

# **Rudder 4.0 - User Manual**

Copyright © 2011-2016 Normation SAS

Rudder User Documentation by Normation is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. Permissions beyond the scope of this license may be available at [normation.com](http://normation.com).

**COLLABORATORS**

	<i>TITLE :</i> Rudder 4.0 - User Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
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
4.0	Sep 2016		N

# Contents

<b>1</b>	<b>Online version</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>2</b>
2.1	Concepts . . . . .	2
2.1.1	Rudder functions . . . . .	2
2.1.2	Asset management concepts . . . . .	2
2.1.2.1	New Nodes . . . . .	2
2.1.2.2	Search Nodes . . . . .	2
2.1.2.3	Groups of Nodes . . . . .	3
2.1.3	Configuration management concepts . . . . .	3
2.2	Rudder components . . . . .	4
<b>3</b>	<b>Installation</b>	<b>5</b>
3.1	Requirements . . . . .	5
3.1.1	Networking . . . . .	5
3.1.1.1	Mandatory flows . . . . .	5
3.1.1.2	Optional flows . . . . .	5
3.1.1.3	DNS - Name resolution . . . . .	6
3.1.2	Supported Operating Systems . . . . .	6
3.1.2.1	For Rudder Nodes . . . . .	6
3.1.2.2	For Rudder Root Server . . . . .	7
3.1.3	Hardware specifications and sizing for Rudder Root Server . . . . .	7
3.1.3.1	Memory . . . . .	7
3.1.3.2	Disk . . . . .	7
3.2	Install Rudder Server . . . . .	8
3.2.1	Install Rudder Root server on Debian or Ubuntu . . . . .	8
3.2.1.1	Add the Rudder packages repository . . . . .	8
3.2.1.2	Install your Rudder Root Server . . . . .	8
3.2.2	Initial configuration of your Rudder Root Server . . . . .	9
3.2.3	Validate the installation . . . . .	9
3.2.4	Install Rudder Root server on SLES . . . . .	9

3.2.4.1	Configure the package manager . . . . .	9
3.2.4.2	Add the Rudder packages repository . . . . .	10
3.2.4.3	Install your Rudder Root Server . . . . .	10
3.2.5	Initial configuration of your Rudder Root Server . . . . .	11
3.2.6	Validate the installation . . . . .	11
3.2.7	Install Rudder Root server on RHEL-like systems . . . . .	11
3.2.7.1	Add the Rudder packages repository . . . . .	11
3.2.7.2	Install your Rudder Root Server . . . . .	12
3.2.8	Initial configuration of your Rudder Root Server . . . . .	12
3.2.9	Validate the installation . . . . .	13
3.3	Install Rudder Agent . . . . .	13
3.3.1	Install Rudder Agent on Debian or Ubuntu . . . . .	13
3.3.2	Install Rudder Agent on RHEL-like systems . . . . .	14
3.3.3	Install Rudder Agent on SLES . . . . .	15
3.3.4	Configure and validate . . . . .	16
3.3.4.1	Configure Rudder Agent . . . . .	16
3.3.4.2	Validate new Node . . . . .	16
3.4	Install Rudder Relay (optional) . . . . .	16
3.4.1	On the relay . . . . .	17
3.4.2	On the root server . . . . .	17
3.4.3	Validation . . . . .	17
3.4.4	Adding nodes to a relay server . . . . .	18
<b>4</b>	<b>Upgrade</b> . . . . .	<b>19</b>
4.1	Upgrade from Rudder 3.1 or 3.2 . . . . .	19
4.2	Upgrade from Rudder 3.0 or older . . . . .	19
4.3	Caution cases . . . . .	19
4.3.1	Compatibility between Rudder agent 4.0 and older server versions . . . . .	19
4.3.1.1	3.1.x and 3.2.x servers . . . . .	19
4.3.2	Compatibility between Rudder server 4.0 and older agent versions . . . . .	20
4.3.2.1	3.1.x and 3.2.x agents . . . . .	20
4.3.2.2	3.0.x or older . . . . .	20
4.3.3	Protocol for reporting . . . . .	20
4.3.4	Known issues . . . . .	20
4.4	On Debian or Ubuntu . . . . .	20
4.5	On RHEL or CentOS . . . . .	21
4.5.1	Rudder server . . . . .	21
4.5.2	Rudder agent . . . . .	21
4.6	On SLES . . . . .	22
4.7	Technique upgrade . . . . .	22
4.8	Upgrade manually installed relays . . . . .	23

<b>5</b>	<b>Rudder Web Interface</b>	<b>24</b>
5.1	Authentication . . . . .	24
5.2	Presentation of Rudder Web Interface . . . . .	24
5.2.1	Rudder Home . . . . .	24
5.2.2	Node Management . . . . .	25
5.2.3	Configuration Management . . . . .	26
5.2.4	Administration . . . . .	27
5.3	Units supported as search parameters . . . . .	27
5.3.1	Bytes and multiples . . . . .	27
5.3.2	Convenience notation . . . . .	27
5.3.3	Supported units . . . . .	28
<b>6</b>	<b>Node Management</b>	<b>29</b>
6.1	Node Inventory . . . . .	29
6.2	Accept new Nodes . . . . .	29
6.3	Search Nodes . . . . .	30
6.3.1	Quick Search . . . . .	30
6.3.2	Advanced Search . . . . .	32
6.4	Group of Nodes . . . . .	33
<b>7</b>	<b>Configuration Management</b>	<b>34</b>
7.1	Techniques . . . . .	34
7.1.1	Concepts . . . . .	34
7.1.2	Manage the Techniques . . . . .	34
7.1.3	Available Techniques . . . . .	35
7.1.3.1	Application management . . . . .	35
7.1.3.2	Distributing files . . . . .	35
7.1.3.3	File state configuration . . . . .	35
7.1.3.4	System settings: Miscellaneous . . . . .	35
7.1.3.5	System settings: Networking . . . . .	35
7.1.3.6	System settings: Process . . . . .	36
7.1.3.7	System settings: Remote access . . . . .	36
7.1.3.8	System settings: User management . . . . .	36
7.2	Directives . . . . .	36
7.3	Rules . . . . .	37
7.4	Variables . . . . .	37
7.4.1	User defined parameters . . . . .	37
7.4.2	System variables . . . . .	37
7.5	Compliance . . . . .	38

---

7.6	Validation workflow in Rudder . . . . .	39
7.6.1	What is a Change request ? . . . . .	39
7.6.1.1	Change request status . . . . .	40
7.6.1.2	Change request management page . . . . .	41
7.6.1.3	Change request detail page . . . . .	41
7.6.2	How to create a Change request ? . . . . .	42
7.6.3	How to validate a Change request ? . . . . .	43
7.6.3.1	Roles . . . . .	43
7.6.3.2	Self Validations . . . . .	44
7.6.4	Change request and conflicts . . . . .	44
7.6.5	Notifications: . . . . .	45
7.6.5.1	Pending change requests . . . . .	45
7.6.5.2	Change already proposed on Rule/Directive/Group . . . . .	45
7.7	Technique editor . . . . .	46
7.7.1	Introduction . . . . .	46
7.7.1.1	First, what is a Technique ? . . . . .	46
7.7.1.2	What is a Generic method? . . . . .	46
7.7.2	Technique Editor . . . . .	46
7.7.2.1	Utility . . . . .	46
7.7.2.2	Interface . . . . .	46
7.7.3	Create your first Technique . . . . .	49
7.7.3.1	1. General information . . . . .	49
7.7.3.2	2. Add and configure generic methods . . . . .	50
7.7.3.3	3. Save and apply your technique . . . . .	50
7.8	Policy Mode (Audit/Enforce) . . . . .	50
7.8.1	How is the effective mode computed? . . . . .	51
<b>8</b>	<b>Configuration Policies</b>	<b>53</b>
8.1	How to . . . . .	53
8.1.1	Enforce a line is present in a file only once . . . . .	53
8.2	Security considerations . . . . .	54
8.2.1	Data confidentiality . . . . .	54
8.2.1.1	Private data . . . . .	54
8.2.1.2	Common data . . . . .	55
8.2.2	Node-Server communication security . . . . .	55
8.2.2.1	File copy . . . . .	55
8.2.2.2	Inventory . . . . .	56

---

<b>9</b>	<b>Administration</b>	<b>57</b>
9.1	Archives . . . . .	57
9.1.1	Archive usecases . . . . .	57
9.1.1.1	Changes testing . . . . .	57
9.1.1.2	Changes qualification . . . . .	57
9.1.2	Concepts . . . . .	57
9.1.3	Archiving . . . . .	58
9.1.4	Importing configuration . . . . .	58
9.1.5	Deploy a preconfigured instance . . . . .	59
9.2	Event Logs . . . . .	59
9.3	Policy Server . . . . .	59
9.3.1	Configure allowed networks . . . . .	59
9.3.2	Clear caches . . . . .	59
9.3.3	Reload dynamic groups . . . . .	59
9.4	Plugins . . . . .	60
9.4.1	Install a plugin . . . . .	60
9.5	Basic administration of Rudder services . . . . .	60
9.5.1	Restart the agent of the node . . . . .	60
9.5.2	Restart the root rudder service . . . . .	60
9.5.2.1	Restart everything . . . . .	60
9.5.2.2	Restart only one component . . . . .	60
9.6	Password upgrade . . . . .	61
9.7	User management . . . . .	61
9.7.1	Configuration of the users using a XML file . . . . .	62
9.7.1.1	Generality . . . . .	62
9.7.1.2	Passwords . . . . .	62
9.7.2	Configuring an LDAP authentication provider for Rudder . . . . .	63
9.7.2.1	LDAP is only for authentication . . . . .	63
9.7.2.2	Enable LDAP authentication . . . . .	63
9.7.3	Authorization management . . . . .	64
9.7.3.1	Pre-defined roles . . . . .	64
9.7.3.2	Custom roles . . . . .	64
9.7.4	Going further . . . . .	64
9.8	Monitoring . . . . .	65
9.8.1	Monitoring Rudder itself . . . . .	65
9.8.1.1	Monitoring a Node . . . . .	65
9.8.1.2	Monitoring a Server . . . . .	65
9.8.2	Monitoring your configuration management . . . . .	65
9.8.2.1	Monitor compliance . . . . .	66



9.8.2.2	Monitor events . . . . .	66
9.9	Use Rudder inventory in other tools . . . . .	67
9.9.1	Export to a spreadsheet . . . . .	67
9.9.2	Use the inventory in Rundeck . . . . .	67
9.9.3	Use the inventory in Ansible . . . . .	67
<b>10</b>	<b>Usecases</b>	<b>68</b>
10.1	Dynamic groups by operating system . . . . .	68
10.2	Library of preventive policies . . . . .	68
10.3	Standardizing configurations . . . . .	68
10.4	Using Rudder as an Audit tool . . . . .	68
10.5	Using Audit mode to validate a policy before applying it . . . . .	69
<b>11</b>	<b>Advanced usage</b>	<b>70</b>
11.1	Node management . . . . .	70
11.1.1	Reinitialize policies for a Node . . . . .	70
11.1.2	Completely reinitialize a Node . . . . .	70
11.1.3	Change the agent run schedule . . . . .	71
11.1.4	Installation of the Rudder Agent . . . . .	71
11.1.4.1	Static files . . . . .	71
11.1.4.2	Generated files . . . . .	71
11.1.4.3	Services . . . . .	71
11.1.4.4	Configuration . . . . .	72
11.1.5	Rudder Agent interactive . . . . .	72
11.1.6	Processing new inventories on the server . . . . .	72
11.1.6.1	Verify the inventory has been received by the Rudder Root Server . . . . .	72
11.1.6.2	Process incoming inventories . . . . .	73
11.1.6.3	Validate new Nodes . . . . .	73
11.1.6.4	Prepare policies for the Node . . . . .	73
11.1.7	Agent execution frequency on nodes . . . . .	74
11.1.7.1	Checking configuration (CFEngine) . . . . .	74
11.1.7.2	Inventory (FusionInventory) . . . . .	75
11.2	Password management . . . . .	75
11.2.1	Configuration of the postgres database password . . . . .	75
11.2.2	Configuration of the OpenLDAP manager password . . . . .	75
11.2.3	Configuration of the WebDAV access password . . . . .	76
11.3	Policy generation . . . . .	76
11.3.1	Update policies button . . . . .	76
11.4	Technique creation . . . . .	76

---

11.4.1	Recommended solution: Technique Editor . . . . .	77
11.4.1.1	Using the Technique Editor . . . . .	77
11.4.1.2	Logs . . . . .	77
11.4.2	Understanding how Technique Editor works . . . . .	77
11.4.2.1	Directory layout . . . . .	77
11.4.3	Sharing ncf code with nodes . . . . .	78
11.4.3.1	From ncf Technique Editor to Rudder Techniques and back . . . . .	78
11.4.3.2	Hooks . . . . .	79
11.4.4	Create Technique manually . . . . .	79
11.4.4.1	Prerequisite . . . . .	79
11.4.4.2	Define your objective . . . . .	79
11.4.4.3	Initialize your new Technique . . . . .	80
11.4.4.4	Define variables . . . . .	81
11.4.4.5	First test in the Rudder interface . . . . .	81
11.4.4.6	Implement the behavior . . . . .	81
11.4.4.7	Read in the variables from Rudder . . . . .	81
11.4.4.8	Add reporting . . . . .	81
11.5	Node properties . . . . .	81
11.5.1	Using properties . . . . .	81
11.5.2	Under the hood . . . . .	82
11.6	Node properties expansion in directives . . . . .	82
11.6.1	Feature availability . . . . .	82
11.6.2	Usage . . . . .	83
11.6.2.1	Providing a default value . . . . .	83
11.6.2.2	Forcing expansion on the node . . . . .	83
11.7	JavaScript evaluation in Directives . . . . .	84
11.7.1	Feature availability . . . . .	84
11.7.2	Usage . . . . .	84
11.7.3	Rudder utility library . . . . .	84
11.7.3.1	Standard hash methods . . . . .	84
11.7.3.2	UNIX password-compatible hash methods . . . . .	84
11.7.4	Status and future support . . . . .	85
11.8	New directives default naming scheme . . . . .	86
11.9	REST API . . . . .	86
11.9.1	Default setup . . . . .	86
11.9.1.1	Rudder Authentication . . . . .	86
11.9.1.2	Apache access rules . . . . .	87
11.9.1.3	User for REST actions . . . . .	87
11.9.2	Status . . . . .	87

---

11.9.3 Promises regeneration . . . . .	87
11.9.4 Dynamic groups regeneration . . . . .	87
11.9.5 Technique library reload . . . . .	87
11.9.6 Archives manipulation . . . . .	87
11.9.6.1 Archiving: . . . . .	87
11.9.6.2 Listing: . . . . .	88
11.9.6.3 Restoring a given archive: . . . . .	88
11.9.6.4 Restoring the latest available archive (from a previously archived action, and so from a Git tag):	88
11.9.6.5 Restoring the latest available commit (use Git HEAD): . . . . .	88
11.9.6.6 Downloading a ZIP archive . . . . .	88
11.10 Use a database on a separate server . . . . .	89
11.10.1 On the database server . . . . .	90
11.10.2 On the root server . . . . .	90
11.11 Multiserver Rudder . . . . .	91
11.11.1 Preliminary steps . . . . .	92
11.11.2 Install rudder-relay-top . . . . .	92
11.11.3 Install rudder-db . . . . .	92
11.11.4 Install rudder-ldap . . . . .	92
11.11.5 Install rudder-web . . . . .	93
11.12 Server migration . . . . .	93
11.12.1 What files you need . . . . .	93
11.12.2 Handle configuration files . . . . .	93
11.12.2.1 Copy /var/rudder/configuration-repository . . . . .	93
11.12.2.2 Use Archive feature of Rudder . . . . .	94
11.12.3 Handle CFEngine keys . . . . .	94
11.12.3.1 Keep your CFEngine keys . . . . .	94
11.12.3.2 Change CFEngine keys . . . . .	94
11.12.4 On your nodes . . . . .	94
11.13 Mirroring Rudder repositories . . . . .	94
<b>12 Handbook</b>	<b>95</b>
12.1 Database maintenance . . . . .	95
12.1.1 Automatic PostgreSQL table maintenance . . . . .	95
12.1.2 PostgreSQL database vacuum . . . . .	96
12.1.3 LDAP database reindexing . . . . .	96
12.2 Migration, backups and restores . . . . .	96
12.2.1 Backup . . . . .	96
12.2.2 Restore . . . . .	97
12.2.3 Migration . . . . .	98

---

12.3 Performance tuning . . . . .	98
12.3.1 Reports retention . . . . .	98
12.3.2 Apache web server . . . . .	98
12.3.3 Jetty . . . . .	98
12.3.4 Java "Out Of Memory Error" . . . . .	98
12.3.5 Configure RAM allocated to Jetty . . . . .	99
12.3.6 Optimize PostgreSQL server . . . . .	99
12.3.6.1 Suggested values on an high end server . . . . .	99
12.3.6.2 Suggested values on a low end server . . . . .	100
12.3.7 CFEngine . . . . .	100
12.3.8 Rsyslog . . . . .	101
12.3.8.1 Maximum number of file descriptors . . . . .	101
12.3.8.2 Network backlog . . . . .	102
12.3.8.3 Conntrack table . . . . .	102
<b>13 Troubleshooting and common issues</b>	<b>103</b>
13.1 Some reports are in "No report" . . . . .	103
13.1.1 If you get no reports at all for the Node . . . . .	103
13.1.2 If you get incomplete reporting for the Node . . . . .	103
13.2 Communication issues between agent and server . . . . .	104
13.2.1 DNS issues . . . . .	104
13.2.2 Inventory issues . . . . .	104
13.3 Technique editing . . . . .	104
13.4 Database is using too much space . . . . .	104
<b>14 Reference</b>	<b>106</b>
14.1 Inventory workflow, from nodes to Root server . . . . .	106
14.1.1 Processing inventories on node . . . . .	107
14.1.2 Processing inventories on relays . . . . .	107
14.1.3 Processing inventories on root server . . . . .	108
14.1.4 Queue of inventories waiting to be parsed . . . . .	108
14.2 Rudder Server data workflow . . . . .	109
14.3 Configuration files for Rudder Server . . . . .	111
14.4 Rudder Agent workflow . . . . .	111
14.4.1 Request data from Rudder Server . . . . .	113
14.4.2 Launch processes . . . . .	113
14.4.3 Identify Rudder Root Server . . . . .	113
14.4.4 Inventory . . . . .	113
14.4.5 Syslog . . . . .	113

---

14.4.6	Apply Directives	113
14.5	Configuration files for a Node	113
14.6	Packages organization	114
14.6.1	Packages	114
14.6.2	Software dependencies and third party components	115
14.7	Generic methods	116
14.7.1	Command	116
14.7.1.1	command_execution	116
14.7.1.2	command_execution_result	116
14.7.2	Condition	117
14.7.2.1	condition_from_command	117
14.7.2.2	condition_from_expression	117
14.7.2.3	condition_from_expression_persistent	118
14.7.3	Directory	118
14.7.3.1	directory_absent	118
14.7.3.2	directory_check_exists	118
14.7.3.3	directory_create	119
14.7.4	File	119
14.7.4.1	file_check_FIFO_pipe	119
14.7.4.2	file_check_block_device	119
14.7.4.3	file_check_character_device	120
14.7.4.4	file_check_exists	120
14.7.4.5	file_check_hardlink	120
14.7.4.6	file_check_regular	121
14.7.4.7	file_check_socket	121
14.7.4.8	file_check_symlink	121
14.7.4.9	file_check_symlinkto	122
14.7.4.10	file_copy_from_local_source	122
14.7.4.11	file_copy_from_local_source_recursion	122
14.7.4.12	file_copy_from_remote_source	123
14.7.4.13	file_copy_from_remote_source_recursion	123
14.7.4.14	file_create	124
14.7.4.15	file_create_symlink	124
14.7.4.16	file_create_symlink_enforce	124
14.7.4.17	file_create_symlink_force	124
14.7.4.18	file_download	125
14.7.4.19	file_enforce_content	125
14.7.4.20	file_ensure_block_in_section	125
14.7.4.21	file_ensure_block_present	126

14.7.4.22	file_ensure_key_value . . . . .	126
14.7.4.23	file_ensure_key_value_parameter_in_list . . . . .	126
14.7.4.24	Example . . . . .	127
14.7.4.25	file_ensure_key_value_parameter_not_in_list . . . . .	127
14.7.4.26	Example . . . . .	127
14.7.4.27	file_ensure_key_value_present_in_ini_section . . . . .	128
14.7.4.28	file_ensure_keys_values . . . . .	128
14.7.4.29	Usage . . . . .	128
14.7.4.30	Example . . . . .	129
14.7.4.31	file_ensure_line_present_in_ini_section . . . . .	129
14.7.4.32	file_ensure_line_present_in_xml_tag . . . . .	130
14.7.4.33	file_ensure_lines_absent . . . . .	130
14.7.4.34	file_ensure_lines_present . . . . .	130
14.7.4.35	file_from_string_mustache . . . . .	130
14.7.4.36	file_from_template . . . . .	131
14.7.4.37	file_from_template_jinja2 . . . . .	131
14.7.4.38	Setup . . . . .	131
14.7.4.39	Syntax . . . . .	131
14.7.4.40	file_from_template_mustache . . . . .	133
14.7.4.41	Syntax . . . . .	133
14.7.4.42	file_from_template_type . . . . .	136
14.7.4.43	Usage . . . . .	136
14.7.4.44	Template types . . . . .	136
14.7.4.45	Example . . . . .	136
14.7.4.46	file_remove . . . . .	137
14.7.4.47	file_replace_lines . . . . .	137
14.7.4.48	Syntax . . . . .	137
14.7.4.49	Example . . . . .	138
14.7.4.50	file_template_expand . . . . .	138
14.7.5	Group . . . . .	138
14.7.5.1	group_absent . . . . .	138
14.7.5.2	group_present . . . . .	139
14.7.6	Http . . . . .	139
14.7.6.1	http_request_check_status_headers . . . . .	139
14.7.6.2	http_request_content_headers . . . . .	139
14.7.7	Log . . . . .	140
14.7.7.1	log_rudder . . . . .	140
14.7.8	Logger . . . . .	140
14.7.8.1	logger_rudder . . . . .	140

14.7.9	Package . . . . .	140
14.7.9.1	package_absent . . . . .	140
14.7.9.2	package_check_installed . . . . .	141
14.7.9.3	package_install . . . . .	141
14.7.9.4	package_install_version . . . . .	141
14.7.9.5	package_install_version_cmp . . . . .	141
14.7.9.6	package_install_version_cmp_update . . . . .	142
14.7.9.7	package_present . . . . .	142
14.7.9.8	package_remove . . . . .	143
14.7.9.9	package_state . . . . .	143
14.7.9.10	Setup . . . . .	143
14.7.9.11	Package parameters . . . . .	144
14.7.9.12	Package providers . . . . .	144
14.7.9.13	Examples . . . . .	144
14.7.9.14	package_state_options . . . . .	145
14.7.9.15	package_verify . . . . .	145
14.7.9.16	package_verify_version . . . . .	146
14.7.10	Permissions . . . . .	146
14.7.10.1	permissions . . . . .	146
14.7.10.2	permissions_dirs . . . . .	146
14.7.10.3	permissions_dirs_recurse . . . . .	147
14.7.10.4	permissions_recurse . . . . .	147
14.7.10.5	permissions_type_recursion . . . . .	147
14.7.11	Schedule . . . . .	148
14.7.11.1	schedule_simple . . . . .	148
14.7.11.2	schedule_simple_catchup . . . . .	148
14.7.11.3	schedule_simple_nodups . . . . .	149
14.7.11.4	schedule_simple_stateless . . . . .	150
14.7.12	Service . . . . .	150
14.7.12.1	service_action . . . . .	150
14.7.12.2	service_check_disabled_at_boot . . . . .	151
14.7.12.3	service_check_running . . . . .	151
14.7.12.4	service_check_running_ps . . . . .	151
14.7.12.5	service_check_started_at_boot . . . . .	151
14.7.12.6	service_ensure_disabled_at_boot . . . . .	152
14.7.12.7	service_ensure_running . . . . .	152
14.7.12.8	service_ensure_running_path . . . . .	152
14.7.12.9	service_ensure_started_at_boot . . . . .	152
14.7.12.10	service_ensure_stopped . . . . .	153

14.7.12.1	service_reload	153
14.7.12.2	service_restart	153
14.7.12.3	service_restart_if	153
14.7.12.4	service_start	154
14.7.12.5	service_stop	154
14.7.13	User	154
14.7.13.1	user_absent	154
14.7.13.2	user_create	154
14.7.14	Variable	155
14.7.14.1	variable_dict	155
14.7.14.2	variable_dict_from_file	155
14.7.14.3	variable_iterator	156
14.7.14.4	variable_iterator_from_file	156
14.7.14.5	variable_string	157
14.7.14.6	variable_string_from_file	157
14.8	Man pages	157
14.8.1	rudder(8)	157
14.8.1.1	NAME	157
14.8.1.2	SYNOPSIS	158
14.8.1.3	DESCRIPTION	158
14.8.1.4	OPTIONS	158
14.8.1.5	COMMANDS	158
14.8.1.6	agent	158
14.8.1.7	remote	161
14.8.1.8	server	161
14.8.1.9	AUTHOR	162
14.8.1.10	RESOURCES	162
14.8.1.11	COPYING	162
14.9	Technique reference	162
14.9.1	Files organisation	162
14.9.1.1	metadata.xml and CFEngine templates (*.st)	163
14.9.1.2	Version number formatting	163
14.9.2	General Rules	163
14.9.3	Details of the metadata.xml file	163
14.9.3.1	The <SECTION> tag	164
14.9.3.2	Variables definitions in the <SECTION> tags	165
14.9.3.3	Available types for an INPUT variable	166
14.9.3.4	The <FILES> tag	166
14.9.4	Examples	167



14.9.4.1	Multivalued sections . . . . .	167
14.9.4.2	Unique variable across several instance . . . . .	167
14.9.4.3	Password handling . . . . .	167
14.9.5	Known limitations . . . . .	168
14.9.5.1	Can't put a multivalued section in a multivalued section . . . . .	169
14.9.5.2	Can't have several multivalued sections that are components with keys . . . . .	169
14.9.5.3	Can't have several sections that are components with keys in multivalued Techniques. . . . .	169
14.10	Reports reference . . . . .	169
14.10.1	Concepts . . . . .	169
14.10.2	Report format . . . . .	169
14.10.2.1	Valid report types . . . . .	170
14.11	Syntax of the Techniques . . . . .	170
14.11.1	Generalities . . . . .	170
14.11.2	Variable replacement . . . . .	170
14.11.2.1	Single-valued variable replacement . . . . .	170
14.11.2.2	Replacement of variable with one or more values . . . . .	172
14.11.2.3	Replacement of variable with one or more value, and writing an index all along . . . . .	172
14.11.2.4	Conditionnal writing of a section . . . . .	172
14.12	Best Practices for Techniques . . . . .	173
14.12.1	Naming convention . . . . .	173
14.12.2	Raising classes . . . . .	173
14.12.3	Writing convention . . . . .	173
14.12.3.1	In the Technique . . . . .	173
14.12.3.2	In the metadata.xml . . . . .	175
14.12.4	Files convention . . . . .	175
14.12.5	Maintenance . . . . .	175
14.12.6	Testing . . . . .	175

## 15 Appendix: Glossary

176

# List of Figures

2.1	Concepts diagram . . . . .	4
3.1	Rudder relay node . . . . .	17
5.1	Rudder Homepage . . . . .	25
5.2	Accept new nodes screen . . . . .	26
5.3	Configuration Policy (Rules) screen . . . . .	26
5.4	Settings screen . . . . .	27
7.1	Compliance on a Rule . . . . .	38
7.2	Compliance history on a Rule . . . . .	39
11.1	Generate policy workflow . . . . .	74
14.1	Rudder data workflow . . . . .	110
14.2	Rudder Agent workflow . . . . .	112
14.3	Rudder packages and their dependencies . . . . .	114

# List of Tables

- 5.1 Units supported by Rudder search engine . . . . . 28
- 6.1 is: keywords . . . . . 30
- 6.2 in: keywords (common) . . . . . 31
- 6.3 in: keywords (nodes) . . . . . 31
- 6.4 in: keywords (groups) . . . . . 31
- 6.5 in: keywords (directives) . . . . . 31
- 6.6 in: keywords (parameters) . . . . . 31
- 6.7 in: keywords (rules) . . . . . 31
- 9.1 Hashed passwords algorithms list . . . . . 62
- 14.1 Report Types . . . . . 171

# Chapter 1

## Online version

You can also read [the \*Rudder\* User Documentation on the Web](#).

## Chapter 2

# Introduction

This chapter presents the main concepts and the architecture of *Rudder*: what are the server types and their interactions. Reading this chapter will help you to learn the terms used, and to prepare the deployment of a *Rudder* installation.

## 2.1 Concepts

### 2.1.1 Rudder functions

*Rudder* addresses two main functions:

1. Configuration management;
2. Asset management;

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

### 2.1.2 Asset management concepts

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

#### 2.1.2.1 New Nodes

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

#### 2.1.2.2 Search Nodes

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

---

### 2.1.2.3 Groups of Nodes

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

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

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

## 2.1.3 Configuration management concepts

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

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

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

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

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

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

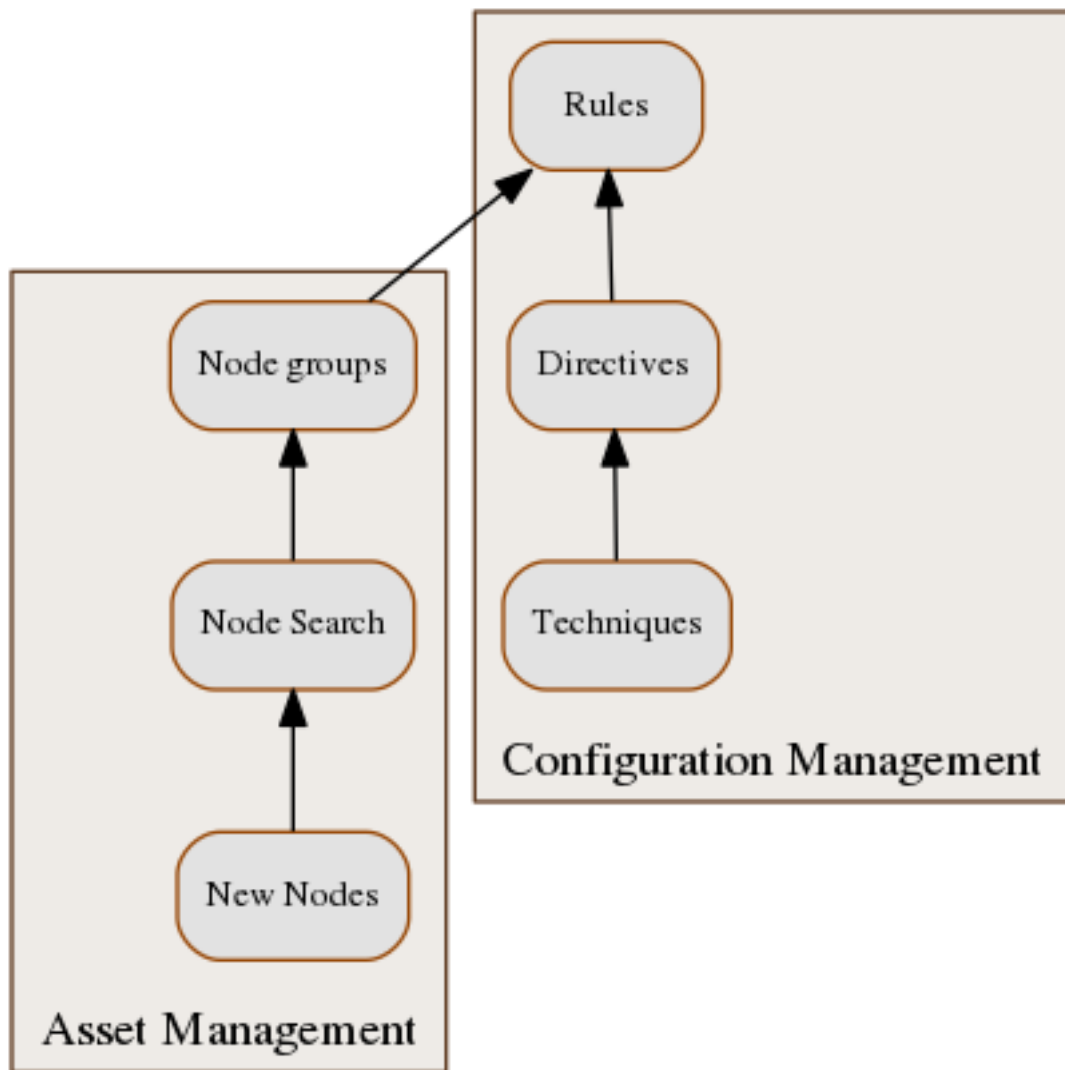


Figure 2.1: Concepts diagram

## 2.2 Rudder components

The *Rudder* infrastructure uses three types of machines:

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

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

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

## Chapter 3

# Installation

### 3.1 Requirements

#### 3.1.1 Networking

##### 3.1.1.1 Mandatory flows

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

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

**Port 443, TCP, for nodes** WebDAV/HTTPS communication port, used to send inventory and fetch the id of the *Rudder Server*.

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

And this one is optional:

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

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

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

##### 3.1.1.2 Optional flows

These flows are recommended for compatibility:

**Port 80, TCP, for nodes** WebDAV/HTTP communication port, kept for compatibility with pre-3.1 nodes and AIX nodes.

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

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

This version used to be called Nova before.

---



### 3.1.1.3 DNS - Name resolution

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

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

## 3.1.2 Supported Operating Systems

### 3.1.2.1 For Rudder Nodes

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

GNU/Linux:

- *Debian* 5 to 8
- RedHat *Enterprise Linux (RHEL)* / *RHEL*-like 3 and 5 to 7
- *Fedora* 18
- *SuSE Linux Enterprise Server (SLES)* 10 SP3, 11 SP1, 11 SP3, 12
- *Ubuntu* 10.04 LTS (Lucid), 12.04 LTS (Precise), 12.10 (Quantal), 14.04 LTS (Trusty), 16.04 LTS (Xenial)

Other Unix systems:

- IBM AIX 5.3, 6.1 and 7.1

Windows:

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

---

### Windows and AIX Nodes

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

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



### Unsupported Operating Systems

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

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

---

### 3.1.2.2 For Rudder Root Server

The following operating systems are supported as a Root server:

GNU/Linux:

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

### 3.1.3 Hardware specifications and sizing for Rudder Root Server

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

#### 3.1.3.1 Memory

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

- less than 50 nodes: 2GB
- between 50 and 1000 nodes: 4GB
- more than 1000 nodes: 4GB + 1GB of RAM by 500 nodes above 1000.
- more than 4000 nodes: 10GB + 2GB of RAM by 500 nodes above 4000.

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 PostgreSQL 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.

---

#### 3.1.3.2 Disk

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

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

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

Follow the [Reports Retention](#) section to configure the retention duration.

---

## 3.2 Install Rudder Server

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

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

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

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

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

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

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

### 3.2.1 Install Rudder Root server on Debian or Ubuntu

#### 3.2.1.1 Add the Rudder packages repository

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-4.0/ $(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.

#### 3.2.1.2 Install your Rudder Root Server

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

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

---

**Warning**

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

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



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

### 3.2.2 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.

---

### 3.2.3 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.

### 3.2.4 Install Rudder Root server on SLES

#### 3.2.4.1 Configure the package manager

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

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

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

---

**Warning**

SLES 11 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`)

### 3.2.4.2 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-4.0/SLES_11/ Rudder
zypper refresh
```

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

### 3.2.4.3 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
```

**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.

### 3.2.5 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.

---

### 3.2.6 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.

### 3.2.7 Install Rudder Root server on RHEL-like systems

#### 3.2.7.1 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_4.0]
name=Rudder 4.0 EL repository
baseurl=http://www.rudder-project.org/rpm-4.0/RHEL_$releasever/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-4.0/RHEL_$releasever/repodata/repomd.xml.key' > / ←
etc/yum.repos.d/rudder.repo
```

### 3.2.7.2 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
```

### 3.2.8 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.

---

### 3.2.9 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.

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

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

---

## 3.3 Install Rudder Agent

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

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

---

#### Components

This agent contains the following tools:

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

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

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

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

---

### 3.3.1 Install Rudder Agent on Debian or Ubuntu

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-4.0/ $(lsb_release -cs) main" > /etc/apt/sources.list.d/rudder.list
```

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

```
sudo apt-get update
```

Install the `rudder-agent` package:

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

### 3.3.2 Install Rudder Agent on RHEL-like systems

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_4.0]
name=Rudder 4.0 EL repository
baseurl=http://www.rudder-project.org/rpm-4.0/RHEL_$releasever/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-4.0/RHEL_$releasever/repodata/repomd.xml.key' > /etc/yum.repos.d/rudder.repo
```

---

#### Tip

The RPM can be directly downloaded for a standalone installation, from the following URL: [http://www.rudder-project.org/rpm-4.0/RHEL\\_7/](http://www.rudder-project.org/rpm-4.0/RHEL_7/) (or *RHEL\_6*, *RHEL\_5*, etc, depending on your host's OS version)

---

Install the package:

```
yum install rudder-agent
```

Or:

```
yum install rudder-agent-4.0.0-1.EL.7.x86_64.rpm
```

---

### 3.3.3 Install Rudder Agent on SLES

Following commands are executed as the `root` user.

#### Warning

You 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-4.0/SLES_12/ ↵
Rudder
```

- on SLES 11:

```
zypper ar -n 'Rudder SLES repository' http://www.rudder-project.org/rpm-4.0/SLES_11_SP1/ ↵
Rudder
```

- on SLES 10:

```
zypper sa 'http://www.rudder-project.org/rpm-4.0/SLES_10_SP3/' Rudder
```

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

```
zypper ref
```

Install the `rudder-agent` package:

```
zypper install rudder-agent
```

#### Tip

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

### 3.3.4 Configure and validate

#### 3.3.4.1 Configure Rudder Agent

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

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

---

**Tip**

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

---

You can now start the *Rudder* service with:

```
service rudder start
```

#### 3.3.4.2 Validate new Node

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

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

```
rudder agent inventory
```

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

```
rudder agent run
```

## 3.4 Install Rudder Relay (optional)

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

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

Using a relay, you are able to:

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

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

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

### 3.4.1 On the relay

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

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

### 3.4.2 On the root server

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

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

### 3.4.3 Validation

When every step has completed successfully:

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

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

The screenshot displays the 'Node Details' page for the node 'relays-relay.cobbler.com' (last updated 2013-10-15 06:25). The interface includes a top navigation bar with tabs: Node summary, Hardware, File systems, Network interfaces, Software, Environment variables, Processes, Virtual machines, and Reports. Below this is a 'Technical logs' section. The main content area is divided into two panels. The left panel, titled 'Groups containing this node', shows a tree structure with 'Root of the group and group categories' and 'System groups'. The right panel, titled 'Node characteristics', contains a 'Delete this node' button and a 'Node characteristics' section. This section is divided into 'General' (Hostname, Machine type, Total physical memory, Total swap space), 'Operating system details' (Operating System, Operating System Type, Operating System Name, Operating System Version, Operating System Service Pack), 'Rudder information' (Role, Inventory date, Date inventory last received, Date first accepted in Rudder, Agent name, Rudder ID, Rudder Policy Server), and 'Accounts' (Administrator account, Local account(s)).

Node Details - relays-relay.cobbler.com (last updated 2013-10-15 06:25)	
<b>Node summary</b>   Hardware   File systems   Network interfaces   Software   Environment variables   Processes   Virtual machines   Reports	<b>Technical logs</b>
<b>Groups containing this node</b> 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)	<b>Action</b> Delete this node
<b>Node characteristics</b>	
<b>General</b> Hostname: relays-relay.cobbler.com Machine type: Virtual machine (QEMU/KVM) Total physical memory (RAM): 490 MB Total swap space: 1.97 GB	
<b>Operating system details</b> 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	
<b>Rudder information</b> 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	
<b>Accounts</b> Administrator account: root Local account(s): abrt, adm, apache, bin, daemon, dbus, ftp, games, gopher, haldaemon, halt, lp, mail, nobody, ntp, operator, postfix, saslauth, shutdown, sshd, sync, tcpdump, uucp, vcsa	

Figure 3.1: Rudder relay node

### 3.4.4 Adding nodes to a relay server

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

You then have two possible cases:

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

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

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

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

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

```
rudder agent inventory
```

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

## Chapter 4

# Upgrade

This short chapter covers the upgrade of the *Rudder Server* Root and *Rudder Agent* from older versions to the latest version, 4.0. 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 from Rudder 3.1 or 3.2

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



---

**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 server.

However, some networking issues may appear when using 4.0 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 4.0 **before** upgrading the nodes so that the configuration distributed to the nodes include the use of the new protocol.

---

### 4.2 Upgrade from Rudder 3.0 or older

Direct upgrades from 3.0.x and older are no longer supported on 4.0. 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 4.0.

### 4.3 Caution cases

#### 4.3.1 Compatibility between Rudder agent 4.0 and older server versions

##### 4.3.1.1 3.1.x and 3.2.x servers

*Rudder* agents 4.0.x are not compatible with *Rudder* servers older than 4.0.0. You need to upgrade your server to 4.0 before the agents.

---

## 4.3.2 Compatibility between Rudder server 4.0 and older agent versions

### 4.3.2.1 3.1.x and 3.2.x agents

*Rudder* agent 3.1.x and 3.2.x are compatible with *Rudder* server 4.0.x, but they do not support the new "Audit" policy mode. It is therefore not strictly necessary to update your agents to 4.0.x, unless you want to be able to use the "Audit" policy mode.

### 4.3.2.2 3.0.x or older

These agents are not compatible with *Rudder* 4.0, and you have to upgrade them. Be careful to follow the upgrade path explained [above](#).

## 4.3.3 Protocol for reporting

*Rudder* 3.1 uses syslog messages over UDP by default for reporting, 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 affect the reliability of policy enforcement, but may only temporarily affect reporting on the server. Read [performance notes](#) about rsyslog for detailed information.

## 4.3.4 Known issues

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

## 4.4 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-4.0/ $(lsb_release -cs) main" > /etc/apt/sources.list.d/rudder.list
```

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

```
apt-get update
```

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

- With `apt-get`:

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

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

```
service rudder-jetty restart
```

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

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

**Warning**

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

---

You can now **upgrade your local techniques**.

## 4.5 On RHEL or CentOS

Following commands are executed as the `root` user.

Update your yum repository:

```
echo '[Rudder_4.0]
name=Rudder 4.0 Repository
baseurl=http://www.rudder-project.org/rpm-4.0/RHEL_$releasever/
gpgcheck=1
gpgkey=http://www.rudder-project.org/rpm-4.0/RHEL_$releasever/repodata/repomd.xml.key' > /etc/yum.repos.d/rudder.repo
```

---

**Tip**

Replace *RHEL\_7* with your *Enterprise* Linux version if necessary.

---

### 4.5.1 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
```

### 4.5.2 Rudder agent

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

```
yum update rudder-agent
```

You can now **upgrade your local techniques**.

---



## 4.6 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-4.0/SLES_11_SP1/ Rudder
```

- On a SLES 10 system:

```
zypper sa "http://www.rudder-project.org/rpm-4.0/SLES_10_SP3/" Rudder
```

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

```
zypper ref
```

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

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



### Warning

SLES 11 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 no warranties are made that this will hold in the future. It is really recommended to migrate to PostgreSQL 9.2 at least.

Please look at [Install Rudder Root server on SLES](#) for details.

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

```
service rudder-jetty restart
```

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

```
zypper update rudder-agent
```

You can now [upgrade your local techniques](#).

## 4.7 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.

## 4.8 Upgrade manually installed relays

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 instructions:

- 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.

## Chapter 5

# Rudder Web Interface

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

### 5.1 Authentication

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

You can change the user accounts by following the [User management](#) procedure.

### 5.2 Presentation of Rudder Web Interface

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

#### 5.2.1 Rudder Home

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

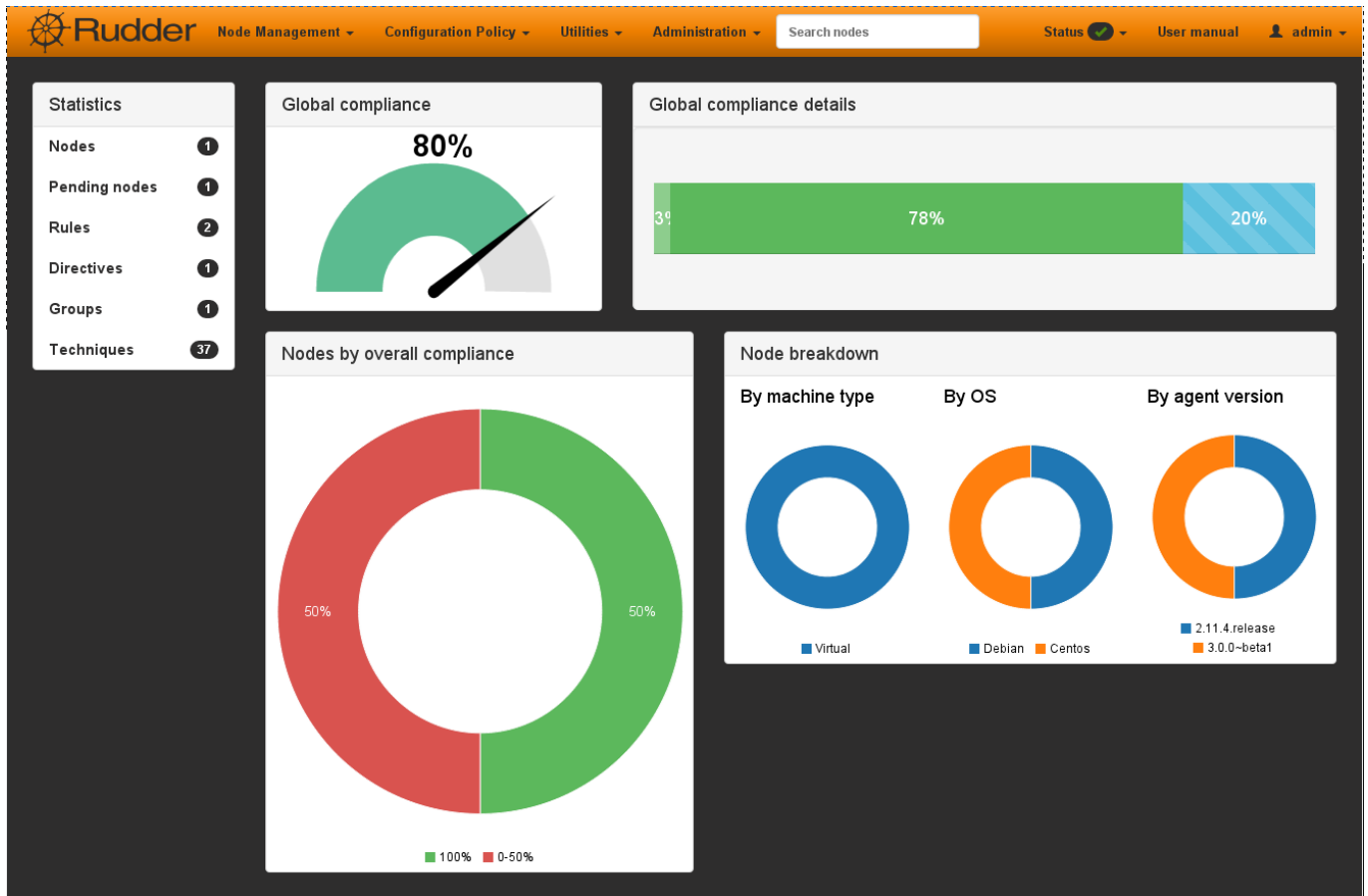


Figure 5.1: Rudder Homepage

### 5.2.2 Node Management

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

**ACCEPT NEW NODES**

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

**Review new nodes** **History**

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

Select/deselect all ☐

Show  entries

Node name	Operating system	IP addresses	Since	Directive
▶ android-box	Android XXX	192.168.1.4	2014-12-04 15:34	<input type="checkbox"/>
▶ centos-6-64	CentOS Linux release 6.0 (Final)	192.168.100.34 127.0.0.1	2014-12-04 15:34	<input type="checkbox"/>
▶ debian-5-32.labo.normation.com	Debian GNU/Linux 5.0.8 (lenny)	192.168.100.12 127.0.0.1	2014-12-04 15:34	<input type="checkbox"/>
▶ node1.rudder.local	Ubuntu 12.04.4 LTS	127.0.0.1 10.0.2.15 192.168.42.11	2014-12-04 15:34	<input type="checkbox"/>
▶ sles-10-64-sp3.labo.normation.com	SUSE Linux Enterprise Server 10 (x86_64)	192.168.100.38 127.0.0.1	2014-12-04 15:34	<input type="checkbox"/>
▶ sles-11-sp1-64.labo.normation.com	SUSE Linux Enterprise Server 11 (x86_64)	192.168.100.29 127.0.0.1	2014-12-04 15:34	<input type="checkbox"/>
▶ sovma136.labo.normation.com	AIX 5.3	127.0.0.1 10.168.168.136 192.168.253.136 10.255.254.136	2014-12-04 15:34	<input type="checkbox"/>
▶ win-a18cnplov5	Microsoft Windows Server 2012 Standard	10.104.14.5	2014-12-04 15:34	<input type="checkbox"/>

Showing 1 to 8 of 8 entries

First Previous 1 Next Last

**Accept into Rudder** **Refuse**

Figure 5.2: Accept new nodes screen

### 5.2.3 Configuration Management

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

**CATEGORIES**

**RULE**

**New Category**

**New Rule** ☒ Display Rules from subcategories

Name	Category	Status	Compliance	Recent changes
<a href="#">Configure users on all nodes</a>		In application	100%	<input type="button" value="Edit"/>
<a href="#">Global configuration for all nodes</a>		Not applied		<input type="button" value="Edit"/>
<a href="#">Set the MOTD on all CentOS machines</a>		In application	100%	<input type="button" value="Edit"/>
<a href="#">test CR</a>		Not applied		<input type="button" value="Edit"/>

Show  entries

Showing 1 to 4 of 4 entries

First Previous 1 Next Last

Figure 5.3: Configuration Policy (Rules) screen

## 5.2.4 Administration

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

**CLIENT-SERVER COMMUNICATION**

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

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

Delete Allowed network

192.168.130.0/24

Add a network Save changes

**Security**

Require time synchronization between nodes and policy server: ☒

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

Save changes

**CONFIGURE COMPLIANCE MODE**

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

☒ Full compliance

☐ Changes only

Save Changes

**AGENT RUN SCHEDULE**

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

Run agent every: 5 minut

First run time

Hour 0 Minute 0

Figure 5.4: Settings screen

## 5.3 Units supported as search parameters

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

### 5.3.1 Bytes and multiples

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

### 5.3.2 Convenience notation

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

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

The following number is not valid:

```
_1234
```

### 5.3.3 Supported units

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

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

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

Table 5.1: Units supported by Rudder search engine

## Chapter 6

# Node Management

### 6.1 Node Inventory

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

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

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

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

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

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

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

### 6.2 Accept new Nodes

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

---

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

---

1. Install and configure the *Rudder Agent* on the new *Node* `debian-node.rudder-project.org`
  2. Wait a few minutes for the first run of the *Rudder Agent*.
  3. Navigate to **Node Management > Accept new Nodes**.
-



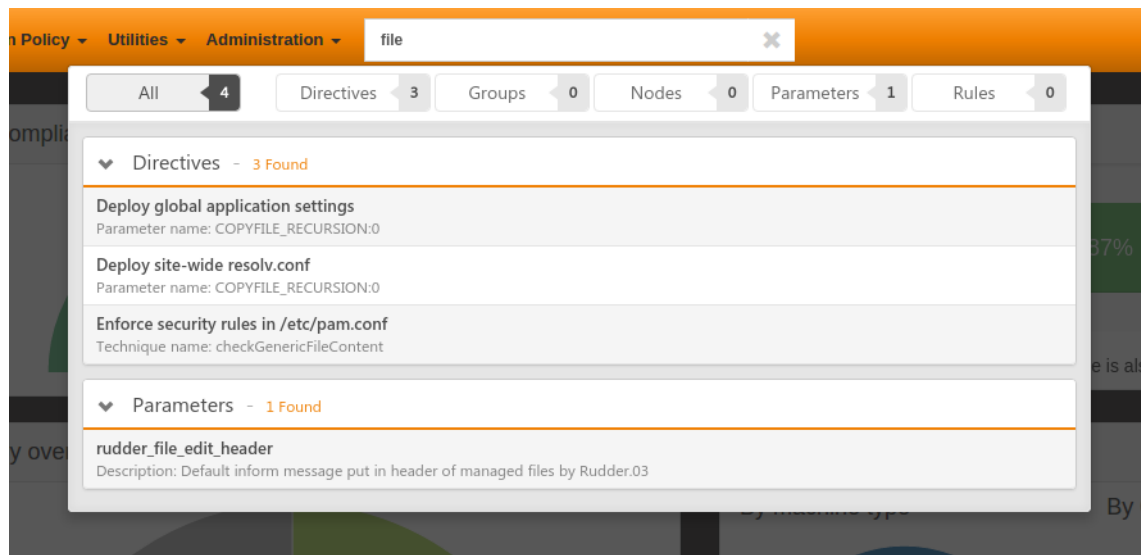
4. Select the new *Node* in the list.
5. Validate the *Node*.
6. The *Node* is now integrated in *Rudder*, you can search it using the search tools.

## 6.3 Search Nodes

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

### 6.3.1 Quick Search

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...



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:

keyword	Description
node	<i>Nodes</i>
group	<i>Groups</i>
parameter	<i>Parameters</i>
directive	<i>Directives</i>
rule	<i>Rules</i>

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 ( <i>Windows</i> Server 2008, <i>Debian</i> ...)
os_version	OS versions (8, 2008 R2, ...)
os	OS Full Names ( <i>Debian</i> GNU/Linux 6.0.10 (squeeze)...) )
os_kernel_version	OS Kernel versions (3.16, 5.1...)
os_service_pack	OS Service Packs (for <i>Windows</i> and <i>SuSE</i> 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	<i>Node</i> properties (arbitrary key=values associated to a node)
rudder_roles	<i>Rudder</i> 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	<i>Directive</i> parameter names, as in the <i>Techniques</i> metadata.xml ("GENERIC_FILE_CONTENT_PATH"...)
dir_param_value	<i>Directive</i> parameter values
technique_id	<i>Technique</i> IDs
technique_name	<i>Technique</i> names ("Enforce a file content"...)
technique_version	<i>Technique</i> version

Table 6.5: in: keywords (directives)

keyword	Search for
parameter_name	<i>Parameter</i> names
parameter_value	<i>Parameter</i> values

Table 6.6: in: keywords (parameters)

keyword	Search for
directives	<i>Rules</i> containing those <i>Directive</i> IDs
groups	<i>Rules</i> containing those <i>Group</i> IDs

Table 6.7: in: keywords (rules)

---

**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]`.
- 

---

**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.

---

**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 Management

### 7.1 Techniques

#### 7.1.1 Concepts

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

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

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

#### 7.1.2 Manage the Techniques

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

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

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

---

#### Tip

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

---



#### Warning

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

---

### 7.1.3 Available Techniques

#### 7.1.3.1 Application management

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

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

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

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

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

#### 7.1.3.2 Distributing files

**Copy a file** Copy a file on the machine

**Distribute ssh keys** Distribute ssh keys on servers

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

#### 7.1.3.3 File state configuration

**Set the permissions of files** Set the permissions of files

#### 7.1.3.4 System settings: Miscellaneous

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

#### 7.1.3.5 System settings: Networking

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

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

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

**NFS Server** Configure a NFS server

### 7.1.3.6 System settings: Process

**Process Management** Enforce defined parameters on system processes

### 7.1.3.7 System settings: Remote access

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

### 7.1.3.8 System settings: User management

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

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

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

## 7.2 Directives

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

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

The screen is divided in three parts:

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

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

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

---

#### Example 7.1 Create a Directive for Name resolution

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

---

## 7.3 Rules

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

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

## 7.4 Variables

### 7.4.1 User defined parameters

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

**LIST OF GLOBAL PARAMETERS**

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

Add Parameter

Show **10** entries

Name	Value	Change
▶ rudder_file_edit_header	### Managed by Rudder, edit with care ###	<button>Edit</button> <button>Delete</button>

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

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

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

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

### 7.4.2 System variables

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

The information about a *Node*:

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

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

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



## 7.5 Compliance

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

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

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

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

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

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

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

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

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

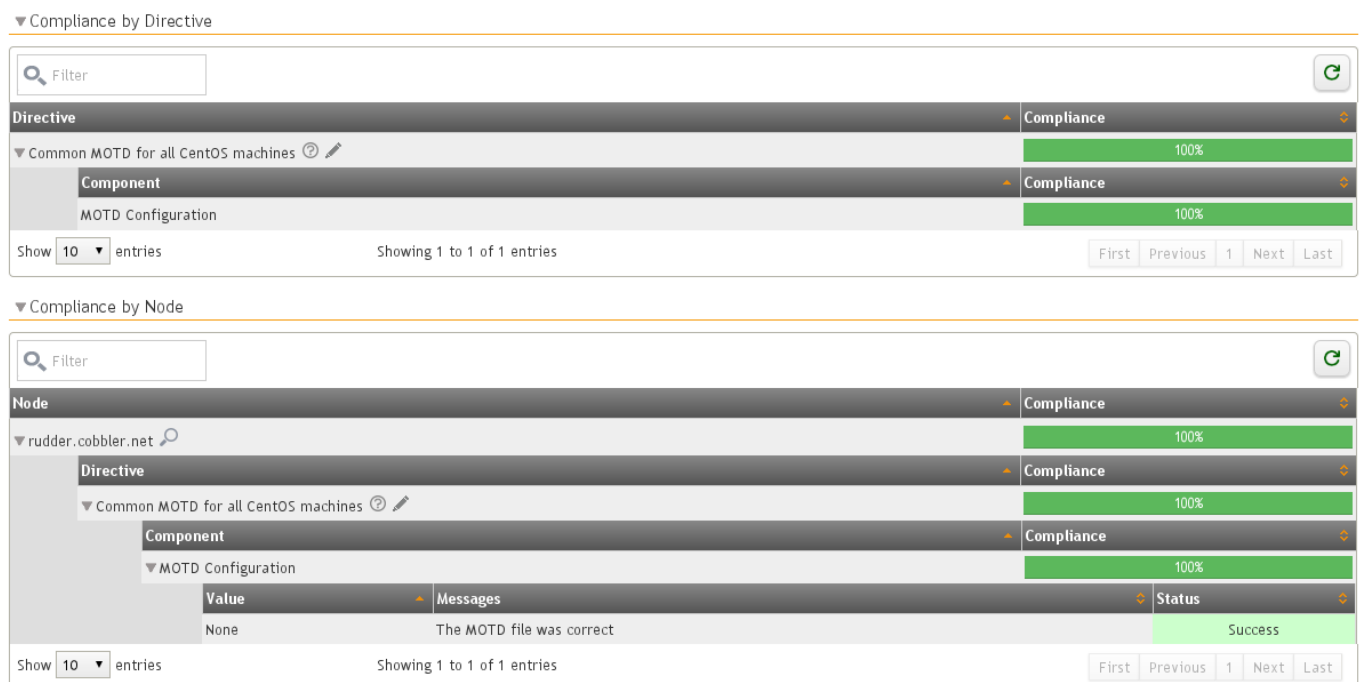
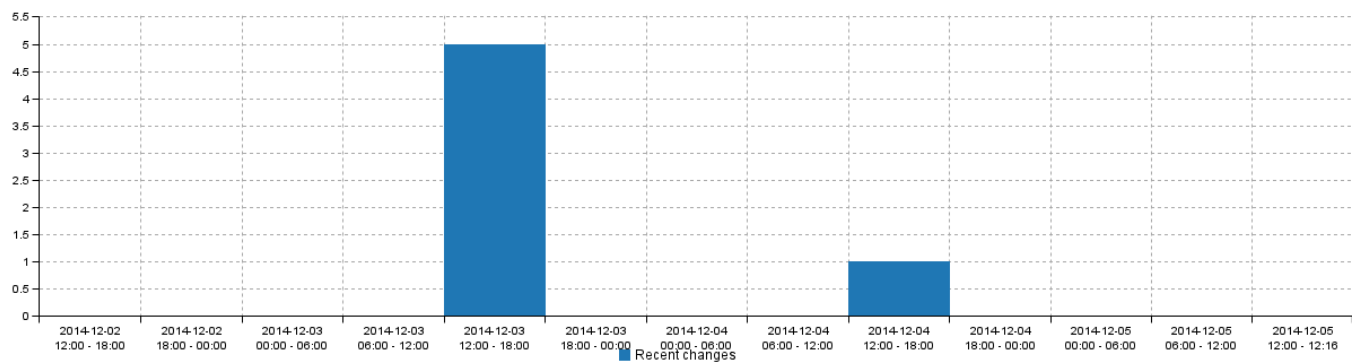


Figure 7.1: Compliance on a Rule

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

## ▼ Recent changes



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

Show 10 entries Showing 1 to 6 of 6 entries

First Previous 1 Next Last

Figure 7.2: Compliance history on a Rule

## 7.6 Validation workflow in Rudder

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

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

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

### CONFIGURE CHANGE REQUESTS (VALIDATION WORKFLOW)

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

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

Enable Change Requests: ☐

Allow self validation: ☐

Allow self deployment: ☒

Save changes

### 7.6.1 What is a Change request ?

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

A Change request has:

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

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

#### 7.6.1.1 Change request status

There is 4 Change request status:

##### Pending validation

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

##### Pending deployment

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

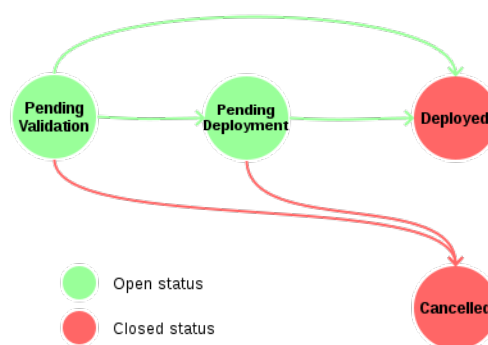
##### Deployed

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

##### Cancelled

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

Here is a diagram about all those states and transitions:



### 7.6.1.2 Change request management page

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

The screenshot shows the Rudder web interface. The top navigation bar includes 'Node Management', 'Configuration Policy', 'Utilities', and 'Administration'. A search bar for nodes is present. The 'CHANGE REQUESTS' section has a status filter set to 'Open'. Below the filter, a table lists two requests:

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). Navigation buttons: First, Previous, 1, Next, Last.

### 7.6.1.3 Change request detail page

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

The screenshot shows the detail page for 'CR #7: Update Rule Set the MOTD on all machines'. The status is 'Pending deployment'. The page includes a 'Back to change request list' link and a 'Pending deployment' badge. The form shows the title, state, ID, and description. A 'Change history' section displays a table of actions:

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 <u>Set the MOTD on all machines</u>	admin	2014-12-05 12:28	

Showing 1 to 3 of 3 entries. Navigation buttons: First, Previous, 1, Next, Last. Buttons at the bottom: Decline, Deploy.

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:	<div></div>

**Update**

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

- History about that change request

Change history
Diff

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 <a href="#">Configure users on all nodes</a>	admin	2014-12-03 12:02	

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

- Display the change proposed

Change history
Diff

**Rule overview:**


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

### 7.6.2 How to create a Change request ?


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

The change message will be used as description for you Change Request, so we advise to fill it anyway to keep an explanation about 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  entries
Showing 1 to 1 of 1 entries

First
Previous
1
Next
Last


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

Change request title: \*

Please enter a message explaining the reason for this change.

Message: \*












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

Cancel

Submit for Validation

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

Here is a small table about all possibilities:

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

## 7.6.3 How to validate a Change request ?

### 7.6.3.1 Roles

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

**Validator** Can validate Change request

**Deployer** To deploy Change Request

Both of those roles:

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

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

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

Please enter a message explaining the reason for this change.

**Message:**

### 7.6.3.2 Self Validations

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

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

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

### 7.6.4 Change request and conflicts

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

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

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

Title: \*

State:

ID:

Description:



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

- Changes
  - Directives
  - Rules
    - Set the MOTD on all CentOS machines
  - Groups
  - Global Parameters

Change history

Diff

Show  entries

Action	Actor	Date	Reason
Status changed from Pending validation to Pending deployment	admin	2014-12-05 12:28	
Change request created	admin	2014-12-05 12:28	
Modify rule <a href="#">Set the MOTD on all machines</a>	admin	2014-12-05 12:28	

Showing 1 to 3 of 3 entries

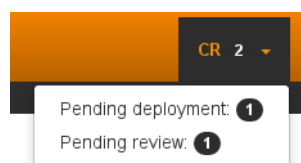
For now, we decided to reduce to the possibility of an error or inconsistency when there are concurrent changes. In a future version of *Rudder*, there will be a system to handle those conflicts, and make sure actual changes are not overwritten.

### 7.6.5 Notifications:

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

#### 7.6.5.1 Pending change requests

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



#### 7.6.5.2 Change already proposed on Rule/Directive/Group

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

Compliance detail

Edit parameters

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 Technique editor

### 7.7.1 Introduction

#### 7.7.1.1 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.

#### 7.7.1.2 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...)

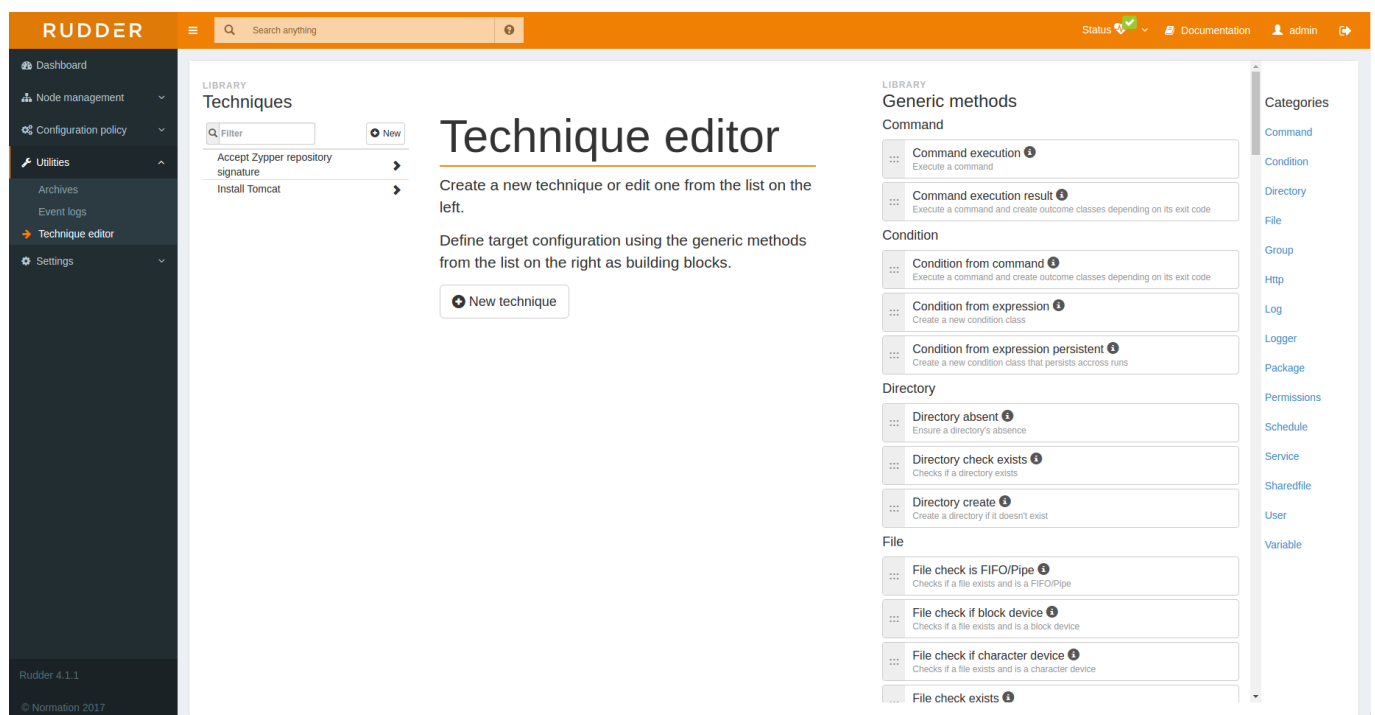
### 7.7.2 Technique Editor

#### 7.7.2.1 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 respond to all of the specific needs of each client. That's why *Rudder* integrates the **Technique editor**, a tool to create advanced *Techniques*. Directly accessible from Rudder 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*.

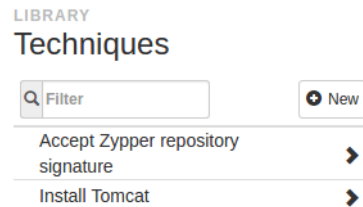
#### 7.7.2.2 Interface

Here is an overview of its interface :



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. These *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

Clone Delete

▼ General information

**Name:**

**Description:**

**Bundle name:**

**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](http://www.rudder-project.org/doc/_generic_methods.html))

The screenshot displays the 'LIBRARY' section titled 'Generic methods'. It is organized into categories: Command, Condition, Directory, and File. Each category contains several methods, each with a three-dot menu icon, a title, and a brief description. To the right of the main list is a vertical sidebar titled 'Categories' which lists the same categories as links: Command, Condition, Directory, File, Group, Http, Log, Logger, Package, Permissions, Schedule, Service, Sharedfile, User, and Variable. The 'Command' category is currently selected, and its methods are visible in the main list.

**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

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:

TECHNIQUE

Configure NTP

Clone Delete

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

METHOD

Package install

Reset

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

Add methods

Reset Save

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.

## 7.7.3 Create your first Technique

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

### 7.7.3.1 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.*

### 7.7.3.2 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*.

### 7.7.3.3 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.

## 7.8 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>>.

## Global policy mode

Audit

Enforce

## 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 **audit**.

### Override policy mode for this node

Global mode (audit)

Audit

Enforce

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

Node by node

Policy by policy

This mode can be set:

- Globally on the *Rudder* root server. In this 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.

### 7.8.1 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 overridden to **Audit** or **Enforce**, the **D** value is used.
    - If **D** uses the **global** value and **N** is overridden to **Audit** or **Enforce**, the **N** value is used.
    - If **N** and **D** are overridden 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
-

## 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 occurrence 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 occurrence 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:	<input type="text" value="/etc/sysconfig/sysctl"/>
Enforce the content of the file: ?	<input type="checkbox"/>
Enable the deletion of lines using a regexp:	<input checked="" type="checkbox"/>
Enable the creation of the file if it doesn't exist:	<input checked="" type="checkbox"/>
Enforce the content of the file only at creation: ?	<input type="checkbox"/>
Enable the replacement of lines using a regexp:	<input checked="" type="checkbox"/>
Limit file modification to a zone of the file: ?	<input type="checkbox"/>

▼ Section: File content

ENABLE\_SYSRQ="yes"

Content of the file (optional): ?

▼ Section: Line deletion regular expressions

Regular expression: ?

▼ Section: Line replacement regular expressions

Regular expression: ?

String used as a replacement (optional):

## 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 transferred 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 listening on loopback) and from *Rudder*-specific users, which passwords are 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 transferred without 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 distributed in the `ncf` package
- the *Rudder* techniques sources (without parameters), which includes all the content distributed 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 containing 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 hostname 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 without 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.
-

## Chapter 9

# Administration

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

### 9.1 Archives

#### 9.1.1 Archive usecases

The archive feature of *Rudder* allows to:

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

##### 9.1.1.1 Changes testing

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

##### 9.1.1.2 Changes qualification

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



#### Versions of the Rudder servers

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

---

#### 9.1.2 Concepts

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

---

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

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

### 9.1.3 Archiving

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

To perform a global archive, the steps are:

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

### 9.1.4 Importing configuration

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

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

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

---

#### Tip

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

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

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

### 9.1.5 Deploy a preconfigured instance

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

## 9.2 Event Logs

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

### Event Categories

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

## 9.3 Policy Server

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

### 9.3.1 Configure allowed networks

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

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

### 9.3.2 Clear caches

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

### 9.3.3 Reload dynamic groups

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

---

## 9.4 Plugins

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

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

### 9.4.1 Install a plugin

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

**/opt/rudder/share/rudder-plugins/** This directory contains the JAR files of the plugins.

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

Then, register the plugin, using its name without the ".jar" extension and restart *Rudder*:

```
# register plugin
/opt/rudder/bin/rudder-plugin register plugin-name-without-jar-extension
# restart Rudder
/etc/init.d/rudder-jetty restart
```

## 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 Password upgrade

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

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

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

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

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

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

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

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

## 9.7 User management

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



## 9.7.1 Configuration of the users using a XML file

### 9.7.1.1 Generality

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

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

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

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

```
service rudder-jetty restart
```

### 9.7.1.2 Passwords

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

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

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

Table 9.1: Hashed passwords algorithms list

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

Here is an example of authentication file with hashed password:

```
<authentication hash="sha256">

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

</authentication>
```

## 9.7.2 Configuring an LDAP authentication provider for Rudder

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

### 9.7.2.1 LDAP is only for authentication

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

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

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

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

```
service rudder-jetty restart
```

### 9.7.2.2 Enable LDAP authentication

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

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

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

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

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

Here are some usage examples,

on standard *LDAP*:

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

on *Active Directory*:

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

### 9.7.3 Authorization management

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

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

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

#### 9.7.3.1 Pre-defined roles

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

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

Example: "rule\_only,administration\_only" will only give access to the "Administration" tab as well as the *Rules*.

#### 9.7.3.2 Custom roles

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

A permission is composed of a type and a level:

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

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

Usage example:

- configuration\_read → Will give read access to the configuration (*Rule* management, *Directives* and *Parameters*)
- rule\_write, node\_read → Will give read and write access to the *Rules* and read access to the *Nodes*

### 9.7.4 Going further

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

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

## 9.8 Monitoring

This section will give recommendations for:

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

### 9.8.1 Monitoring Rudder itself

#### 9.8.1.1 Monitoring a Node

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

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*.

#### 9.8.1.2 Monitoring a Server

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

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

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

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

### 9.8.2 Monitoring your configuration management

There are two interesting types of information:

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

### 9.8.2.1 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.

### 9.8.2.2 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*).

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

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

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

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

## 9.9 Use Rudder inventory in other tools

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

### 9.9.1 Export to a spreadsheet

You can export the list of your nodes to a spreadsheet file (xls format) by using a [tool](#) available in the *rudder-tools* repository. Simply follow the installation instructions, and run it against your *Rudder* server. You will get a file containing:

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

You can easily modify the script to add other information.

### 9.9.2 Use the inventory in Rundeck

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

### 9.9.3 Use the inventory in Ansible

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

## Chapter 10

# Usecases

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

### 10.1 Dynamic groups by operating system

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

### 10.2 Library of preventive policies

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

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

### 10.3 Standardizing configurations

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

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

### 10.4 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.

---

## 10.5 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 11

# Advanced usage

This chapter describe advanced usage of *Rudder*.

### 11.1 Node management

#### 11.1.1 Reinitialize policies for a Node

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

```
rudder agent reset
```

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

**Caution**

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

---

#### 11.1.2 Completely reinitialize a Node

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

**Warning**

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

---

The command to reinitialize a *Node* is:

```
rudder agent reinit
```

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

---

### 11.1.3 Change the agent run schedule

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



#### Warning

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

---

### 11.1.4 Installation of the Rudder Agent

#### 11.1.4.1 Static files

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

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

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

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

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

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

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

#### 11.1.4.2 Generated files

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

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

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

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

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

#### 11.1.4.3 Services

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

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

**cf-serverd** This *CFEngine Community* daemon is listening on the network on *Rudder Root* and Relay servers, serving policies and files to *Rudder Nodes*.

---

#### 11.1.4.4 Configuration

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

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

#### 11.1.5 Rudder Agent interactive

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

```
rudder agent run
```

---

##### Error: the name of the Rudder Root Server can't be resolved

If the *Rudder Root Server* name is not resolvable, the *Rudder Agent* will issue this error:

```
rudder agent run
```



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

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

The *Rudder Root Server* name is defined in this file

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

---

##### Error: the CFEngine service is not responding on the Rudder Root Server

If the *CFEngine* is stopped on the *Rudder Root Server* you will get this error:

```
# rudder agent run
!! Error connecting to server (timeout)
!!! System error for connect: "Operation now in progress"
!! No server is responding on this port
Unable to establish connection with rudder-root
```



Restart the *CFEngine* service:

```
service rudder-agent restart
```

---

#### 11.1.6 Processing new inventories on the server

##### 11.1.6.1 Verify the inventory has been received by the Rudder Root Server

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

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

### 11.1.6.2 Process incoming inventories

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

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

```
rudder agent run
```

### 11.1.6.3 Validate new Nodes

User interaction is required to validate new *Nodes*.

### 11.1.6.4 Prepare policies for the Node

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

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

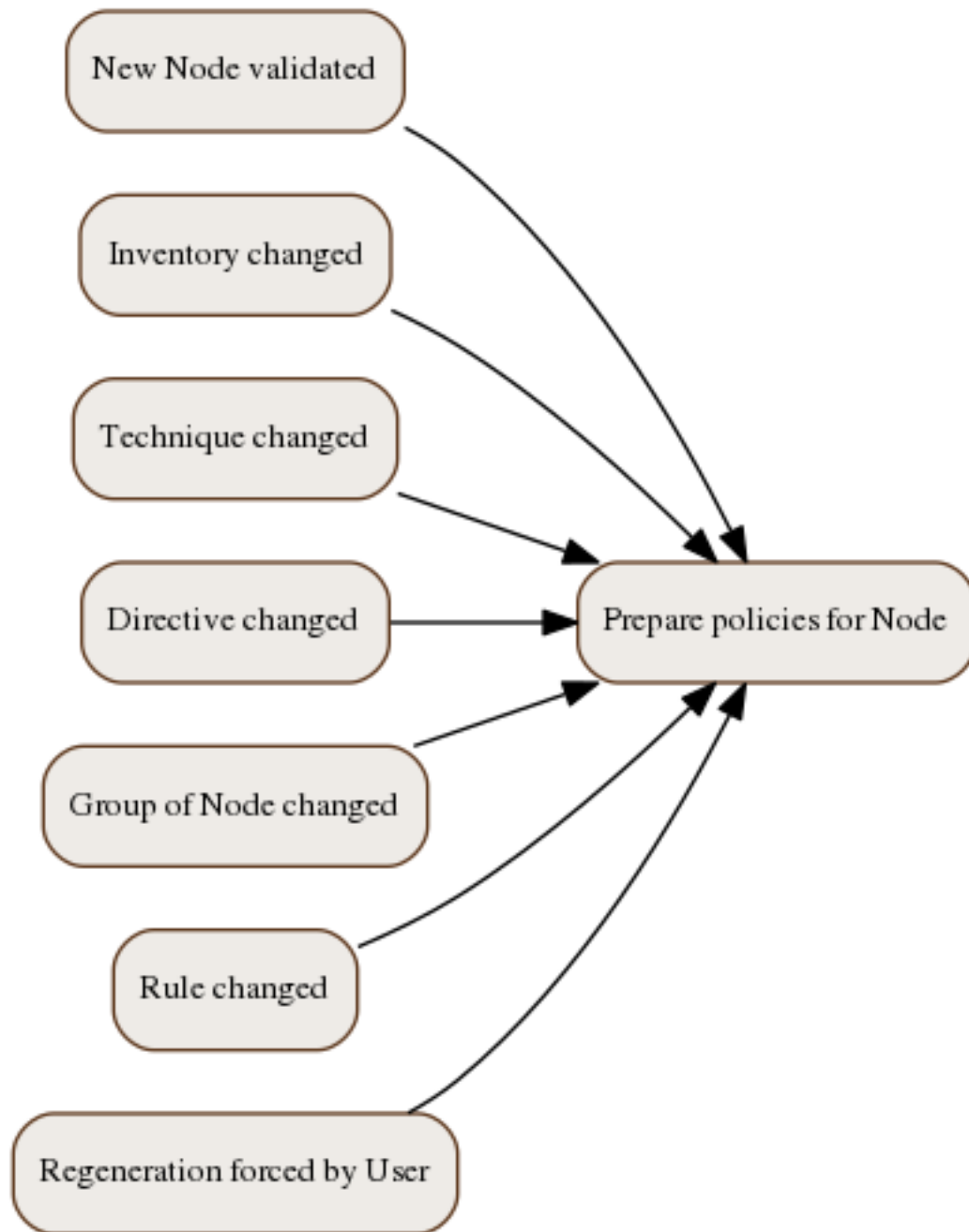


Figure 11.1: Generate policy workflow

## 11.1.7 Agent execution frequency on nodes

### 11.1.7.1 Checking configuration (CFEngine)

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

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

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

### 11.1.7.2 Inventory (FusionInventory)

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

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

## 11.2 Password management

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

### 11.2.1 Configuration of the postgres database password

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

Here is a semi-automated procedure:

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

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

- Update the Postgres database user

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

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

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

### 11.2.2 Configuration of the OpenLDAP manager password

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

Here is a semi-automated procedure:

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

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

- Update the password in the slapd configuration

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

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

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

### 11.2.3 Configuration of the WebDAV access password

This time, the procedure is a bit more tricky, as you will have to update the *Technique* library as well as a configuration file. Here is a semi-automated procedure:

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

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

- Update the password in the apache htaccess file

---

#### Tip

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

---

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

- Update the password in *Rudder's* system *Techniques*

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

- Update the *Rudder Directives* by either reloading them in the web interface (in the "Configuration Management/*Techniques*" tab) or restarting jetty (NOT recommended)

## 11.3 Policy generation

Each time a change occurs in the *Rudder* interface, having an impact on the 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.

### 11.3.1 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.

## 11.4 Technique creation

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

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

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

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

---

### 11.4.1 Recommended solution: Technique Editor

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

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

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

#### 11.4.1.1 Using the Technique Editor

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

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

#### 11.4.1.2 Logs

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

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

### 11.4.2 Understanding how Technique Editor works

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

#### 11.4.2.1 Directory layout

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

ncf directory structure exists in two root folders:

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



### 11.4.3 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.

#### 11.4.3.1 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* server 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.

### 11.4.3.2 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
```

## 11.4.4 Create Technique manually

### 11.4.4.1 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!

### 11.4.4.2 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*.

#### 11.4.4.3 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 its directory name, so choose wisely:

```
mkdir sampleTechnique
```

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

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

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

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

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

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

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

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

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

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

#### 11.4.4.4 Define variables

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

#### 11.4.4.5 First test in the Rudder interface

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

#### 11.4.4.6 Implement the behavior

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

#### 11.4.4.7 Read in the variables from Rudder

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

#### 11.4.4.8 Add reporting

**WORK IN PROGRESS** The reports format Cf <http://www.rudder-project.org/foswiki/Development/ReportsInTechniques>

## 11.5 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-updateNodeProperties>

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" }`

### 11.5.1 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 [11.6](#).

In a mustache template, use:

```
{{vars.node.properties.datacenter}}
```

## 11.5.2 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.

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 overridden by the same key in a different namespace
- a key that is overridden 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]}
```

## 11.6 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](#).

### 11.6.1 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.

## 11.6.2 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 (`[,],{,}`..)

### 11.6.2.1 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}" }
```

### 11.6.2.2 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 replace this expression by its value if it's defined, or kept as is if it's unknown.

The variable's content is read from `/var/rudder/cfengine-community/inputs/properties.d/properties.json`. You can find more information on node properties in [Section 11.5](#).

## 11.7 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.

### 11.7.1 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.

### 11.7.2 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
- 

### 11.7.3 Rudder utility library

#### 11.7.3.1 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.

#### 11.7.3.2 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.

### 11.7.4 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.



## 11.8 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 is 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 needs to be committed 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]) ---
```

## 11.9 REST API

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

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

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



### Warning

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

### 11.9.1 Default setup

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

#### 11.9.1.1 Rudder Authentication

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

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

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

### 11.9.1.2 Apache access rules

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

---

**Example 11.1** Example usage of non authenticated REST API

---

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

---

### 11.9.1.3 User for REST actions

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

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

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

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

## 11.9.2 Status

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

### 11.9.3 Promises regeneration

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

### 11.9.4 Dynamic groups regeneration

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

### 11.9.5 Technique library reload

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

## 11.9.6 Archives manipulation

Various methods are available to import and export items:

### 11.9.6.1 Archiving:

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

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

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

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

---

### 11.9.6.2 Listing:

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

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

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

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

### 11.9.6.3 Restoring a given archive:

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

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

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

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

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

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

### 11.9.6.5 Restoring the latest available commit (use Git HEAD):

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

### 11.9.6.6 Downloading a ZIP archive

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

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

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

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

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

---

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

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

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

## 11.10 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.

## 11.10.1 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
```

## 11.10.2 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.

## 11.11 Multiserver Rudder

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

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

### 11.11.1 Preliminary steps

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

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

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

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

Add firewall rules:

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

### 11.11.2 Install rudder-relay-top

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

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

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

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

### 11.11.3 Install rudder-db

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

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

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

### 11.11.4 Install rudder-ldap

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

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

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

### 11.11.5 Install rudder-web

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

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

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

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

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

## 11.12 Server migration

### 11.12.1 What files you need

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

You need to keep:

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

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

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

You will have to accept nodes

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

### 11.12.2 Handle configuration files

#### 11.12.2.1 Copy /var/rudder/configuration-repository

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

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



### 11.12.2.2 Use Archive feature of Rudder

Alternatively, you can follow the Archive/Import procedures described in [Archives](#)

### 11.12.3 Handle CFEngine keys

#### 11.12.3.1 Keep your CFEngine keys

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

#### 11.12.3.2 Change CFEngine keys

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

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

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

### 11.12.4 On your nodes

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

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

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

Your configuration is now totally migrated.

## 11.13 Mirroring Rudder repositories

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

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

To synchronize with the APT repository just type:

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

To synchronize with the RPM repository just type:

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

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

# Chapter 12

## Handbook

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

### 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 than 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-slaped stop

# Reindex the databases
service rudder-slaped reindex

# Restart OpenLDAP
service rudder-slaped 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/slappcat -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/alock /var/rudder/ldap/openldap-data/*.bdb /var/ ←
rudder/ldap/openldap-data/__db* /var/rudder/ldap/openldap-data/log*

# Import your backups

#~Configuration repository
tar -C /var/rudder -zxvf /root/rudder-backup-XXXXXXXXX.tar.gz

#~LDAP backup
/opt/rudder/sbin/slappadd -l /root/rudder-backup-XXXXXXXXX.ldif

#~PostgreSQL backup
sudo -u postgres psql -d rudder < /root/rudder-backup-XXXXXXXXX.sql
#~or
psql -u rudder -d rudder -W < /root/rudder-backup-XXXXXXXXX.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.

Please remember that The *CFEngine* key restoration is mandatory for the clients to update properly, but if the *Rudder* server address changes, the agents will block. You have to delete every root-\*.pub key in /var/rudder/cfengine-community/ppkeys/ for things to work again.

## 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.OutOfMemoryError: 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 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 nfile 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 nfile 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
```

### 12.3.8.2 Network backlog

You can also have issues with the network queues (which may for example lead to sending SYN cookies):

- You can increase the maximum number of connection requests awaiting acknowledgment by changing *net.ipv4.tcp\_max\_syn\_backlog = 4096* (for example, the default is 1024) in */etc/sysctl.conf*.
- You may also have to increase the socket listen() backlog in case of bursts, by changing *net.core.somaxconn = 1024* (for example, default is 128) in */etc/sysctl.conf*.

### 12.3.8.3 Conntrack table

You may reach the size of the conntrack table, especially if you have other applications running on the same server. You can increase its size in */etc/sysctl.conf*, see [the Netfilter FAQ](#) for details.

## Chapter 13

# Troubleshooting and common issues

This chapter covers common issues and the available solutions.

### 13.1 Some reports are in "No report"

#### 13.1.1 If you get no reports at all for the Node

First thing to check is to see if reports were received by *Rudder* server.

Check the last report time (called **Last seen**) in **List Nodes** page. If you see:

- **Never:** your *Node* is misconfigured or has a communication issue with the server
- A date far (more than 15 minutes) from current time: Synchronize server and node time
- A recent date: check if the node has correctly updated

Now we will check if promises were updated on the *Node*. Maybe the node could not update its promises anymore, even if the reporting looks ok and *Rules* seems to be applied but report keeps in 'No report'.

To check if a node can update its promises, run (on the node) *rudder agent run*. You'll get the result of the 'Update' component in the execution (in the *Common Technique*).

To update its promises, a *Node* needs to get a directory on *Rudder* server (*/var/rudder/share/node\_uuid*), and *Rudder* checks if the node is authorized to access that directory. This check is based on the capability to resolve the ip as an accepted node. So if your node can't update its promises, it's probably because of a DNS issue!

#### 13.1.2 If you get incomplete reporting for the Node

When the agent does not perform a normal execution, the reports will be incomplete, and some components will appear as missing on the server. Reasons could be:

- the execution of the agent encountered an error during its execution and could not complete. In this case, an error message is displayed when running *rudder agent run*. It is very likely a bug in the agent, please [report a bug](#)
  - the agent is executed to launch specific bundles (with the *-b* option)
  - reporting is missing on a *Technique*, in this case [report a bug](#)
-

## 13.2 Communication issues between agent and server

### 13.2.1 DNS issues

If one of the following problems happen:

- the agent does not manage to get its configuration back from the server with weird errors
- the server complains about being unable to resolve the node hostname
- when starting or restarting *Rudder* (or *rudder-agent*) service, *cf-serverd* start hangs

You probably have a name resolution problem. Please keep in mind that *Rudder* needs a working name resolution environment to operate properly, and therefore every machine should be at least able to resolve the name of their peer.

You have two options:

- Fix your DNS server or the */etc/hosts* on both the server and the node, so they can resolve each other (you can check using *nslookup*). You need to restart *rudder-agent* on the server to apply it
- Disable hostname checking on the server in **Administration** → **Settings** → **Use reverse DNS lookups on nodes to reinforce authentication to policy server**. This is the preferred solution if you have nodes behind a NAT.

### 13.2.2 Inventory issues

If you cannot send inventories to the server, it may be because of a proxy configured in */etc/profile* or shell configuration. *Rudder* agents use *cURL* to send inventories to their server, and the server actually uses it too to send received inventories to the inventory web application. There are two solutions usable to prevent this problem:

- Disable the proxy temporarily in your shell session, so *Rudder* can operate freely:

```
unset http_proxy; unset https_proxy; unset ftp_proxy; unset ftps_proxy; unset HTTP_PROXY; ↵  
unset HTTPS_PROXY; unset FTP_PROXY; unset FTPS_PROXY
```

- If you are using the Squid proxy, you are in luck, as the workaround might simply be to add this entry to your */etc/squid/squid.conf*: *ignore\_expect\_100 on*, it will make Squid more tolerant to programs like *cURL* than send some terse http requests. (Thanks to Albaro A. for this tip!)

## 13.3 Technique editing

If you have committed an invalid technique, then fixed it and committed it again, but the webapp still doesn't start, you have to force *Technique* library reloading.

To do this, deleting the attribute *techniqueLibraryVersion* from entry *techniqueCategoryId=Active Techniques,ou=Rudder,cn=rudder-configuration* in your *Rudder LDAP* backend. When re-starting, the webapp should now reload latest techniques.

## 13.4 Database is using too much space

*Rudder* stores a lot of data in the Postgresql database, and most historical data is removed from it. You can **configure** how many days of logs you want to keep in the database. However, due to the nature of Postgresql, when data are removed, space is not reclaimed on the storage system, it is simply marked as "free" for the database to write again in the removed rows. This space can be reclaimed by a *VACUUM FULL*, but it needs at least as much free space on the drive as the database size. If you are using Postgresql 8.4, which is not a recommended version, you'll be likely to experience indexes bloating, where the physical size of

the indexes grows without real reason, and need to be regularly purged. With PostgreSQL 9.2 and later, database size should remain stable, based on the number of nodes and configurations applied as described in [this estimation of disk usage](#).

There are two ways to reclaim space, the fast one (which doesn't reclaim completely all wasted space), and the complete one (which is unfortunately very slow)

Fast solution (especially for 8.x version of postgresql): Simply reindexing the database will save some space; depending on the size of your database, it may take several minutes to a couple of hours

```
# First, stop the Rudder server
rudder agent disable
service rudder-jetty stop

# Then log into postgresql
psql -U rudder -h localhost
REINDEX TABLE ruddersysevents;

#Exit postgresql
\q

# Restart the Rudder server
rudder agent enable
service rudder-jetty start
```

Complete solution: this solution will reclaim all that can be reclaimed, but is really really slow (can last several hours)

```
# First, stop the Rudder server
rudder agent disable
service rudder-jetty stop

# Then log into postgresql
psql -U rudder -h localhost
drop index component_idx;
drop index composite_node_execution_idx;
drop index keyvalue_idx;
drop index nodeid_idx;
drop index ruleid_idx;
drop index executiontimestamp_idx;
vacuum full ruddersysevents; -- this will take several hours

create index nodeid_idx on RudderSysEvents (nodeId);
CREATE INDEX executionTimeStamp_idx on RudderSysEvents (executionTimeStamp);
CREATE INDEX composite_node_execution_idx on RudderSysEvents (nodeId, executionTimeStamp);
CREATE INDEX component_idx on RudderSysEvents (component);
CREATE INDEX keyValue_idx on RudderSysEvents (keyValue);
CREATE INDEX ruleId_idx on RudderSysEvents (ruleId);

# Exit postgresql
\q

# Restart the Rudder server
rudder agent enable
service rudder-jetty start
```

## Chapter 14

# Reference

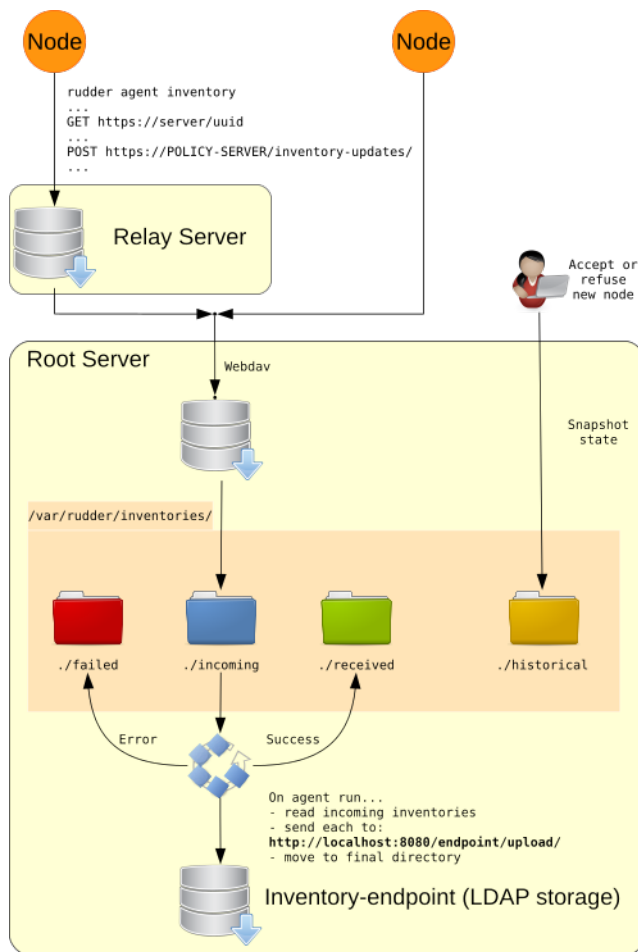
This chapter contains the reference *Rudder* configuration files

### 14.1 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 schema sum up the whole process.



### 14.1.1 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.

### 14.1.2 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.

### 14.1.3 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.

### 14.1.4 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
}
```

## 14.2 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*.



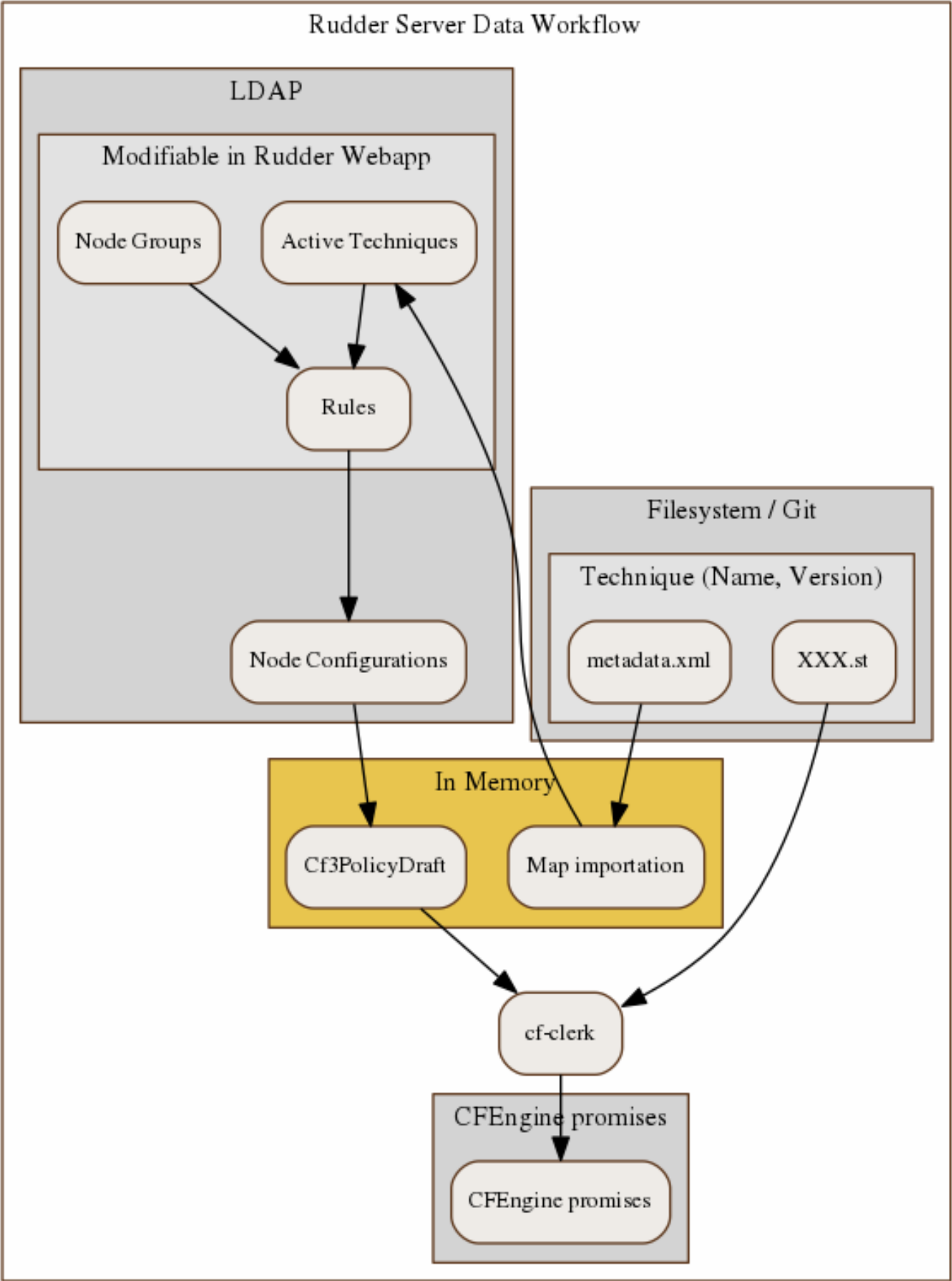


Figure 14.1: Rudder data workflow

## 14.3 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

## 14.4 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 schema summarizes the process that will be described in the next pages.

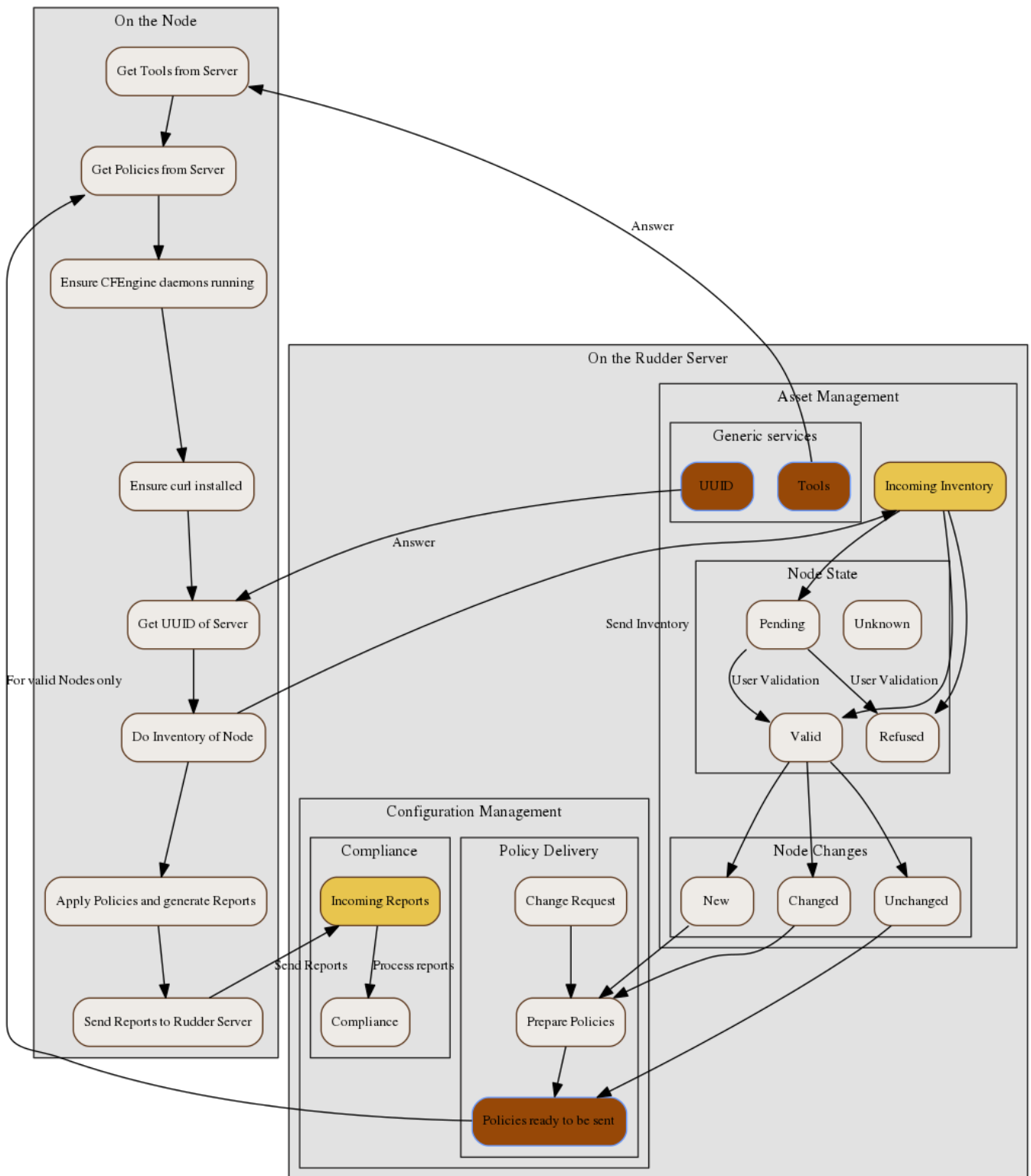


Figure 14.2: Rudder Agent workflow

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:

### 14.4.1 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.

### 14.4.2 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.

### 14.4.3 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`

### 14.4.4 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*.

### 14.4.5 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`.

### 14.4.6 Apply Directives

Apply other policies and write reports locally.

## 14.5 Configuration files for a Node

- `/etc/default/rudder-agent`

## 14.6 Packages organization

### 14.6.1 Packages

*Rudder* components are distributed as a set of packages.

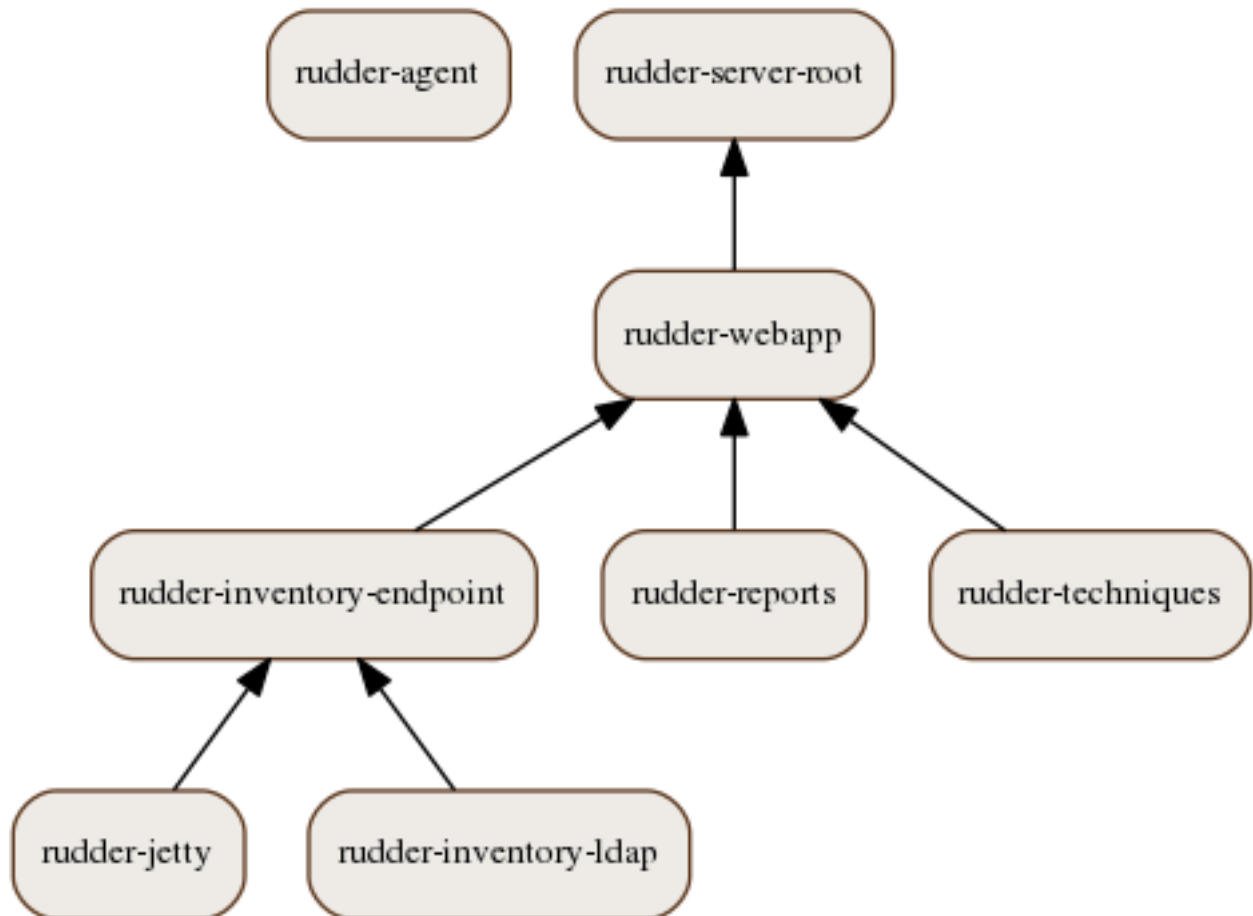


Figure 14.3: Rudder packages and their dependencies

**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 and parses the files sent by *FusionInventory* and insert the valuable data into the *LDAP* database.

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

**rudder-techniques** Package for the *Techniques*. They are installed in `/opt/rudder/share/techniques`. At 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* Community, *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
- liblmbd (On platforms where it is available as a package - on others the `rudder-agent` package bundles it)
- uuidgen

### 14.6.2 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 recommended 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`.

## 14.7 Generic methods

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

### 14.7.1 Command

#### 14.7.1.1 `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}
```

#### 14.7.1.2 `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}
```

## 14.7.2 Condition

### 14.7.2.1 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 bundle will define a class `condition_from_command_${condition_prefix}_{kept,not_ok,ok,reach ed}` depending on the exit codes given in parameters.

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, this will produce an `error` outcome class and no classes with `${condition_prefix}`.

The created condition (class in cfengine speaking) is global to the agent. To make the condition specific to current technique instance (or directive), add `${class_prefix}` within the condition prefix, this variable is automatically defined in your technique and is available if you use *Rudder* agent  $\geq 4.0$  or *CFEngine*  $\geq 3.8$ .

Parameters

- **condition\_prefix:** The condition name, use `${class_prefix}` to create a local condition
- **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}
```

### 14.7.2.2 condition\_from\_expression

Create a new condition class

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This bundle will define a class `condition_from_expression_${condition_prefix}_{kept,ok,reached}`

This bundle will additionally produce a `${condition_prefix}_true` or a `${condition_prefix}_false` condition depending on the result on the expression.

Calling this method with a condition expression transforms a complex expression into a single class condition.

The created condition (class in cfengine speaking) is global to the agent. To make the condition specific to current technique instance (or directive), add `${class_prefix}` within the condition prefix, this variable is automatically defined in your technique and is available if you use *Rudder* agent  $\geq 4.0$  or *CFEngine*  $\geq 3.8$ .

Parameters

- **condition\_prefix:** The condition prefix, use `${class_prefix}_name` to create a local condition
- **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}
```



### 14.7.2.3 condition\_from\_expression\_persistent

Create a new condition class that persists accross runs

Compatible with nodes running *Rudder* 3.1 or higher.

Usage

This bundle will define a class `condition_from_expression_persistent_${condition_prefix}_{kept,ok, reached}`

This bundle will additionnaly produce a `${condition_prefix}_true` or a `${condition_prefix}_false` condition depending on the result on the expression

The created condition (class in cfengine speaking) is global to the agent and is persisted accross runs. The persistence duration is controlled using `${duration}`. There is no way to persist indefinitely. To make the condition specific to current technique instance (or directive), add `${class_prefix}` within the condition prefix, this variable is automatically defined in your technique and is available if you use *Rudder* agent  $\geq 4.0$  or *CFEngine*  $\geq 3.8$

*Parameters*

- **condition\_prefix:** The condition prefix, use `${class_prefix}_name` to create a local condition
- **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}
```

## 14.7.3 Directory

### 14.7.3.1 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}
```

### 14.7.3.2 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}
```

### 14.7.3.3 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}
```

## 14.7.4 File

### 14.7.4.1 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}
```

### 14.7.4.2 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}
```

#### 14.7.4.3 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}
```

#### 14.7.4.4 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 exist

*Parameters*

- **file\_name:** File name (absolute path on the target node)

Classes defined

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

#### 14.7.4.5 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 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}
```

#### 14.7.4.6 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}
```

#### 14.7.4.7 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}
```

#### 14.7.4.8 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}
```

#### 14.7.4.9 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 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}
```

#### 14.7.4.10 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}
```

#### 14.7.4.11 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}
```

#### 14.7.4.12 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}
```

#### 14.7.4.13 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}
```

#### 14.7.4.14 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}
```

#### 14.7.4.15 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}
```

#### 14.7.4.16 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}
```

#### 14.7.4.17 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}
```

#### 14.7.4.18 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}
```

#### 14.7.4.19 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
- **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}
```

#### 14.7.4.20 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}
```



#### 14.7.4.21 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}
```

#### 14.7.4.22 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 " ")

Classes defined

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

#### 14.7.4.23 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

#### 14.7.4.24 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}
```

#### 14.7.4.25 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

#### 14.7.4.26 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}
```

#### 14.7.4.27 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}
```

#### 14.7.4.28 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.

#### 14.7.4.29 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.

#### 14.7.4.30 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 " ")

#### Classes defined

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

#### 14.7.4.31 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}
```

#### 14.7.4.32 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}
```

#### 14.7.4.33 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}
```

#### 14.7.4.34 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}
```

#### 14.7.4.35 file\_from\_string\_mustache

Build a file from a mustache string

Compatible with nodes running *Rudder* 4.0 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}
```

#### 14.7.4.36 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}
```

#### 14.7.4.37 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.

#### 14.7.4.38 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.

#### 14.7.4.39 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 than 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 `{# . . . #}`.

Classes

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.

### 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 }}
```

### 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}
```

#### 14.7.4.40 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.

#### 14.7.4.41 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. As 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}
```

#### 14.7.4.42 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.

#### 14.7.4.43 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 system 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.

#### 14.7.4.44 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.

#### 14.7.4.45 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", "${sys.workdir}/ ←
    templates/ntp.conf.mustache")
# Define the 'ntp' varibale in the 'configuration' prefix from the json file
variable_dict_from_file("configuration", "ntp", "/tmp/ntp.json")
# Expand your template
file_from_template_type("${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}
```

#### 14.7.4.46 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}
```

#### 14.7.4.47 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

#### 14.7.4.48 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.

#### 14.7.4.49 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
```

#### 14.7.4.50 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:** Group of destination file

##### Classes defined

```
file_template_expand_${target_file}_kept, repaired, not_ok, reached
```

### 14.7.5 Group

#### 14.7.5.1 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
```

### 14.7.5.2 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}
```

## 14.7.6 Http

### 14.7.6.1 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}
```

### 14.7.6.2 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}
```

## 14.7.7 Log

### 14.7.7.1 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}
```

## 14.7.8 Logger

### 14.7.8.1 logger\_rudder

Logging output for *Rudder* reports. This interface is for compatibility with older generic methods and techniques, and is replaced by `log_rudder`.

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}
```

## 14.7.9 Package

### 14.7.9.1 package\_absent

Enforce the absence of a package

Compatible with nodes running *Rudder* 4.0 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", "pkg" or "default" for system default package manager (defaults to "default")

Classes defined

```
package_absent_${name}_{kept, repaired, not_ok, reached}
```

### 14.7.9.2 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}
```

### 14.7.9.3 package\_install

Install or update a package in its latest version available

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}
```

### 14.7.9.4 package\_install\_version

Install or update a package in a specific version

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}
```

### 14.7.9.5 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

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}
```

#### 14.7.9.6 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)

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}
```

#### 14.7.9.7 package\_present

## Enforce the presence of a package

Compatible with nodes running *Rudder* 4.0 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", "pkg" or "default" for system default package manager (defaults to "default")

Classes defined

```
package_present_${name}_{kept, repaired, not_ok, reached}
```

#### 14.7.9.8 package\_remove

Remove a package

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}
```

#### 14.7.9.9 package\_state

Enforce the state of a package

Compatible with nodes running *Rudder* 4.0 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.

#### 14.7.9.10 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.

#### 14.7.9.11 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).

#### 14.7.9.12 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.

*pkg*

This package provider uses FreeBSD's *pkg* to manage packages on the system.

#### 14.7.9.13 Examples

```
# To install postgresql in version 9.1 for x86_64 architecture
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", "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}
```

#### 14.7.9.14 package\_state\_options

Enforce the state of a package with options

Compatible with nodes running *Rudder* 4.0 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", "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}
```

#### 14.7.9.15 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}
```

#### 14.7.9.16 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}
```

### 14.7.10 Permissions

#### 14.7.10.1 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}
```

#### 14.7.10.2 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}
```

### 14.7.10.3 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}
```

### 14.7.10.4 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}
```

### 14.7.10.5 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}
```

## 14.7.11 Schedule

### 14.7.11.1 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}
```

### 14.7.11.2 schedule\_simple\_catchup

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}
```

### 14.7.11.3 schedule\_simple\_nodups

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}
```



#### 14.7.11.4 schedule\_simple\_stateless

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}
```

### 14.7.12 Service

#### 14.7.12.1 service\_action

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

Compatible with nodes running *Rudder* 3.1 or higher.

Parameters

- **service\_name**: Service
- **action**: Action to trigger on the service (start, stop, restart, reload, ...)

Classes defined

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

#### 14.7.12.2 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}
```

#### 14.7.12.3 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}
```

#### 14.7.12.4 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}
```

#### 14.7.12.5 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}
```

#### 14.7.12.6 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}
```

#### 14.7.12.7 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}
```

#### 14.7.12.8 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}
```

#### 14.7.12.9 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}
```

#### 14.7.12.10 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}
```

#### 14.7.12.11 service\_reload

Reload a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

*Parameters*

- **service\_name**: Service

Classes defined

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

#### 14.7.12.12 service\_restart

Restart a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

*Parameters*

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

Classes defined

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

#### 14.7.12.13 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.

*Parameters*

- **service\_name**: Service
- **trigger\_class**: class(es) which will trigger the restart of Service "(package\_service\_installed|service\_conf\_changed)" by example

Classes defined

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

#### 14.7.12.14 service\_start

Start a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

*Parameters*

- **service\_name**: Service

Classes defined

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

#### 14.7.12.15 service\_stop

Stop a service using the appropriate method

Compatible with nodes running *Rudder* 3.1 or higher.

*Parameters*

- **service\_name**: Service

Classes defined

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

### 14.7.13 User

#### 14.7.13.1 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}
```

#### 14.7.13.2 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}
```

## 14.7.14 Variable

### 14.7.14.1 variable\_dict

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

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}
```

### 14.7.14.2 variable\_dict\_from\_file

Define a variable that contains key,value pairs (a dictionary) 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}
```

### 14.7.14.3 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}
```

### 14.7.14.4 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}
```

#### 14.7.14.5 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}
```

#### 14.7.14.6 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}
```

## 14.8 Man pages

### 14.8.1 rudder(8)

#### 14.8.1.1 NAME

rudder - execute commands to control the *Rudder* configuration management tool.



### 14.8.1.2 SYNOPSIS

**rudder** *component* [-h] [-il-vl-d] *command*

**rudder** *component* help

### 14.8.1.3 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.

### 14.8.1.4 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.

### 14.8.1.5 COMMANDS

The commands below are listed by component.

#### 14.8.1.6 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

**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/rudder-agent health -n

**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.

**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

**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.

---

### 14.8.1.7 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 the 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)

### 14.8.1.8 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

#### 14.8.1.9 AUTHOR

Normation SAS ([contact@normation.com](mailto:contact@normation.com))

#### 14.8.1.10 RESOURCES

Main web site: <https://rudder-project.org/>

Sources of the rudder command-line: <https://github.com/Normation/rudder-agent/>

#### 14.8.1.11 COPYING

Copyright (C) 2014-2015 Normation SAS.

## 14.9 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

### 14.9.1 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
```

#### 14.9.1.1 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*.

#### 14.9.1.2 Version number formatting

The version number follows a formatting "a la *Debian*" as described here: <http://www.debian.org/doc/debian-policy/ch-controlfields.html#f-Version>, (without the *debian\_revision* version)

### 14.9.2 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">
```

### 14.9.3 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>
</TML>
<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 -->
<SAME SIZE AS>VariableName</SAME SIZE AS>           <!-- 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>

```

#### 14.9.3.1 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 repeatable 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*)

**14.9.3.2 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 all 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; define 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 \\

**14.9.3.3 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 formatted 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)

**14.9.3.4 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 techniques 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")

## 14.9.4 Examples

### 14.9.4.1 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>
```

### 14.9.4.2 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>
```

### 14.9.4.3 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.

## ▼ Section: Password

Enter password + hash
Enter pre-hashed value

Use clear text password
Enter script to set passwords

New password

Password for this account - Optional

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 metadata 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> constraint 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>
```

**14.9.5 Known limitations**

There are several known limitations at the moment, that are acknowledged, and will be solved in a "not too distant" future:

#### 14.9.5.1 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.

#### 14.9.5.2 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.

#### 14.9.5.3 Can't have several sections that are components with keys in multivalued Techniques.

It is a side effect of the previous limitation.

## 14.10 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*

### 14.10.1 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

### 14.10.2 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
- *DirectiveId* : 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
- **NodeId** : the id of the node
- **HumanReadableMessage** : a message than a Human can understand

#### 14.10.2.1 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*

## 14.11 Syntax of the Techniques

### 14.11.1 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).

### 14.11.2 Variable replacement

Note : *Rudder* use a StringTemplate grammar slightly different from the default one. Rather than using "\$" as a variable identifier, the *Techniques* use "&" to avoid collision with the *CFEngine* variables

#### 14.11.2.1 Single-valued variable replacement

```
&UUID&
```

- Will be replaced by the value of the variable **UUID**

Name	Type	Mode	Max number
Details	log_trace	log	any
infinity	Should be used for advanced debugging 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	audit_na	result
	Component to Error. Should be used when something was		

### 14.11.2.2 Replacement of variable with one or more values

```
&DNS_RESOLVERS: { "&it&" };separator=", "&
```

- Will be replaced 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

### 14.11.2.3 Replacement 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";
```

### 14.11.2.4 Conditionnal writing of a section

```
&if (NOVA) &
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://theantlruguy.atlassian.net/wiki/display/ST/ST+condensed+--+Templates+and+expressions>

## 14.12 Best Practices for Techniques

### 14.12.1 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_`

### 14.12.2 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

### 14.12.3 Writing convention

#### 14.12.3.1 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  $\Rightarrow$  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

#### 14.12.3.2 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.

#### 14.12.4 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*

#### 14.12.5 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).

#### 14.12.6 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.
-

## Chapter 15

# Appendix: Glossary

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

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

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

**cf-serverd** This *CFEngine* Community daemon is listening on the network on *Rudder Root* and Relay servers, serving policies and files to *Rudder Nodes*.

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

This version used to be called *Nova* before.

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

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

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

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

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

**Port 443, TCP, for nodes** WebDAV/HTTPS communication port, used to send inventory and fetch the id of the *Rudder Server*.

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

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

**Port 5308, TCP** *CFEngine Enterprise* communication port, which is required to manage *Windows* nodes.

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

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

**Port 80, TCP, for nodes** WebDAV/HTTP communication port, 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 or 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

Copyright © 2011-2016 *Normation SAS*

*Rudder* User Documentation by *Normation SAS* is licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#).

Permissions beyond the scope of this license may be available at *Normation SAS*.

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.