representing the sub-value to ensure is present in the list of parameters that form the value part of that line
- **parameter_separator**: Character used to separate parameters in the list
- **leading_char_separator**: leading character of the parameters
- **closing_char_separator**: closing character of the parameters

Classes defined

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

16.8.5.28 file_ensure_key_value_parameter_not_in_list

Ensure that a parameter doesn't exist in a list of parameters, on one single line, in the right hand side of a key→values line

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Edit the file, and ensure it does not contain the defined parameter in the list of values on the right hand side of a key→values line. If the parameter is there, it will be removed. Please note that the parameter can be a regular expression. It will also remove any whitespace character between the parameter and parameter_separator Optionnaly, you can define leading and closing character to enclose the parameters

16.8.5.29 Example

If you have an initial file (`/etc/default/grub`) containing

```
GRUB_CMDLINE_XEN="dom0_mem=16G dom0_max_vcpus=32"
```

To remove parameter `dom0_max_vcpus=32` in the right hand side of the line, you'll need the following policy

```
file_ensure_key_value_parameter_not_in_list("/etc/default/grub", "GRUB_CMDLINE",  ↩
    "=", "dom0_max_vcpus=32", " ", "\"", "\"");
```

Parameters

- **file**: File name to edit (absolute path on the target node)
- **key**: Full key name
- **key_value_separator**: character used to separate key and value in a key-value line
- **parameter_regex**: Regular expression matching the sub-value to ensure is not present in the list of parameters that form the value part of that line
- **parameter_separator**: Character used to separate parameters in the list
- **leading_char_separator**: leading character of the parameters
- **closing_char_separator**: closing character of the parameters

Classes defined

```
file_ensure_key_value_parameter_not_in_list_${file}_{kept, repaired, not_ok,  ↩
    reached}
```

16.8.5.30 file_ensure_key_value_present_in_ini_section

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.5.31 file_ensure_keys_values

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method ensures key-value pairs are present in a file.

16.8.5.32 Usage

This method will iterate over the key-value pairs in the dict, and:

- If the key is not defined in the destination, add the *key separator* + *value* line.
- If the key is already present in the file, replace the *key separator* + anything by *key* + *separator* + *value*

This method always ignores spaces and tabs when replacing (which means for example that `key = value` will match the = separator).

Keys are considered unique (to allow replacing the value), so you should use file_ensure_lines_present if you want to have multiple lines with the same key.

16.8.5.33 Example

If you have an initial file (`/etc/myfile.conf`) containing:

```
key1 = something
key3 = value3
```

To define key-value pairs, use the variable_dict or variable_dict_from_file methods.

For example, if you use the following content (stored in `/tmp/data.json`):

```
{
   "key1": "value1",
   "key2": "value2"
}
```

With the following policy:

```
# Define the `content` variable in the `configuration` prefix from the json file
variable_dict_from_file("configuration", "content", "/tmp/data.json")
# Enforce the presence of the key-value pairs
file_ensure_keys_values("/etc/myfile.conf", "configuration.content", " = ")
```

The destination file (`/etc/myfile.conf`) will contain:

```
key1 = value1
key3 = value3
key2 = value2
```

Parameters

- **file**: File name to edit (absolute path on the target node)
- **keys**: Name of the dict structure (without "${}") containing the keys (keys of the dict), and values to define (values of the dict)
- **separator**: Separator between key and value, for example "=" or " " (without the quotes)

Classes defined

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

16.8.5.34 file_ensure_line_present_in_ini_section

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.5.35 file_ensure_line_present_in_xml_tag

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.5.36 file_ensure_lines_absent

Ensure that a line is absent in a specific location

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **file**: File name to edit (absolute path on the target node)
- **lines**: Line(s) to remove in the file

Classes defined

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

16.8.5.37 file_ensure_lines_present

Ensure that one or more lines are present in a file

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **file**: File name to edit (absolute path on the target node)
- **lines**: Line(s) to add in the file

Classes defined

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

16.8.5.38 file_from_shared_folder

Ensure that a file or directory is copied from *Rudder* shared folder (/var/rudder/configuration-repository/shared-files)

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **source**: Source file (path relative to *Rudder* shared folder)
- **destination**: Destination file (absolute path on the target node)
- **hash_type**: Hash algorithm used to check if file is updated (md5, sha1, sha256). Only used on dsc agent, cfengine agent use it's own system for now.

Classes defined

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

16.8.5.39 file_from_string_mustache

Build a file from a mustache string

Compatible with nodes running *Rudder* 4.1 or higher.

Parameters

- **template**: String containing a template to be expanded
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.40 file_from_template

Build a file from a legacy *CFEngine* template

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See file_from_template_type for general documentation about templates usage.

Parameters

- **source_template**: Source file containing a template to be expanded (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.41 file_from_template_jinja2

Build a file from a jinja2 template

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See file_from_template_type for general documentation about templates usage.

This generic method will build a file from a jinja2 template using data (classes and variables) found in the execution context.

16.8.5.42 Setup

It requires to have the jinja2 python module installed on the node, it can usually be done in ncf with `package_present("python-jinja2", "", "", "")`.

Warning
If you are using a jinja2 version older than 2.7 trailing newlines will not be preserved in the destination file.

16.8.5.43 Syntax

Jinja2 is a powerful templating language, running in Python. The Jinja2 syntax reference documentation is http://jinja.pocoo.org/docs/dev/templates/ which will likely be useful, as Jinja2 is very rich and allows a lot more that what is explained here.

This section presents some simple cases that cover what can be done with mustache templating, and the way the agent data is provided to the templating engine.

The main specificity of jinja2 templating is the use of two root containers:

- `classes` to access currently defined classes
- `vars` to access all currently defined variables

Note: You can add comments in the template, that will not be rendered in the output file with `{# ... #}`.

You can extend the Jinja2 templating engine by adding custom FILTERS and TESTS in the script `/var/rudder/configuration-repository/ncf/10_ncf_internals/modules/templates/jinja2_custom.py`

For instance, to add a filter to upperstring a string and a test if a number is odd, you can create the file `/var/rudder/configuration-repository/ncf/10_ncf_internals/modules/extensions/jinja2_custom.py` on your *Rudder* server with the following content:

```
def upperstring(input):
    return input.upper()

def odd(value):
    return True if (value % 2) else False

FILTERS = {'upperstring': upperstring}
TESTS = {'odd': odd}
```

These filters and tests will be usable in your jinja templates automatically.

Conditions

To display content based on classes definition:

```
{% if classes.my_class is defined  %}
   display this if defined
{% endif %}
{% if not classes.my_class is defined %}
   display this if not defined
{% endif %}
```

Note: You cannot use class expressions here.

You can also use other tests, for example other bilt-in ones or those defined in `jinja2_custom.py`:

```
{% if vars.variable_prefix.my_number is odd  %}
   display if my_number is odd
{% endif %}
```

Scalar variables

Here is how to display a scalar variable value (integer, string, …), if you have defined `variable_string("variable_prefix", "my_variable", "my_value")`:

```
{{ vars.variable_prefix.my_variable }}
```

You can also modify what is displayed by using filters. The built-in filters can be extended in `jinja2_custom.py`:

```
{{ vars.variable_prefix.my_variable | upperstring }}
```

Will display the variable in uppercase.

Iteration

To iterate over a list, for example defined with:

```
variable_iterator("variable_prefix", "iterator_name", "a,b,c", ",")
```

Use the following file:

```
{% for item in vars.variable_prefix.iterator_name %}
{{ item }} is the current iterator_name value
{% endfor %}
```

Which will be expanded as:

```
a is the current iterator_name value
b is the current iterator_name value
c is the current iterator_name value
```

To iterate over a container defined by the following json file, loaded with `variable_dict_from_file("variable_prefix", "dict_name", "path")`:

```
{
   "hosts": [
       "host1",
       "host2"
   ],
   "files": [
       {"name": "file1", "path": "/path1", "users": [ "user1", "user11" ] },
       {"name": "file2", "path": "/path2", "users": [ "user2" ] }
   ],
   "properties": {
       "prop1": "value1",
       "prop2": "value2"
   }
}
```

Use the following template:

```
{% for item in vars.variable_prefix.dict_name.hosts %}
{{ item }} is the current hosts value
{% endfor %}

# will display the name and path of the current file
{% for file in vars.variable_prefix.dict_name.files %}
{{ file.name }}: {{ file.path }}
{% endfor %}

# will display the users list of each file
{% for file in vars.variable_prefix.dict_name.files %}
{{ file.name }}: {{ file.users|join(’ ’) }}
{% endfor %}


# will display the current properties key/value pair
{% for key, value in vars.variable_prefix.dict_name.properties %}
{{ key }} -> {{ value }}
{% endfor %}
```

Which will be expanded as:

```
host1 is the current hosts value
host2 is the current hosts value

# will display the name and path of the current file
file1: /path1
file2: /path2

# will display the users list of each file
file1: user1 user11
file2: user2
```

```
# will display the current properties key/value pair
prop1 -> value1
prop2 -> value2
```

Parameters

- **source_template**: Source file containing a template to be expanded (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.44 file_from_template_mustache

Build a file from a mustache template

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See file_from_template_type for general documentation about templates usage.

16.8.5.45 Syntax

Mustache is a logic-less templating language, available in a lot of languages, and used for file templating in *CFEngine*. The mustache syntax reference is https://mustache.github.io/mustache.5.html.

We will here describe the way to get agent data into a template. Ass explained in the general templating documentation, we can access various data in a mustache template.

The main specificity compared to standard mustache syntax of prefixes in all expanded values:

- `classes` to access classes
- `vars` to access all variables

Classes

Here is how to display content depending on classes definition:

```
{{#classes.my_class}}
   content when my_class is defined
{{/classes.my_class}}

{{^classes.my_class}}
   content when my_class is *not* defined
{{/classes.my_class}}
```

Note: You cannot use class expressions here.

Scalar variable

Here is how to display a scalar variable value (integer, string, . . .), if you have defined `variable_string("variable_prefix", "my_variable", "my_value")`:

`{{{vars.variable_prefix.my_variable}}}`

We use the triple `{{{ }}}` to avoid escaping html entities.

Iteration

Iteration is done using a syntax similar to scalar variables, but applied on container variables.

- Use `{{#vars.container}} content {{/vars.container}}` to iterate
- Use `{{{.}}}` for the current element value in iteration
- Use `{{{.key}}}` for the `key` value in current element
- Use `{{{@}}}` for the current element key in iteration

To iterate over a list, for example defined with:

```
variable_iterator("variable_prefix", "iterator_name", "a,b,c", ",")
```

Use the following file:

```
{{#vars.variable_prefix.iterator_name}}
{{{.}}} is the current iterator_name value
{{/vars.variable_prefix.iterator_name}}
```

Which will be expanded as:

```
a is the current iterator_name value
b is the current iterator_name value
c is the current iterator_name value
```

To iterate over a container defined by the following json file, loaded with `variable_dict_from_file("variable_prefix", "dict_name", "path")`:

```
{
   "hosts": [
       "host1",
       "host2"
   ],
   "files": [
       {"name": "file1", "path": "/path1", "users": [ "user1", "user11" ] },
       {"name": "file2", "path": "/path2", "users": [ "user2" ] }
   ],
   "properties": {
       "prop1": "value1",
       "prop2": "value2"
   }
}
```

Use the following template:

```
{{#vars.variable_prefix.dict_name.hosts}}
{{{.}}} is the current hosts value
{{/vars.variable_prefix.dict_name.hosts}}

# will display the name and path of the current file
{{#vars.variable_prefix.dict_name.files}}
{{{.name}}}: {{{.path}}}
{{/vars.variable_prefix.dict_name.files}}

# will display the users list of each file
```

```
{{#vars.variable_prefix.dict_name.files}}
{{{.name}}}:{{#users}} {{{.}}}{{/users}}
{{/vars.variable_prefix.dict_name.files}}


# will display the current properties key/value pair
{{#vars.variable_prefix.dict_name.properties}}
{{{@}}} -> {{{.}}}
{{/vars.variable_prefix.dict_name.properties}}
```

Which will be expanded as:

```
host1 is the current hosts value
host2 is the current hosts value

# will display the name and path of the current file
file1: /path1
file2: /path2

# will display the users list of each file
file1: user1 user11
file2: user2

# will display the current properties key/value pair
prop1 -> value1
prop2 -> value2
```

Note: Starting from *CFEngine* 3.7, you can use `{{#-top-}} ... {{/-top-}}` to iterate over the top level container.

Parameters

- **source_template**: Source file containing a template to be expanded (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)

Classes defined

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

16.8.5.46 file_from_template_type

Build a file from a template

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

These methods write a file based on a provided template and the data available to the agent.

16.8.5.47 Usage

To use these methods (`file_from_template_*`), you need to have:

- a template file
- data to fill this template

The template file should be somewhere on the local file system, so if you want to use a file shared from the policy server, you need to copy it first (using file_copy_from_remote_source).

It is common to use a specific folder to store those templates after copy, for example in `${sys.workdir}/templates/`.

The data that will be used while expanding the template is the data available in the agent at the time of expansion. That means:

- *CFEngine*'s sytem variables (`${sys.*}`, `...`) and classes (`linux`, ``)
- data defined during execution (outcome classes of generic methods, . . .)
- classes based on `condition_` generic methods
- data defined in ncf using `variable_*` generic methods, which allow for example to load data from local json or yaml files.

16.8.5.48 Template types

ncf currently supports three templating languages:

- *mustache* templates, which are documented in file_from_template_mustache
- *jinja2* templates, which are documented in file_from_template_jinja2
- *CFEngine* templates, which are a legacy implementation that is here for compatibility, and should not be used for new templates.

16.8.5.49 Example

Here is a complete example of templating usage:

The (basic) template file, present on the server in `/PATH_TO_MY_FILE/ntp.conf.mustache` (for syntax reference, see file_from_template_mustache):

```
{{#classes.linux}}
server {{{vars.configuration.ntp.hostname}}}
{{/classes.linux}}
{{^classes.linux}}
server hardcoded.server.example
{{/classes.linux}}
```

And on your local node in `/tmp/ntp.json`, the following json file:

```
{ "hostname": "my.hostname.example" }
```

And the following policy:

```
# Copy the file from the policy server
file_copy_from_remote_source("/PATH_TO_MY_FILE/ntp.conf.mustache", "${sys.workdir ↩
    }/templates/ntp.conf.mustache")
# Define the `ntp` varibale in the `configuration` prefix from the json file
variable_dict_from_file("configuration", "ntp", "/tmp/ntp.json")
# Expand yout template
file_from_template_type("${sys.workdir}/templates/ntp.conf.mustache", "/etc/ntp. ↩
    conf", "mustache")
# or
# file_from_template_mustache("${sys.workdir}/templates/ntp.conf.mustache", "/etc/ ↩
    ntp.conf")
```

The destination file will contain the expanded content, for example on a Linux node:

```
server my.hostname.example
```

Parameters

- **source_template**: Source file containing a template to be expanded (absolute path on the target node)
- **destination**: Destination file (absolute path on the target node)
- **template_type**: Template type (cfengine, jinja2 or mustache)

Classes defined

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

16.8.5.50 file_remove

Remove a file if it exists

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **target**: File to remove (absolute path on the target node)

Classes defined

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

16.8.5.51 file_replace_lines

Ensure that a line in a file is replaced by another one

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

You can replace lines in a files, based on regular expression and captured pattern

16.8.5.52 Syntax

The content to match in the file is a PCRE regular expression, unanchored that you can replace with the content of replacement.

Content can be captured in regular expression, and be reused with the notation `${match.1}` (for first matched content), `${match.2}` for second, etc, and the special captured group `${match.0}` for the whole text.

16.8.5.53 Example

Here is an example to remove enclosing specific tags

```
file_replace_lines("/PATH_TO_MY_FILE/file", "<my>(.*)<pattern>", "my ${match.1} ↩
    pattern")
```

Parameters

- **file**: File name to edit (absolute path on the target node)
- **line**: Line to match in the file
- **replacement**: Line to add in the file as a replacement

Classes defined

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

16.8.5.54 file_template_expand

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

WARNING: This generic method is deprecated. Use file_from_template instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.6 Group

16.8.6.1 group_absent

Make sure a group is absent

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **group**: *Group* name

Classes defined

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

16.8.6.2 group_present

Create a group

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **group**: *Group* name

Classes defined

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

16.8.7 Http

16.8.7.1 http_request_check_status_headers

Checks status of an HTTP URL

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.7.2 http_request_content_headers

Make an HTTP request with a specific header

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.8 Log

16.8.8.1 log_rudder

Logging output for *Rudder* reports

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **message**: The common part of the message to display
- **old_class_prefix**: The prefix of the class for different states (0.x version, empty to force new style logging only)
- **origin_class_prefix**: The prefix of the class for different states (1.x version)
- **args**: The arguments used to call the generic method (slist)

Classes defined

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

16.8.9 Logger

16.8.9.1 logger_rudder

Logging output for *Rudder* reports. This interface is for compatiblity with older generic methods and techniques, and is replaced by log_rudder.

WARNING: This generic method is deprecated. Use log_rudder instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **message**: The common part of the message to display
- **old_class_prefix**: The prefix of the class for different states (0.x version, empty to force new style logging only)

Classes defined

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

16.8.10 Package

16.8.10.1 package_absent

Enforce the absence of a package

Compatible with nodes running *Rudder* 4.1 or higher.

Usage

See package_state for documentation.

Parameters

- **name**: Name of the package
- **version**: Version of the package or "any" for any version (defaults to "any")
- **architecture**: Architecture of the package, can be an architecture name or "default" (defaults to "default")
- **provider**: Package provider to use, can be "yum", "apt", "slackpkg", "pkg" or "default" for system default package manager (defaults to "default")

Classes defined

```
package_absent_${name}_{kept, repaired, not_ok, reached}
```

16.8.10.2 package_check_installed

Verify if a package is installed in any version

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.10.3 package_install

Install or update a package in its latest version available

WARNING: This generic method is deprecated. Use package_present instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.10.4 package_install_version

Install or update a package in a specific version

WARNING: This generic method is deprecated. Use package_present instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.10.5 package_install_version_cmp

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

WARNING: This generic method is deprecated. Use package_present instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Example:

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

Parameters

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

Classes defined

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

16.8.10.6 package_install_version_cmp_update

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

WARNING: This generic method is deprecated. Use package_present instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Example:

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

Parameters

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

Classes defined

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

16.8.10.7 package_present

Enforce the presence of a package

Compatible with nodes running *Rudder* 4.1 or higher.

Usage

See package_state for documentation.

Parameters

- **name**: Name of the package, or path to a local package
- **version**: Version of the package, can be "latest" for latest version or "any" for any version (defaults to "any")
- **architecture**: Architecture of the package, can be an architecture name or "default" (defaults to "default")
- **provider**: Package provider to use, can be "yum", "apt", "slackpkg", "pkg" or "default" for system default package manager (defaults to "default")

Classes defined

```
package_present_${name}_{kept, repaired, not_ok, reached}
```

16.8.10.8 package_remove

Remove a package

WARNING: This generic method is deprecated. Use package_absent instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Example:

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

Parameters

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

Classes defined

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

16.8.10.9 package_state

Enforce the state of a package

Compatible with nodes running *Rudder* 4.1 or higher.

Usage

These methods manage packages using a package manager on the system.

`package_present` and `package_absent` use a new package implementation, different from `package_install_*`, `package_remove_*` and `package_verify_*`. It should be more reliable, and handle upgrades better. It is compatible though, and you can call generic methods from both implementations on the same host. The only drawback is that the agent will have to maintain double caches for package lists, which may cause a little unneeded overhead.

16.8.10.10 Setup

If you are using ncf inside *Rudder*, no specific setup is necessary.

If your are using ncf without *Rudder*, you need to call the `initialization` bundle before using package methods.

16.8.10.11 Package parameters

There is only one mandatory parameter, which is the package name to install. When it should be installed from a local package, you need to specify the full path to the package as name.

The version parameter allows specifying a version you want installed. It should be the complete versions string as used by the used package manager. This parameter allows two special values:

- *any* which is the default value, and is satisfied by any version of the given package
- *latest* which will ensure, at each run, that the package is at the latest available version.

The last parameter is the provider, which is documented in the next section.

You can use package_state_options to pass options to the underlying package manager (currently only with *apt* package manager).

16.8.10.12 Package providers

This method supports several package managers. You can specify the package manager you want to use or let the method choose the default for the local system.

The package providers include a caching system for package information. The package lists (installed, available and available updates) are only updated when the cache expires, or when an operation is made by the agent on packages.

Note: The implementation of package operations is done in scripts called modules, which you can find in `${sys.workdir}/modules/packages/`.

apt

This package provider uses *apt*/*dpkg* to manage packages on the system. *dpkg* will be used for all local actions, and *apt* is only needed to manage update and installation from a repository.

rpm

This package provider uses *yum*/*rpm* to manage packages on the system. *rpm* will be used for all local actions, and *yum* is only needed to manage update and installation from a repository.

It is able to downgrade packages when specifying an older version.

zypper

This package provider uses *zypper*/*rpm* to manage packages on the system. *rpm* will be used for all local actions, and *zypper* is only needed to manage update and installation from a repository.

Note: If the package version you want to install contains an epoch, you have to specify it in the version in the `epoch:version` form, like reported by `zypper info`.

slackpkg

This package provider uses Slackware's installpkg and upgradepkg tools to manage packages on the system

pkg

This package provider uses FreeBSD's *pkg* to manage packages on the system.

16.8.10.13 Examples

```
# To install postgresql in version 9.1 for x86_64 atchitecture
package_present("postgresql", "9.1", "x86_64", "");
# To ensure postgresql is always in the latest available version
package_present("postgresql", "latest", "", "");
# To ensure installing postgresql in any version
package_present("postgresql", "", "", "");
# To ensure installing postgresql in any version, forcing the yum provider
package_present("postgresql", "", "", "yum");
# To ensure installing postgresql from a local package
package_present("/tmp/postgresql-9.1-1.x86_64.rpm", "", "", "");
# To remove postgresql
package_absent("postgresql", "", "", "");
```

See also : package_present, package_absent, package_state_options

Parameters

- **name**: Name of the package, or path to a local package if state is present
- **version**: Version of the package, can be "latest" for latest version or "any" for any version (defaults to "any")
- **architecture**: Architecture of the package, can be an architecture name or "default" (defaults to "default")
- **provider**: Package provider to use, can be "yum", "apt", "zypper", "slackpkg", "pkg" or "default" for system default package manager (defaults to "default")
- **state**: State of the package, can be "present" or "absent" (defaults to "present")

Classes defined

```
package_state_${name}_{kept, repaired, not_ok, reached}
```

16.8.10.14 package_state_options

Enforce the state of a package with options

Compatible with nodes running *Rudder* 4.1 or higher.

Usage

See package_state for documentation.

Parameters

- **name**: Name of the package, or path to a local package if state is present
- **version**: Version of the package, can be "latest" for latest version or "any" for any version (defaults to "any")
- **architecture**: Architecture of the package, can be an architecture name or "default" (defaults to "default")
- **provider**: Package provider to use, can be "yum", "apt", "zypper", "slackpkg", "pkg" or "default" for system default package manager (defaults to "default")
- **state**: State of the package, can be "present" or "absent" (defaults to "present")
- **options**: Options no pass to the package manager (defaults to empty)

Classes defined

```
package_state_options_${name}_{kept, repaired, not_ok, reached}
```

16.8.10.15 package_verify

Verify if a package is installed in its latest version available

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.10.16 package_verify_version

Verify if a package is installed in a specific version

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.11 Permissions

16.8.11.1 permissions

Set permissions on a file or directory (non recursively)

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.11.2 permissions_dirs

Verify if a directory has the right permissions non recursively

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.11.3 permissions_dirs_recurse

Verify if a directory has the right permissions recursively

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.11.4 permissions_recurse

Verify if a file or directory has the right permissions recursively

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.11.5 permissions_type_recursion

Ensure that a file or directory is present and has the right mode/owner/group

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.12 Registry

16.8.12.1 registry_entry_absent

This generic method checks that a registry entry does not exists

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **key**: Registry key (ie, HKLM:)
- **entry**: Registry entry name

Classes defined

```
registry_entry_absent_${entry}_{kept, repaired, not_ok, reached}
```

16.8.12.2 registry_entry_present

This generic method defines if a registry entry exists with the correct value

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **key**: Registry key (ie, HKLM:)
- **entry**: Registry entry
- **value**: Registry value
- **registryType**: Registry value type (String, ExpandString, MultiString, Dword, Qword)

Classes defined

```
registry_entry_present_${entry}_{kept, repaired, not_ok, reached}
```

16.8.12.3 registry_key_absent

This generic method checks that a registry key does not exists

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **key**: Registry key (ie, HKLM:)

Classes defined

```
registry_key_absent_${key}_{kept, repaired, not_ok, reached}
```

16.8.12.4 registry_key_present

This generic method checks that a registry key exists

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **key**: Registry key (ie, HKLM:)

Classes defined

```
registry_key_present_${key}_{kept, repaired, not_ok, reached}
```

16.8.13 Schedule

16.8.13.1 schedule_simple

Trigger a repaired outcome when a job should be run

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run

- **mode**: "nodups": avoid duplicate runs in the same period / "catchup": avoid duplicates and one or more run have been missed, run once before next period / "stateless": no check is done on past runs

Classes defined

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

16.8.13.2 schedule_simple_catchup

Trigger a repaired outcome when a job should be run (avoid losing a job)

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning If the agent run is skipped during the period, method tries to catchup the run on next agent run. If the agent run is skipped twice,, only one run is catched up. If the agent is run twice (for example from a manual run), the job is run only once.

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run

Classes defined

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

16.8.13.3 schedule_simple_nodups

Trigger a repaired outcome when a job should be run (avoid running twice)

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning If the agent is run twice (for example from a manual run), the jo is run only once. However if the agent run is skipped during the period, the job is never run.

Parameters

- **job_id**: A string to identify this job

- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run

Classes defined

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

16.8.13.4 schedule_simple_stateless

Trigger a repaired outcome when a job should be run (without checks)

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This bundle will define a class `schedule_simple_${job_id}_{kept,repaired,not_ok,ok,reached}` * _ok or _kept for when there is nothing to do * _repaired if the job should run * _not_ok and _reached have their usual meaning No effort is done to check if a run has already been done for this period or not. If the agent is run twice, the job will be run twice, and if the agent is not run, the job will no be run.

Parameters

- **job_id**: A string to identify this job
- **agent_periodicity**: How often you run the agent in minutes
- **max_execution_delay_minutes**: On how many minutes you want to spread the job
- **max_execution_delay_hours**: On how many hours you want to spread the job
- **start_on_minutes**: At which minute should be the first run
- **start_on_hours**: At which hour should be the first run
- **start_on_day_of_week**: At which day of week should be the first run
- **periodicity_minutes**: How often should the job run
- **periodicity_hours**: How often should the job run
- **periodicity_days**: How often should the job run

Classes defined

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

16.8.14 Service

16.8.14.1 service_action

Trigger an action on a service using the approriate tool

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

The `service_*` methods manage the services running on the system.

16.8.14.2 Parameters

Service name

The name of the service is the name understood by the service manager, except for the `is-active-process` action, where it is the regex to match against the running processes list.

Action

The action is the name of an action to run on the given service. The following actions can be used:

- `start`
- `stop`
- `restart`
- `reload` (or `refresh`)
- `is-active` (or `status`)
- `is-active-process` (in this case, the "service" parameter is the regex to match againt process list)
- `enable`
- `disable`
- `is-enabled`

Other actions may also be used, depending on the selected service manager.

16.8.14.3 Implementation

These methods will detect the method to use according to the platform. You can run the methods with an `info` verbosity level to see which service manager will be used for a given action.

Warning
Due to compatibility issues when mixing calls to systemctl and service/init.d, when an init script exists, we will not use systemctl compatibility layer but directly service/init.d.

The supported service managers are:

- systemd (any unkown action will be passed directly)
- upstart
- smf (for Solaris)

- service command (for non-boot actions, any unkown action will be passed directly)
- /etc/init.d scripts (for non-boot actions, any unkown action will be passed directly)
- SRC (for AIX) (for non-boot actions)
- chkconfig (for boot actions)
- update-rc.d (for boot actions)
- chitab (for boot actions)
- links in /etc/rcX.d (for boot actions)
- *Windows* services

16.8.14.4 Examples

```
# To restart the apache2 service
service_action("apache2", "restart");
service_restart("apache2");
```

Parameters

- **service_name**: Name of the service
- **action**: Action to trigger on the service (start, stop, restart, reload, . . .)

Classes defined

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

16.8.14.5 service_check_disabled_at_boot

Check if a service is set to not start at boot using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.6 service_check_running

Check if a service is running using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **service_name**: Process name

Classes defined

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

16.8.14.7 service_check_running_ps

Check if a service is running using ps

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.8 service_check_started_at_boot

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.9 service_ensure_disabled_at_boot

Force a service not to be enabled at boot

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.10 service_ensure_running

Ensure that a service is running using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.11 service_ensure_running_path

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.12 service_ensure_started_at_boot

Force a service to be started at boot

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

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

Classes defined

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

16.8.14.13 service_ensure_stopped

Ensure that a service is stopped using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **service_name**: Service

Classes defined

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

16.8.14.14 service_reload

Reload a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See service_action for documentation.

Parameters

- **service_name**: Name of the service

Classes defined

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

16.8.14.15 service_restart

Restart a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See service_action for documentation.

Parameters

- **service_name**: Name of the service

Classes defined

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

16.8.14.16 service_restart_if

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

WARNING: This generic method is deprecated. Use a condition with service_restart instead.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See service_action for documentation.

Parameters

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

Classes defined

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

16.8.14.17 service_start

Start a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See service_action for documentation.

Parameters

- **service_name**: Name of the service

Classes defined

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

16.8.14.18 service_status

This generic method defines if service should run or be stopped

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **service_name**: Service name
- **status**: Desired state for the user - can be *Stopped* or *Running*

Classes defined

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

16.8.14.19 service_stop

Stop a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

See service_action for documentation.

Parameters

- **service_name**: Name of the service

Classes defined

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

16.8.15 Sharedfile

16.8.15.1 sharedfile_from_node

This method retreives a file shared from another *Rudder* node

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method retreives a file shared from a *Rudder* node using a unique file identifier.

The file will be downloaded using *CFEngine* protocol and copied into a new file. The destination path must be the complete absolute path of the destination file.

See sharedfile_to_node for a complete example.

INFO: Please note that this method must be used on an agent that is connected to *Rudder* relay or server (>=4.1)

Parameters

- **source_uuid**: which node to take the file from
- **file_id**: unique name that was used to identify the file on the sender
- **file_path**: where to put the file content

Classes defined

```
sharedfile_from_node_${file_id}_{kept, repaired, not_ok, reached}
```

16.8.15.2 sharedfile_to_node

This method shares a file with another *Rudder* node

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method shares a file with another *Rudder* node using a unique file identifier.

Read the *Rudder* documentation for a high level overview of file sharing between nodes.

The file will be kept on the policy server and transmitted to the destination node's policy server if it is different. It will be kept on this server for the destination node to download as long as it is not replaced by a new file with the same id or remove by expiration of the TTL.

16.8.15.3 Parameters

This section describes the generic method parameters.

16.8.15.4 target_uuid

The node you want to share this file with. The uuid of a node is visible in the *Nodes* details (in the Web interface) or by entering `rudder agent info` on the target node.

file_id

This is a name that will be used to identify the file in the target node. It should be unique and describe the file content.

file_path

The local absolute path of the file to share.

ttl

The TTL can be:

- A simple integer, in this case it is assumed to be a number of *seconds*
- A string including units indications, the possible units are:
- *days*, *day* or *d*
- *hours*, *hour*, or *h*
- *minutes*, *minute*, or *m*
- *seconds*, *second* or *s*

The ttl value can look like *1day 2hours 3minutes 4seconds* or can be abbreviated in the form *1d 2h 3m 4s*, or without spaces *1d2h3m4s* or any combination like *1day2h 3minute 4seconds* Any unit can be skipped, but the decreasing order needs to be respected.

file_id

This is a name that will be used to identify the file once stored on the server. It should be unique and describe the file content.

16.8.15.5 Example:

We have a node *A*, with uuid `2bf1afdc-6725-4d3d-96b8-9128d09d353c` which wants to share the `/srv/db/application.properties` with node *B* with uuid `73570beb-2d4a-43d2-8ffc-f84a6817849c`.

We want this file to stay available for one year for node *B* on its policy server.

The node *B* wants to download it into `/opt/application/etc/application.properties`.

They have to agree (i.e. it has to be defined in the policies of both nodes) on the id of the file, that will be used during the exchange, here it will be `application.properties`.

To share the file, node *A* will use:

```
sharedfile_to_node("73570beb-2d4a-43d2-8ffc-f84a6817849c", "application.properties ↩
    ", "/srv/db/application.properties", "356 days")
```

To download the file, node *B* will use sharedfile_from_node with:

```
sharedfile_from_node("2bf1afdc-6725-4d3d-96b8-9128d09d353c", "application. ↩
    properties", "/opt/application/etc/application.properties")
```

INFO: Please note that this method must be used on a *Rudder* agent (>=4.1) that is connected to *Rudder* relay or server (>=4.1)

Parameters

- **target_uuid**: which node to share the file with
- **file_id**: unique name that will be used to identify the file on the receiver
- **file_path**: path of the file to share
- **ttl**: time to keep the file on the policy server in seconds or in human readable form (see long description)

Classes defined

```
sharedfile_to_node_${file_id}_{kept, repaired, not_ok, reached}
```

16.8.16 User

16.8.16.1 user_absent

Remove a user

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

- **login**: User login

Classes defined

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

16.8.16.2 user_create

Create a user

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

Parameters

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

Classes defined

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

16.8.16.3 user_fullname

Define the fullname of the user, user must already exists.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user.

Parameters

- **login**: User's login
- **fullname**: User's fullname

Classes defined

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

16.8.16.4 user_home

Define the home of the user. User must already exists.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user, nor the home directory. entry example: /home/myuser The home given will be set, but not created.

Parameters

- **login**: User's login
- **home**: User's home

Classes defined

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

16.8.16.5 user_locked

Ensure the user is locked. User must already exist.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user. Note that locked accounts will be marked with "!" in /etc/shadow, which is equivalent to "*". To unlock a user, apply a user_password method.

Parameters

- **login**: User's login

Classes defined

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

16.8.16.6 user_password_clear

Ensure a user's password. as used in the UNIX /etc/shadow file.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

User must exists, password will appear in clear text in code. An empty password will lead to an error and be notified.

Parameters

- **login**: User login
- **password**: User clear password

Classes defined

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

16.8.16.7 user_password_hash

Ensure a user's password. Password must respect `$id$salt$hashed` format as used in the UNIX /etc/shadow file.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

User must exists, password must be pre-hashed. Does not handle empty password accounts. See UNIX /etc/shadow format. entry example: `$1$jp5rCMS4$mhvf4utonDubW5M00z0Ow0`

An empty password will lead to an error and be notified.

Parameters

- **login**: User login
- **password**: User hashed password

Classes defined

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

16.8.16.8 user_present

Ensure a user exists on the system.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user's home directory. Primary group will be created and set with default one. As in most UNIX system default behavior user creation will fail if a group with the user name already exists.

Parameters

- **login**: User login

Classes defined

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

16.8.16.9 user_primary_group

Define the primary group of the user. User must already exist.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user.

Parameters

- **login**: User's login
- **primary_group**: User's primary group

Classes defined

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

16.8.16.10 user_shell

Define the shell of the user. User must already exist.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user. entry example: /bin/false

Parameters

- **login**: User's login
- **shell**: User's shell

Classes defined

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

16.8.16.11 user_status

This generic method defines if user is present or absent

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **user**: User name
- **status**: Desired state for the user - can be *Present* or *Absent*

Classes defined

```
user_status_${user}_{kept, repaired, not_ok, reached}
```

16.8.16.12 user_uid

Define the uid of the user. User must already exists, uid must be non-allowed(unique).

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This method does not create the user.

Parameters

- **login**: User's login
- **uid**: User's uid

Classes defined

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

16.8.17 Variable

16.8.17.1 variable_dict

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

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

Classes defined

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

16.8.17.2 variable_dict_from_file

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

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **file_name**: The file name with JSON content

Classes defined

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

16.8.17.3 variable_dict_merge

Define a variable resulting of the merge of two other variables

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

The resulting variable will be the merge of the two parameters, which means it is built by:

- Taking the content of the first variable
- Adding the content of the second variable, and replacing the keys that were already there

It is only a one-level merge, and the value of the first-level key will be completely replaced by the merge.

This method will fail if one of the variables is not defined. See variable_dict_merge_tolerant if you want to allow one of the variables not to be defined.

16.8.18 Usage

If you have a `prefix.variable1` variable defined by:

```
{ "key1": "value1", "key2": "value2", "key3": { "keyx": "valuex" } }
```

And a `prefix.variable2` variable defined by:

```
{ "key1": "different", "key3": "value3", "key4": "value4" }
```

And that you use:

```
variablr_dict_merge("prefix", "variable3, "prefix.variable1", "prefix.variable2")
```

You will get a `prefix.variable3` variable containing:

```
{
  "key1": "different",
  "key2": "value2",
  "key3": "value3",
  "key4": "value4"
}
```

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **first_variable**: The first variable, which content will be overriden in the resulting variable if necessary (written in the form variable_prefix.variable_name)
- **second_variable**: The second variable, which content will override the first in the resulting variable if necessary (written in the form variable_prefix.variable_name)

Classes defined

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

16.8.18.1 variable_dict_merge_tolerant

Define a variable resulting of the merge of two other variables, allowing merging undefined variables

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

See variable_dict_merge for usage documentation. The only difference is that this method will not fail if one of the variables do not exist, and will return the other one. If both are undefined, the method will still fail.

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **first_variable**: The first variable, which content will be overriden in the resulting variable if necessary (written in the form variable_prefix.variable_name)
- **second_variable**: The second variable, which content will override the first in the resulting variable if necessary (written in the form variable_prefix.variable_name)

Classes defined

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

16.8.18.2 variable_iterator

Define a variable that will be automatically iterated over

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

The generated variable is a special variable (slist in cfengine speaking) that is automatically iterated over. When you call a generic method with this variable as a parameter, n calls will be made, one for each items of the variable. Note: there is a limit of 10000 items

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

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

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **value**: The variable content
- **separator**: Regular expression that is used to split the value into items (usually: ,)

Classes defined

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

16.8.18.3 variable_iterator_from_file

Define a variable that will be automatically iterated over

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

The generated variable is a special variable (slist in cfengine speaking) that is automatically iterated over. When you call a generic method with this variable as a parameter, n calls will be made, one for each items of the variable. Note: there is a limit of 10000 items Note: empty items are ignored

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

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

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **file_name**: The path to the file
- **separator_regex**: Regular expression that is used to split the value into items (usually:)
- **comments_regex**: Regular expression that is used to remove comments (usually: #.*?(?=))

Classes defined

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

16.8.18.4 variable_string

Define a variable from a string parameter

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

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

Classes defined

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

16.8.18.5 variable_string_default

Define a variable from another variable name, with a default value if undefined

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **source_variable**: The source variable name
- **default_value**: The default value to use if source_variable is not defined

Classes defined

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

16.8.18.6 variable_string_from_command

Define a variable from a command output

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **command**: The command to execute

Classes defined

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

16.8.18.7 variable_string_from_file

Define a variable from a file content

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **file_name**: The path of the file

Classes defined

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

16.8.18.8 variable_string_from_math_expression

Define a variable from a mathematical expression

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

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

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

16.9 Usage

This function will evaluate a mathematical expression that may contain variables and format the result according to the provided formatt string.

The formatting string uses the standard POSIX printf format.

16.9.1 Supported mathematical expressions

All the mathematical computations are done using floats.

The supported infix mathematical syntax, in order of precedence, is:

- `(` and `)` parentheses for grouping expressions
- `^` operator for exponentiation
- `*` and `/` operators for multiplication and division
- `%` operators for modulo operation
- + and − operators for addition and subtraction
- == "close enough" operator to tell if two expressions evaluate to the same number, with a tiny margin to tolerate floating point errors. It returns 1 or 0.
- >= "greater or close enough" operator with a tiny margin to tolerate floating point errors. It returns 1 or 0.
- > "greater than" operator. It returns 1 or 0.
- <= "less than or close enough" operator with a tiny margin to tolerate floating point errors. It returns 1 or 0.
- < "less than" operator. It returns 1 or 0.

The numbers can be in any format acceptable to the C `scanf` function with the `%lf` format specifier, followed by the `k`, `m`, `g`, `t`, or `p` SI units. So e.g. `-100` and `2.34m` are valid numbers.

In addition, the following constants are recognized:

- `e`: 2.7182818284590452354
- `log2e`: 1.4426950408889634074
- `log10e`: 0.43429448190325182765
- `ln2`: 0.69314718055994530942
- `ln10`: 2.30258509299404568402
- `pi`: 3.14159265358979323846
- `pi_2`: 1.57079632679489661923 (pi over 2)
- `pi_4`: 0.78539816339744830962 (pi over 4)
- `1_pi`: 0.31830988618379067154 (1 over pi)
- `2_pi`: 0.63661977236758134308 (2 over pi)
- `2_sqrtpi`: 1.12837916709551257390 (2 over square root of pi)
- `sqrt2`: 1.41421356237309504880 (square root of 2)
- `sqrt1_2`: 0.70710678118654752440 (square root of 1/2)

The following functions can be used, with parentheses:

- `ceil` and `floor`: the next highest or the previous highest integer
- `log10`, `log2`, `log`
- `sqrt`
- `sin`, `cos`, `tan`, `asin`, `acos`, `atan`
- `abs`: absolute value
- `step`: 0 if the argument is negative, 1 otherwise

16.9.2 Formatting options

The format field supports the following specifiers:

- `%d` for decimal integer
- `%x` for hexadecimal integer
- `%o` for octal integer
- `%f` for decimal floating point

You can use usual flags, width and precision syntax.

16.9.3 Examples

If you use:

```
variable_string("prefix", "var", "10");
variable_string_from_math_expression("prefix", "sum", "2.0+3.0", "%d");
variable_string_from_math_expression("prefix", "product", "3*${prefix.var}", "%d") ↩
    ;
```

The `prefix.sum` string variable will contain `5` and `prefix.product` will contain `30`.

Parameters

- **variable_prefix**: The prefix of the variable name
- **variable_name**: The variable to define, the full name will be variable_prefix.variable_name
- **expression**: The mathematical expression to evaluate
- **format**: The format string to use

Classes defined

```
variable_string_from_math_expression_${variable_name}_{kept, repaired, not_ok,  ↩
    reached}
```

16.9.4 Windows

16.9.4.1 windows_component_absent

Ensure that a specific windows component is absent from the system.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Ensure that a specific windows component is absent from the system.

Parameters

- **component**: *Windows* component name

Classes defined

```
windows_component_absent_${component}_{kept, repaired, not_ok, reached}
```

16.9.4.2 windows_component_present

Ensure that a specific windows component is present on the system.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Ensure that a specific windows component is present on the system.

Parameters

- **component**: *Windows* component name

Classes defined

```
windows_component_present_${component}_{kept, repaired, not_ok, reached}
```

16.9.4.3 windows_hotfix_absent

Ensure that a specific windows hotfix is absent from the system.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Ensure that a specific windows hotfix is absent from the system.

Parameters

- **hotfix**: *Windows* hotfix name (ex: KB4033369)

Classes defined

```
windows_hotfix_absent_${hotfix}_{kept, repaired, not_ok, reached}
```

16.9.4.4 windows_hotfix_present

Ensure that a specific windows hotfix is present from the system.

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

Ensure that a specific windows hotfix is present from the system.

Parameters

- **hotfix**: *Windows* hotfix name (ex: KB4033369)
- **package_path**: *Windows* hotfix package absolute path, can be a .msu archive or a .cab file

Classes defined

```
windows_hotfix_present_${hotfix}_{kept, repaired, not_ok, reached}
```

16.10 Man pages

16.10.1 rudder(8)

16.10.1.1 NAME

rudder - execute commands to control the *Rudder* configuration management tool.

16.10.1.2 SYNOPSIS

rudder *component* [-h] [-i|-v|-d] *command*

rudder *component* help

16.10.1.3 DESCRIPTION

A tool to trigger actions or get information about a running rudder-agent, whether on agent or server. It only targets administration actions, for all node configuration tasks you can use the rudder-cli tool.

16.10.1.4 OPTIONS

-h Print command-line syntax and command options.

-i Print general information.

-v Print detailed information.

-d Print all available information.

-c Do not colorize output.

16.10.1.5 COMMANDS

The commands below are listed by component.

16.10.1.6 agent

commands for rudder agent, run with **rudder agent** *command*

check check if rudder agent has no problem and is running properly. Check that rudder agent is working properly.

- generate missing UUID
- kill cfengine if there are too many processes
- run cfengine if its daemon is missing
- clean lock file if it is too big
- check and restore init files
- check that promises have been properly copied
  Options:
  -q: run the agent in quiet mode (display only error messages)
  -c: run the agent without color output

diff show diff between current file and the one before agent modification. This command will output file change in a diff format

Options:

-l: show diff from the given backup

-n: show diff from the nth backup before the last one

-d: show diff from a given date in the date command format (man date for details)

filename: the file to show diff from

disable forbid rudder-agent to be run by cron or service. This is useful when you want to temporarily prevent your *Rudder* agent from doing any modification to your system.

Options:

-s: stop rudder-agent in addition to disabling it

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

enable re-enable a disabled rudder-agent.

Options:

-s: start rudder-agent in addition to enabling it

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

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

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

Options:

-f: force the reinitialization without asking for confirmation

-i: run the agent in information mode, prints basic information

-v: run the agent in verbose mode, prints detailed information

-d: run the agent in debug mode, prints low-level information

-q: run the agent in quiet mode (display only error messages)

-w: show full strings, never cut output

-c: run the agent without color output

-r: run the agent with raw output

-R: run the agent in completely unparsed mode, with no return code of 1 in case of error. A little faster.

-m: run the agent with multiline output

health monitor agent health. Check that rudder agent has no problem

Options:

-n: run in nrpe mode, print a single line and return 0,1 or 2 put this line in your nrpe.cfg to use it command[check_rudder]=/opt/rudc agent health -n

history read log of old agent runs. This command will output historic logs of agent runs.

Options:

-c: show history without color output

-n: show maximum n lines of history

info display a summary of agent information. Outputs detailed information about the agent configuration, especially what defines the node (hostname, uuid and key hash) and its policy server.

Options:

-v: run the agent in verbose mode, prints detailed information

inventory force the agent to create and send a new inventory. This will trigger a new inventory creation and send it to the policy server. Even if the agent will do it regularly, it can be used to force the update after a modification on the node. This won't affect the node state, but only update server-side information.

Options:

-i: run the agent in information mode, prints basic information

-v: run the agent in verbose mode, prints detailed information

-d: run the agent in debug mode, prints low-level information

-q: run the agent in quiet mode (display only error messages)

-w: show full strings, never cut output

-c: run the agent without color output

-r: run the agent with raw output

-R: run the agent in completely unparsed mode, with no return code of 1 in case of error. A little faster.

-m: run the agent with multiline output

-f: run the agent even if it is disabled

log read log of old agent runs. This command will output historic logs of agent runs.

Options:

-w: show full strings, never cut output

-c: show log without color output

-r: show log with raw output

-R: show log in completely unparsed mode, with no return code of 1 in case of error. A little faster.

-m: show log with multiline output

-l: show log from the given file

-n: show log from the nth run before the last one

-d: show log from a given date in the date command format (man date for details)

reinit alias of command "rudder agent factory-reset". This command is a wrapper for "rudder agent factory-reset", that has replaced it.

reset reset agent status and cache. Remove all locks and state cache of the agent, and restore initial promises. This won't affect the desired state of the node, but will only reset the internal state of the agent. It is useful to test a rule without caching interference or when you have trouble with the promises updates, and is in most cases sufficient to resolve issues.

To completely reinitialize the agent and make it appear as a new node again, please use "rudder agent factory-reset" instead.

Options:

-i: run the agent in information mode, prints basic information

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

run force run agent promises. This command will force the agent to enforce current policies. You can run **rudder agent update** before to update the promises.

Options:

-u: update policy before running the agent (default is to run existing policy)

-i: run the agent in information mode, prints basic information

-v: run the agent in verbose mode, prints detailed information

-d: run the agent in debug mode, prints low-level information

-q: run the agent in quiet mode (display only error messages)

-w: show full strings, never cut output

-c: run the agent without color output

-r: run the agent with raw output

-R: run the agent in completely unparsed mode, with no return code of 1 in case of error. A little faster.

-m: run the agent with multiline output

-b: run the agent on a specific bundle

-D: define a class for this run

-f: run the agent even if it is disabled

start start the agent. Start the agent service using the appropriate service manager.

Options:

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

status show the agent status. **Options**:

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

stop stop the agent. Stop the agent service using the appropriate service manager.

Options:

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

update update promises on agent. The agent will fetch the last version of its promises from its configured policy server.

Options:

-i: run the agent in information mode, prints basic information

-v: run the agent in verbose mode, prints detailed information

-d: run the agent in debug mode, prints low-level information

-q: run the agent in quiet mode (display only error messages)

-c: run the agent without color output

-f: force full update

version get the agent version. Displays the version of the *Rudder* agent and of the underlying *CFEngine* agent.

16.10.1.7 remote

commands for rudder remote, run with **rudder remote** *command*

run trigger the execution of a remote agent. This command allows to override the agent run schedule and to immediately update the promises and enforce them on th specified node. This command is currently only allowed from the policy server of the target node.

Arguments:

node: IP or hostname of the target node or *all* for all nodes of the server

Options:

-i: run the agent in information mode, prints basic information

-v: run the agent in verbose mode, prints detailed information

-d: run the agent in debug mode, prints low-level information

-q: run the agent in quiet mode (display only error messages)

-w: show full strings, never cut output

-c: run the agent without color output

-r: run the agent with raw output

-R: run the agent in completely unparsed mode, with no return code of 1 in case of error. A little faster.

-m: run the agent with multiline output

-D: define a class for this run

-a: run the agent on all known nodes

-g: run the agent on all nodes of the group UUID given in parameter

-j: run this number of jobs in parallel

-t: provide an alternate token for group query (default from ~/.rudder)

-u: provide an alternate url for group query (default from ~/.rudder)

-C: provide an alternate config section in ~/.rudder for group query (default to first found)

16.10.1.8 server

commands for rudder server, run with **rudder server** *command*

debug run a debug `cf-serverd` intended for a specific node. This command targets a specific node and does not affect the running infrastructure. It uses *iptables* to redirect the specific node communications to the port the debug server is listening on (5310 by default).

Use Ctrl+C to stop the debug server.

Arguments:

-e: debug the cfengine enterprise server

-i: run a debug server for the given node

node: IP or hostname of the host you want to debug

disable-policy-distribution Stop *Rudder* from distributing new policies as a server. This is useful when you want to temporarily prevent your *Rudder* server from doing any changes on your agents

enable-policy-distribution Re-enable *Rudder* to distribute new policies as a server. This is useful after you have run "rudder server disable-policy-distribution" to allow the agent to restart the policy server. This will restart the policy server immediately.

reload-groups reload dynamic groups. By default, dynamic groups are evaluated every 5 minutes. This command triggers a reload of all dynamic groups.

Options:

-i: run the agent in information mode, displays all executed commands

-c: run the agent without color output

reload-techniques reload techniques. This command will reload the technique library into memory from the filesystem and regenerate the promises if necessary.

Options:

-i: run the agent in information mode, displays all executed commands

-c: run the agent without color output

upgrade-techniques upgrade techniques in the configuration repository from the packaged ones. This command will replace the techniques in /var/rudder/configuration-repository/techniques by the techniques found in /opt/rudder/share/techniques which is installed by rudder-technique package. The upgrade can take care of user defined changes.

Options:

-u: merge updated techniques into the configuration repository

-i: create the initial version of the update branch

-o: override existing technique without looking for local changes

-f: suppress any warning and run without prompting for input

-c: use the givent commit id as the update branch origin

16.10.1.9 AUTHOR

Normation SAS (contact@normation.com)

16.10.1.10 RESOURCES

Main web site: https://rudder-project.org/

Sources of the rudder command-line: https://github.com/*Normation*/rudder-agent/

16.10.1.11 COPYING

Copyright (C) 2014-2015 *Normation* SAS.

16.11 Technique reference

A technique is described by a XML file that lists:

- the template files
- the sections of the technique
- the variables that must be defined
- the compatibility list

16.11.1 Files organisation

The techniques are ordered in Categories. A Category is described by a category.xml file, that defines the name and description of a category. A Category can contain other Categories, or *Techniques*. A *Technique* is versioned, and can exist in several versions. The description of a *Technique* is the metadata.xml file.

```
techniques
+--- category.xml
+--- fileConfiguration
|   +--- category.xml
|   +--- security
|   |   +--- filesPermissions
|   |   |   +--- 1.0
|   |   |   |   +--- permlist.st
|   |   |   |   +--- metadata.xml
|   |   |   |   +--- filesPermissions.st
|   |   +--- category.xml
|   |   +--- sudoCheck
|   |   |   +--- 2.0
|   |   |   |   +--- metadata.xml
|   |   |   |   +--- sudoCheck.st
|   |   |   +--- 1.0
|   |   |   |   +--- metadata.xml
|   |   |   |   +--- sudoCheck.st
```

16.11.1.1 metadata.xml and CFEngine templates (*.st)

These files must reside in a folder with a version number. For each *Technique*, there can be several versions, *Rudder* will let you choose the version when creating a *Directive*.

16.11.1.2 Version number formating

The version number follows a formating "a la *Debian*" as described here: https://www.debian.org/doc/debian-policy/index.html#s-f-version, (without the debian_revision version)

16.11.2 General Rules

All the tag name in the .xml are in upper case, all the attributes are in camel case:

```
<SECTION name="example" component="true" componentKey="variable_name">
```

16.11.3 Details of the metadata.xml file

```
<TECHNIQUE id="technique_unique_id" name="human_name_of_the_technique">
  <DESCRIPTION>Description of the Technique</DESCRIPTION>
  <LONG_DESCRIPTION>Long description of the technique</LONG_DESCRIPTION>
  <DEPRECATED>Deprecation message</DEPRECATED>                <!-- Mark the Technique as  ↩
      deprecated, deprecation message is mandatory, Only available since Rudder 3.0 -->
  <DISPLAY>true/false</DISPLAY>                               <!-- Define if the Technique  ↩
      is displayed in the interface or not. Default value : true -->
  <COMPATIBLE>                                                <!-- Optional, describe the  ↩
      version of the OS and CFEngine Agent the Technique has been tested on. Only for  ↩
      information purpose -->
    <OS version=">=2.5">OS Name</OS>                          <!-- Optional; OS Name and  ↩
        version on which the Technique has been tested -->
    <AGENT version=">=3.6">cfengine-community</AGENT>         <!-- Optional; Agent name and ↩
         version on which the Technique has been tested -->
  </COMPATIBLE>
  <MULTIINSTANCE>true/false</MULTIINSTANCE>                   <!-- Optional; defines if  ↩
      several instances of this template with differents variables can be deployed on a node ↩
      ; default value : false -->
  <SYSTEM>true/false</SYSTEM>                                 <!-- Optional, defines if  ↩
      this Technique is a system Technique (internal Rudder usage); default value : false  ↩
      -->
  <BUNDLES>                                                   <!-- List of the bundles that ↩
       must be included in the bundlesequence -->
    <NAME>BundleName</NAME>
  </BUNDLES>
  <TMLS>                                                      <!-- List of all the  ↩
      templates defined by this Technique -->
   <TML name="tmlName">                                       <!-- Container for a TML ( ↩
       without the trailing .st -->
    <OUTPATH>relativ/path/of/file</OUTPATH>                   <!-- Optional; defines the  ↩
        relative path for the generated file for this template; default : techniqueName/ ↩
        version/tmlName.cf -->
    <INCLUDED>true/false</INCLUDED>                           <!-- Optional; defines if the ↩
         template must be in the inputs list of the generated promises; default : true -->
   </TML>
  </TMLS>
  <FILES>                                                     <!-- List of files to be  ↩
      copied "as-is" with this Technique. StringTemplate parser is NOT used on these. -->
    <FILE name="file.txt">                                    <!-- Container for a FILE.  ↩
        name (mandatory) = path to the file to copy, can be relative or absolute from  ↩
        RUDDER_CONFIGURATION_REPOSITORY/ (see below) -->
    <FILE name="file2.txt"><OUTPATH>technique_name/newname.txt</OUTPATH></FILE>
    <FILE name="RUDDER_CONFIGURATION_REPOSITORY/directory/other/file.txt"><OUTPATH> ↩
        technique_name/filename</OUTPATH></FILE>
  </FILES>
  <TRACKINGVARIABLE>                                          <!-- Defines a special system ↩
       variable TRACKINGKEY that contains all the necessary information to track which  ↩
      Directive generated the promises -->
    <SAMESIZEAS>VariableName</SAMESIZEAS>                     <!-- Optional; defines the  ↩
        cardinality of this variable based on the cardinality of the VariableName -->
  </TRACKINGVARIABLE>
  <SECTIONS>                                                  <!-- Lists all the sections  ↩
      of the promises -->
    <SECTION name="sectionName">                              <!-- Container of a section ( ↩
        see below) -->
    </SECTION>
  </SECTIONS>

</TECHNIQUE>
```

16.11.3.1 The <SECTION> tag

In a metadata.xml, there can be only one SECTIONS tag, that encloses one or several SECTION tags. A SECTION tag contains variables declaration and subsections. A SECTION can contains Variables definitions and SECTION.

```
<SECTION name="sectionName" multivalued="true/false" component="true/false" componentKey=" ↩
    variableName/None">
```

A SECTION has the following attributes:

- name : mandatory, the name of the section

- multivalued : optional, default false, defines if the section is repetable or not. If so, the Web Interface will display a "Add another" and "Delete" button for this section
- component : optional, default false; defines if the section is a component, and if true, the section will appear in the reporting, with its section name
- componentKey: optional, default None; defines the variable that is the key of the component. Note that the componentKey can only be defined if *component* is *true*
- displayPriority: optional, default high; defines if the section is displayed by default (high) or hidden by default (low)

Note
A multivalued section can only contain variable, and cannot contain section

Note
If there are no SECTION defined with *component="true"*, a default SECTION for reporting will be generated, named after the id of the *Technique* (the folder name of the *Technique*)

16.11.3.2 Variables definitions in the <SECTION> tags

There are three tags to create a variable:

- SELECT1: Can select only one value out of several. If there are less than 3 possible values, displays radio buttons, otherwise a select field.
- SELECT: Can select several values out of al the possibles. Displays checkboxes.
- INPUT: Displays an input field (that can be tuned)

```
<SELECT1/SELECT/INPUT>                                                   <!-- Depend  ↩
   on the display and behaviour needed -->
  <NAME>variableName</NAME>
  <DESCRIPTION>variableDescription</DESCRIPTION>
  <LONGDESCRIPTION>longDescription</LONGDESCRIPTION>                     <!-- Optional ↩
     , set the text in the tooltips -->
  <UNIQUEVARIABLE>true/false</UNIQUEVARIABLE>                            <!-- Optional ↩
     , default false; if true, this variable will have the same value over all the instance ↩
      of this template for a given node -->
  <ITEM>                                                                 <!-- Only for ↩
      SELECT and SELECT1, list of selectable values -->
    <VALUE>value</VALUE>                                                 <!-- value  ↩
       that will be put in the template-->
    <LABEL>humanReadableText</LABEL>                                     <!-- value  ↩
       displayed in the web interface -->
  </ITEM>
  <CONSTRAINT>                                                           <!-- Optional ↩
     , defines some constraints on values -->
    <DEFAULT>defaultValue</DEFAULT>                                      <!-- Optional ↩
       ; Defines a default value -->
    <TYPE>variableType</TYPE>                                            <!-- Optional ↩
       ; default string; variable type -->
    <MAYBEEMPTY>true/false</MAYBEEMPTY>                                  <!-- Optional ↩
       ; default false; defines if the variable is optional or not; only for the INPUT  ↩
       variable -->
    <REGEX error="errorMsg">regex</REGEX>                                <!-- Optional ↩
       ; only for the INPUT variable; efine a regular expression the variable should match, ↩
        and an optional error message -->
    <PASSWORDHASH>hashtype</PASSWORDHASH>                                    <!-- Optional ↩
        ; only for the password TYPE variable; define the way a password will be handled ( ↩
        hashed or not, hash types allowed ...) -->
  </CONSTRAINT>
</SELECT1/SELECT/INPUT>
```

Note: It is possible to inline LABEL and VALUE in the ITEM tag

```
<ITEM label="Red" value="red"/>
```

is equivalent to

```
<ITEM>
 <LABEL>Red</LABEL>
 <VALUE>red</VALUE>
</ITEM>
```

Note
INPUT fields are automatically escaped, meaning any quote will be written in the policies as \" ; and any backslash will be written as \\

16.11.3.3 Available types for an INPUT variable

- **string** : any string is accepted (no specific displayer)
- **textarea** : accept any strings, but use a textarea in place of the input text.
- **perm** : display a matrix of read/write/execute by user/group/all
- **integer** : only accept integers
- **datetime** : display a JQuery calendar and check date format
- **boolean** : display a checkbox
- **mail** : only accept emails
- **ip** : only accept ips. Before *Rudder* 3.1.14, 3.2.7 and 4.0.0, "ip" was accepting only IPv4 ip. Since these releases, it accepts both IPv4 and IPv6 format. <br />
- **ipv4** [since *Rudder* 3.1.14, 3.2.7, 4.0.0]: only accept IPv4 formatedt IPs
- **ipv6** [since *Rudder* 3.1.14, 3.2.7, 4.0.0]: only accept IPv6 formatted IPs
- **size-<unit>** : (size-b, size-kb, size-mb, size-gb ou size-tb)
- **raw** : the content of this field will not be escaped when written in the promises (*Rudder* >= 2.6)
- **password** : the content of this field will be handled as a password, and thus be hidden and transformed if necessary (see "Password handling" below) (*Rudder* >= 2.6)

16.11.3.4 The <FILES> tag

Example:

```
<FILES>
<FILE name="file.txt"><OUTPATH>foo/bar/other-name.txt</OUTPATH></FILE>
<FILE name="RUDDER_CONFIGURATION_REPOSITORY/some/absolute/file.txt"><OUTPATH>foo/bar/some- ↩
    name.txt</OUTPATH></FILE>
</FILES>
```

- **name** is mandatory. It's the path to file to copy, either relative to the technique directory (i.e, at the same level as metadata.xml) or absolute from the configuration repository directory if it starts with RUDDER_CONFIGURATION_REPOSITORY (usually /var/rudder/configuration-repository) (and yes, this forbids the use case where you want to have a sub-directory named RUDDER_CONFIGURATION_REPOSITORY under the technique directory - I'm sure one will find other way to do it if really needed :). The file will be taken from git, at the same git revision as other tehniques files.
- **OUTPATH** is optional. If not specified, the file will be copied into the target node promises at the same place as other files for the technique, with the same name. If specified, you have to give a path+name, where path is relative to the directory for agent promises on the node (i.e, if you want to put the file in the technique directory, you need to use "techniqueName/new-file-name.txt")

16.11.4 Examples

16.11.4.1 Multivalued sections

In the "NFS Client settings" *Technique*, there is a multivalued section with several entries. Here is a partial extract from it, with

- A multivalued section, named NFS mountpoint, that is multivalued and is a component. The variable reference for this component (the key) is NFS_CLIENT_LOCAL_PATH
- One SELECT1 field, that will show two radio buttons, Mount and Unmount, with the default value to Mount
- One INPUT field, named NFS_CLIENT_LOCAL_PATH, that is a text

```
<SECTION name="NFS mountpoint" multivalued="true" component="true" componentKey=" ↩
    NFS_CLIENT_LOCAL_PATH">
    <SELECT1>
      <NAME>NFS_CLIENT_UMOUNT</NAME>
      <DESCRIPTION>Which operation should be done on this mountpoint</DESCRIPTION>
      <ITEM>
        <LABEL>Mount</LABEL>
        <VALUE>no</VALUE>
      </ITEM>
      <ITEM>
        <LABEL>Unmount</LABEL>
        <VALUE>yes</VALUE>
      </ITEM>
      <CONSTRAINT>
        <DEFAULT>no</DEFAULT>
      </CONSTRAINT>
    </SELECT1>
    <INPUT>
      <NAME>NFS_CLIENT_LOCAL_PATH</NAME>
      <DESCRIPTION>Local path to mount the remote on</DESCRIPTION>
    </INPUT>
 ...
</SECTION>
```

16.11.4.2 Unique variable across several instance

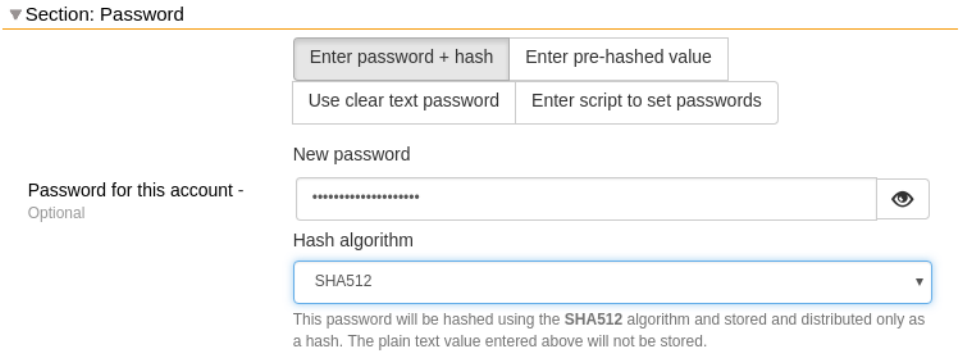
This variable can have only one value, over all the instances of this *Technique*, on a node

```
 <SECTIONS>
     <INPUT>
       <NAME>UNIQUE</NAME>
       <DESCRIPTION>Unique variable</DESCRIPTION>
       <CONSTRAINT>
         <TYPE>string</TYPE>
```

```
        <CONSTRAINT>
        <UNIQUEVARIABLE>true</UNIQUEVARIABLE>
    </INPUT>
  </SECTIONS>
```

16.11.4.3 Password handling

The password type allows to show an input text field whose content will be hashed when the form is submitted so that the password is never store in clear text.



Password for this account - Optional

New password

••••••••••••••••••••

Hash algorithm

SHA512

This password will be hashed using the **SHA512** algorithm and stored and distributed only as a hash. The plain text value entered above will not be stored.

Available hash formats

For now, the password field support these hash algorithms :

- **PLAIN** : that is not an hash algorithm, it just save the password in plain text, as inputed by the user.
- **MD5, SHA1, SHA256, SHA512** : uses the matching hash algorithm
- **LINUX-SHADOW-MD5, LINUX-SHADOW-SHA256, LINUX-SHADOW-SHA512** : build a string compatible with the Linux /etc/shadow format, as "specified" in http://man7.org/linux/man-pages/man3/crypt.3.html

***Technique* metatdata content**

To configure a password, you must specify two things in the `<CONSTRAINT>` section of the field:

- `<TYPE>password</TYPE>` : use the password type
- `<PASSWORDHASH>comma,separated,list,of,hash</PASSWORDHASH>` : specify the list of hash algo from witch the user will be allowed to choose.
- Available algorithm names are the ones from the section above (case insensitive).
- Choices are presented in order given by the list, the first being the default one.
- If the list contains only one algo, the drop down select if change to a phrase saying to the user that the given algo will be used.
- The list can not be empty. Moreover, if the `<MAYBEEMPTY>` contraint is set to false, the "None" option is not displayed to the user.

Password field definition example

```
<SECTION name="Password" component="true" componentKey="USERGROUP_USER_LOGIN">
    <INPUT>
        <NAME>USERGROUP_USER_PASSWORD</NAME>
        <DESCRIPTION>Password for this account</DESCRIPTION>
        <CONSTRAINT>
            <MAYBEEMPTY>true</MAYBEEMPTY>
            <TYPE>password</TYPE>
            <PASSWORDHASH>linux-shadow-md5,linux-shadow-sha256,linux-shadow-sha512</ ↩
                PASSWORDHASH>
        </CONSTRAINT>
    </INPUT>
</SECTION>
```

16.11.5 Known limitations

There are several known limitations at the moment, that are acknowleged, and will be solved in a "not too distant" future:

16.11.5.1 Can’t put a multivalued section in a multivalued section

It is not possible, due to limitation in the format in which the variable’s values are stored in the *LDAP* tree, to put multivalued sections within multivalued sections.

16.11.5.2 Can’t have several multivalued sections that are components with keys

For the moment, there is only one TRACKINGKEY, so it is not possible to have several multivalued sections that have keys.

16.11.5.3 Can’t have several sections that are components with keys in multivalued Techniques.

It is a side effect of the previous limitation.

16.12 Reports reference

This page describes the concept behind the reporting in *Rudder*, and specifically how to write the *Techniques* to get proper reporting in *Rudder*

16.12.1 Concepts

Each *Technique*, when converted into a *Directive* and applied to a *Node*, must generate reports for *Rudder* to get proper compliance reports. This reports must contains specific information :

- The Report type, that can be logs for information purpose or result to express a compliance
- The *Rule* Id (autogenerated)
- The *Directive* Id (autogenerated)
- The Version Id (revision of the *Rule*) (autogenerated)
- The name of the component the report is related to
- The value of the key variable in the component (or None if not available)
- The Execution Timestamp, to know in which execution of the agent the promise has been generated

These reports are sent via Syslog to the *Rudder Root Server*, parsed and put in a database, that is queried to generate the reporting

16.12.2 Report format

A report has the following format :

```
@@Technique@@Type@@RuleId@@DirectiveId@@VersionId@@Component@@Key@@ExecutionTimeStamp## ↩
    NodeId@#HumanReadableMessage
```

- *Technique* : Human readable *Technique* name
- Type : type of report (see bellow)
- RuleId : The Id of the *Configuration* Rule, autogenerated
- *Directive*Id : The Id of the *Directive*, autogenerated
- VersionId : the revision of the ConfigurationRule, autogenerated
- Component : the name of the component this *Directive* is related to (if no component are defined in the metadata.xml, then the *Technique* name is used)
- Key : the value of the reference variable. If there is no reference variable, then the value None should be used
- ExecutionTimeStamp : the timestamp of the current *CFEngine* execution
- *Node*Id : the id of the node
- HumanReadableMessage : a message than a Human can understand

16.12.2.1 Valid report types

Variables used to generate the reports

Some facilities have been created to help putting the right values at the right place

- `&TRACKINGKEY&`: this is an auto generated variable, put in the technique file, that *Rudder* will replace when writing the promises by

```
<pre>RuleId@@DirectiveId@@VersionId
```

- `$(g.execRun)`: this is replaced at runtime by *CFEngine* 3 to the current execution time
- `$(g.uuid)`: this is replaced at runtime by *CFEngine* 3 to the *Node* Id

16.13 Syntax of the Techniques

16.13.1 Generalities

The *Techniques* use the StringTemplate engine. A *Technique* **must** have the .st extension to be extended by *Rudder* (have some variables replaced, some part removed or added given some parameters).

16.13.2 Variable remplacement

Note : *Rudder* use a StringTemplate grammar slighlty different from the default one. Rather than using "$" as a variable identifier, the *Techniques* use "&" to avoid collision with the *CFEngine* variables

Name	Type	Mode	Max number	Details
log_trace	log	any	infinity	Should be used for advanced debuging purpose only.
log_debug	log	any	infinity	Should be used for debug purpose only.
log_info	log	any	infinity	Use for standard logging purposes.
log_warn	log	any	infinity	Used for logging only for the moment. Should be used when something unexpected happens.
log_repaired	log	enforce	infinity	Used for logging purposes, to list all that is repaired by the promises.
result_na	result	enforce	one per component/key	Defines the status of the Component to Not Applicable (if there are no result_success, result_repaired, result_error). Should be used only when the component is not applicable because it does not match the target context.
result_success	result	enforce	one per component/key	Defines the status of the Component to Success (if there are no result_repaired or result_error). Should be used only when everything is already in the correct state in this component for this key.
result_repaired	result	enforce	one per component/key	Defines the status of the Component to Repaired (if there are no result_error). Should be used only when something was not in the correct state, but could be corrected.
result_error	result	enforce	infinity per component/key	Defines the status of the Component to Error. Should be used when something was not in the correct state, and could not be corrected.
audit_na	result	audit	one per component/key	Defines the status of an Component to Not Applicable (if there are no result_success, result_repaired, result_error). Should be used only when the component is not applicable because it does not match the target context.
audit_compliant nent was not applicable to the node.	result	audit	one per component/key	Defines the status of the Component to Compliant (if there are no audit_noncompliant or audit_error). Should be used only when everything is already in the correct state in this component for this key.
audit_noncompliant	result	audit	one per component/key	Defines the status of the Component to Non Compliant (if there are no audit_error). Should be used only when something was not in the correct state.
audit_error	result	audit	infinity per component/key	Defines the status of the Component to Error. Should be used when the audit could not be done or was interrupted.

Table 16.1: Report Types

16.13.2.1 Single-valued variable remplacement

```
&UUID&
```

- Will be remplaced by the value of the variable UUID

16.13.2.2 Remplacement of variable with one or more values

```
&DNS_RESOLVERS: { "&it&" };separator=", "&
```

- Will be remplaced by `"8.8.8.8", "8.8.4.4"`
- Here, `&it&` is an alias for the current item in the list (with no confusion, because there is only one variable)

```
&POLICYCHILDREN, CHILDRENID : {host, uuid |
"/var/rudder/share/&uuid&/"
maproot => { host2ip("&host&"), escape("&host&") },
admit => { host2ip("&host&"), escape("&host&") };

} &
```

- `host` is an alias for the current value of POLICYCHILDREN
- `uuid` is an alias for the current value of CHILDRENID
- Both item are iterated at the same time, so both list must have the same length

16.13.2.3 Remplacement of variable with one or more value, and writing an index all along

```
&FILE_AND_FOLDER_MANAGEMENT_PATH:{path |"file[&i&][path]" string => "&path&";
}&
```

- *i* is an iterator, starting at 1

The result would be:

```
"file[1][path]" string => "/var";
"file[2][path]" string => "/bin";
```

16.13.2.4 Conditionnal writing of a section

```
&if(INITIAL)&

something

&endif&
```

The variable must either be:

- A boolean: If its value is true, then the section will be displayed
- A variable with the parameter `MAYBEEMPTY="true"`: If the value is not set, then the section won't be displayed, otherwise it will be displayed

More information can be found here: https://theantlrguy.atlassian.net/wiki/display/ST/ST+condensed+--+Templates+and+expressions

16.14 Best Practices for Techniques

16.14.1 Naming convention

- The name of bundle and classes should be written with underscore (i.e: this_is_a_good_example) instead of CamelCase (i.e: ThisIsABadExample)
- All variable, class and bundle names should be prefixed by "rudder_"
- The bundle entry point for the *Technique* should be named rudder_<name_of_the_technique>
- The bundles which makes all the actions should be suffixed by a meaningful name ("rudder_<name_of_the_*Technique*>_installation", "rudder_<name_of_the_*Technique*>_configuration", "rudder_<name_of_the_*Technique*>_reporting", ..). This rule applies even if there is only one bundle
- The prefix of classes should all be "rudder_<name of the *Technique*>_"
- The classes defined as an outcome should be named:
- `rudder_<name of the Technique>_<action>_kept`
- `rudder_<name of the Technique>_<action>_repaired`
- `rudder_<name of the Technique>_<action>_failed`
- `rudder_<name of the Technique>_<action>_denied`
- `rudder_<name of the Technique>_<action>_timeout`
- `rudder_<name of the Technique>_<action>_error` (error include failed, denied and timeout)
- The name of the bodies written in the *Rudder* Library should be prefixed: `rudder_common_`

16.14.2 Raising classes

- `rudder_<name of the Technique>_<action>_error` should be raised simultaneously as `rudder_<name of the Technique>_<action>_failed`, `rudder_<name of the Technique>_<action>_denied` or `rudder_<name of the Technique>_<action>_timeout`.
- The body **rudder_common_classes** automatically abide by this rule

16.14.3 Writing convention

16.14.3.1 Technique naming guidelines

The following rules should be followed when naming a new *Technique*:

- Try to keep names as short as possible, to improve readability
- Read the existing technique list, and particularly techniques related to what you are writing. The new names should be consistent with existing ones.
- The name should be a nominal group, use "File content" and "Service state" but never "Manage file content" or "Set Service state". It describes the target of the action, not the action itself.
- The name should look like: General Concept (package, file, etc.) + Source (from file, etc.) + Implementation details (platform, software name, etc.)
- Package sources (Zypper)

- HTTP server (*Apache*)
- Variable from local file (string)
- The general idea is to go from the most general information to the most precise.
- Use "directory" and never "folder"
- Use "settings" and never "configuration"
- Use **sentence case**, only the first word is capitalised, like in a normal sentence ("Variable from local file" and not "Variable from Local File").

16.14.3.2 In the Technique

- We try to follow *CFEngine* conventions but with some exceptions like using brackets "{}" instead of parenthesis "()"
- When defining bundles or bodies, the opening bracket should be on a dedicated line. Exemple:

```
bundle common control
{
  bundlesequence => { "exemple" };
}
```

- Indentation should be made by spaces. A incrementation of indentation is equal to two spaces
- The promise type should be indented by two spaces (instead of being at the same indentation level than the bundle name)
- The class expression should be indented by four spaces (two spaces after the promise type)
- The promiser should be indented by six spaces (two spaces after the class expression or four spaces after the promise type if no class expression is defined)
- Attributes of promises should be indented by eight spaces (two spaces after the promiser) and it should be only one attribute by line.
- Attribute's arrows ⇒ should all be at the same level, one character after the largest attribute name

```
bundle agent example
{
  type:
      "promiser"
        attribute  => "value1";

    class::
      "promiser2"
        attribute2 => "value2";
}
```

- Attributes of promise type "vars" and "classes" should be on only one line except if there are more than one attribute.
- For promise type "vars" and "classes" on one line, attribute names and the arrows should be aligned
- A list should be written multilines if it needs more than 80 characters in one line
- Multilines list should have comma after each element, except the last one.
- Multilines list should begin with only a bracket "{"

```
    vars:
        "value" slist =>
          {
            "one",
            "two",
            "three"
          };
```

- The name of the variable in argument of the bundle should be named "params"
- The call of the variables should be made with by using brackets `${var_correctly_called}` instead of parenthesis `$(var_wrongly_called)`
- Alternance of brackets and parenthesis are tolerated when lots of variables are imbricated for more readability: `${var_lvl1[$(var_lvl2[${var_lvl3}])]}`
- A *Technique* should have its bundle wrote with parameters
- All the bundles should have as first argument "prefix" which contains the prefix to use for all the classes made from an outcome. This prefix should never be hardcoded in the bundle.
- Always write comments with # when a promise needs more than 30 seconds of thought.
- If classes should be created in order to iterate for make a workaround of the normal ordering (i.e: "iteration_1", "iteration_2", "iteration_3"), they should always be defined at the end of the promise type "classes".
- The order to the promise type must always be in the order of the normal ordering : https://docs.cfengine.com/docs/3.10/reference-language-concepts-normal-ordering.html
- StringTemplate variables should always be written in UPPERCASE
- StringTemplate variables should be written with underscore
- StringTemplate variables should always be prefixed by the *Technique* name in uppecase too. i.e: `CHECK_GENERIC_FILE_FILE_NAME`

16.14.3.3 In the metadata.xml

- Name of sections should always be written in literary English (no CamelCase or underscores).
- The value of variable "Don't change" should always be "dontchange" or "" if the easier.

16.14.4 Files convention

- File names in a *Technique* should not be prefixed by the name of the *Technique*
- When a *Technique* needs specific bodies, the bodies should be written in a bodies.st file
- The file containing the bundle which makes all the actions (and containing the bundle "run") should be named "main.cf"
- The file containing all the variables and calling the bundle "run" should be name config.st
- Initialization of a new *Technique* should always be made from the file "technique-metadata-sample.xml" which is present on the root of the "rudder-techniques" repository
- *Rudder* standard library should be located in "common" *Technique*

16.14.5 Maintenance

- These rules were introduced after the 2.5 release of *Rudder* and before the 2.6 release. Therefore, they were enforced as of rudder-techniques-2.6.*.
- Always follow the conventions above when *Techniques* are updated but only for the lines edited. This rule concerns the *Techniques* on all the branches of git.
- On any branches that have released versions on them, we only allow minimal modifications. No lines should be modified if not to fix a bug (respecting these best practices is not currently considered a bug).

16.14.6 Testing

- There is a test suite in scripts/check-techniques.sh that check metadata.xml and normal ordering in code
- The list of all maintained techniques (techniques and versions) is in maintained-techniques file, and should be updated when new techniques or versions are created.

16.15 Package format

Rudder has a specific package format for plugins.

You can manage *Rudder* packages with the rudder-pkg command. This is the documentation of how they are created.

16.15.1 File description

A *Rudder* package file ends with the `.rpkg` extension.

A *Rudder* package file is an archive file and can be managed with the *ar* command.

The archive contains:

- A metadata file in JSON format named medatata
- A tarball file in txz format name scripts.txz that contains package setup utility scripts
- One or more tarball files in txz format that contain the package files

The metadata file is a JSON file and is named *metadata*:

```
{
  # the only currently supported type in "plugin" (mandatory)
  "type": "plugin",
  # the package name must consist of ascii characters without whitespace (mandatory)
  "name": "myplugin",
  # the package version has the form "rudder_major-version_major.version_minor" for a  ↩
      plugin (mandatory)
  "version": "4.1-1.0",
  # these are is purely informative (optional)
  "build-date": "2017-02-22T13:58:23Z",
  "build-commit": "34aea1077f34e5abdaf88eb3455352aa4559ba8b",
  # the list of jar files to enable if this is a webapp plugin (optional)
  "jar-files": [ "test.jar" ],
  # the list of packages or other plugins that this package depends on (optional)
  # this is currently only informative
  "depends": {
    # dependency on a specific binary that must be in the PATH
    "binary": [ "zip" ]
    # dependencies on dpkg based systems
    "dpkg": [ "apache2" ],
    "rpm": [ ],
    # dependency specific to debian-8
    "debian-8": [ ],
    "sles-11": [ ],
    # rudder dependency, ie this is a Rudder format package
    "rudder": [ "new-plugin" ]
  },
  # the plugin content (mandatory)
  "content": {
    # this will put the content of the extracted files.txz into /opt/rudder/share
    "files.txz": "/opt/rudder/share",
    "var_rudder.txz": "/var/rudder"
  }
}
```

To see a package metadata file use:

```
ar p package.rpkg medatada
```

The scripts.txz is a tarball that can contain zero or more executable files named:

- preinst that will be run before installing the package files
- postinst that will be run after installing the package files
- prerm that will be run before removing the package files
- postrm that will be run after removing the package files

preinst and postinst take one parameter that can be *install* or *upgrade*. The value *upgrade* is used when a previous version of the package is already installed.

To create the scripts.txz file use:

```
tar cvfJ scripts.txz preinst postinst prerm postrm
```

To create a *Rudder* package file use the ar command:

```
ar r mypackage-4.1-3.0.rpkg medatada scripts.txz files.txz
```

Note that ar r inserts or replaces files so you can create your package with incremental inserts.

To extract files, *use ar x* instead.

16.16 Rudder relay API

The `rudder-server-relay` package provides an HTTP API. It is available on simple relays and root servers. It is an internal API, not exposed to users, and used to provide various *Rudder* features.

16.16.1 Remote Run

The remote run API is available at `https://relay/rudder/relay-api/remote-run`. It allows triggering a run on nodes (like with the `rudder remote run` command).

16.16.1.1 Description

The remote run API applies to all nodes that are below the target relay server, which means:

- All nodes directly connected to the server
- All the relays that have the target node as policy server, and all nodes that are below them

In particular, it does not act on the target relay itself (except for the root server which is its own policy server).

There are different methods, whether you want to trigger all nodes, or only a part of them.

16.16.1.2 Security

The remote run calls are not authenticated, but restricted to:

- Local calls on the relay
- The relay's policy server

They requires allowing the policy server to connect to its nodes on port 5309.

16.16.1.3 Usage

This API provides the following methods:

- **POST** `/rudder/relay-api/remote-run/all`: Trigger a run on all nodes below the target relay.
- **POST** `/rudder/relay-api/remote-run/nodes/`<*node-id*>: Trigger a run on the *node-id* node (which must be under the target relay).
- **POST** `/rudder/relay-api/remote-run/nodes` Trigger a run on the given nodes (which must be under the target relay), see the `nodes` parameter below.

The general parameters are:

- `keep_output` = **true** or **false**: Should the agent output be returned (default: **false**)
- `asynchronous` = **true** or **false**: Should the server return immediately after trigerring the agent or wait for remote runs to end (default: **false**)
- `classes` = **class** or **class1,class2,etc.** for multiple classes: Classes to pass to the agent (default: none)

And, only for the `/rudder/relay-api/remote-run/nodes` call:

- `nodes` = **node_uuid** or **node_uuid1,node_uuid2,etc.** for multiple nodes: *Nodes* to trigger (default: none)

16.16.2 Shared Files

16.16.2.1 Description

The goal of this API is to share a file from node to node. The source nodes uses an API call to send the file, and the destination node will get the file using the same protocol as files shared from the policy server.

The relay that receives an API call to share a file (**PUT**) will:

- share the file directly if the target nodes is one of its managed nodes.

- send the file to a sub-relay if the target is somewhere under it
- forward the file to its policy server if the target is nowhere under it

The relay that receive a **HEAD** call will:

- If the file exists, compare the provided hash with the hash of the stored file, and return the result (**true** or **false**).
- If the file does not exist, return **false**.

The ttl is stored along with the file, and a clean task will regularly run and check for outdated files to remove.

There are ncf generic method that allow easy access to those methods from the nodes.

16.16.2.2 Security

This call is open to all nodes in the allowed networks of the target relay. The sent files are signed with the node's key, and the signature is checked before being traited.

16.16.2.3 Usage

This API provides the following methods:

- **PUT** `/shared-files/` *<target-uuid>* `/` *<source-uuid>* `/` *<file-id>*
- **HEAD** `/shared-files/` *<target-uuid>* `/` *<source-uuid>* `/` *<file-id>* `?hash=`*file-hash*

The common URL parameters are:

- `target-uuid` = **destination_node_uuid**: where to send the file to
- `source-uuid` = **source_node_uuid**: who sent the file
- `file-id` = **my_file_id**: under which name to store the file, this needs to be unique

The URL parameters specific to the **HEAD** call are:

- `file-hash` = **value of the hash**: hash of the shared file

The following are only needed for the **PUT** call:

- `hash_value` = **value of the hash**: hash of the shared file
- `algorithm` = **sha1**, **sha256** or **sha512**: algorithm used to hash the file
- `digest` = **: signature of the file
- `pubkey` = **: public key
- `ttl` = **: can be a number of second or a string of the long form "1day 2hours 3minute 4seconds" or abbreviated in the form "5h 3s"
- `header` = **rudder-signature-v1**: signing format (for now, only one possible value)

Chapter 17

Appendix: Glossary

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

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

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

`cf-serverd` This *CFEngine Community* daemon is listening on the network on *Rudder Root* and Relay servers, serving policies and files to *Rudder Nodes*.

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

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

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

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

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

Port 443, TCP, for nodes WebDAV/HTTPS communication port, used to send inventory and fetch the id of the *Rudder Server*. Powershell DSC agent communication port, used to fetch policy and shared files from the policy server on *Windows*

Port 443, TCP, for users HTTPS communication port, used to access the *Rudder* web interface or API.

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

Port 5309, TCP *Agent* communication port, used to trigger an agent run on a node from its policy server.

Port 5309, TCP *Agent* communication port, used to fetch policy and shared files from the policy server.

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

Port 80, TCP, for nodes WebDAV/HTTP communication port, kept for compatibility with pre-3.1 nodes and AIX nodes.

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

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

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

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

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

SQL server Store the received reports from the nodes.

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

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

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

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

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

License

External contributions:

CSS styles from the OpenStack manuals under *Apache* License version 2.0.

Font Awesome by Dave Gandy - http://fontawesome.io

Lato fonts by Łukasz Dziedzic, under SIL Open Font License 1.1.