CFEngine

CFEngine Nova Technical Supplement

CFEngine Enterprise Documentation
for version 2.0

CFEngine

Table of Contents

1 Introduction to CFEngine Nova

CFEngine Nova is a commercially licensed version of the core CFEngine software[1] with enterprise library extensions. All of the documentation for CFEngine 3 applies to CFEngine Nova. This document is a supplement describing features particular to CFEngine Nova.

The aim of CFEngine Nova is to offer a knowledge-enhanced framework for configuration management that goes beyond mere technical configuration to support the needs of businesses. Features include compliance management, reporting and business integration, and tools for handling necessary complexity. CFEngine Nova has features to support Cloud Computing for public and private clouds, as well as greater integration facilities with database resources.

You should use this brief guide to CFEngine Nova's features in concert with the CFEngine Concept Guide and CFEngine Reference Manual, available from the CFEngine website, or in the '`docs`' directory of your Nova hub installation.

1.1 What are enterprise versions of CFEngine?

CFEngine Enterprise versions contain features designed to both extend and simplify the use of CFEngine in enterprise scenarios. This includes quick setup, business alignment features and improved self-documentation, ultimately providing a single framework for self-healing, hands-free automation with integrated knowledge management. Each extended feature has been carefully designed to meet a specific need in the server life-cycle, replacing cumbersome or insecure technologies currently available in datacentre products. CFEngine Nova adds the following capabilities:

- Simple bootstrap and installation.
- Scalable management of complex, federated environments.
- Lightweight reporting engine.
- Distributed orchestration of tasks.
- Virtualization control (using libvirt).
- Simplified policy writing with Content-Driven Policies.
- Integrated Knowledge Management.
- Bundle managed services.
- Provides a simple window onto IT Operations for Business.

Of course, as a Nova user, you have access to all of the features of the CFEngine Community Edition, and access to its on-line and community resources.

1.2 What's new in Nova 2.0?

CFEngine Nova 2.0 is a major upgrade of CFEngine's commercial software, including the latest community core and many improvements to the language interface. In addition, some of the major features in Nova 2.0 include:

- Fast, lightweight report collection to a hub.
- A browsable Web interface allows you to see policy and state side by side.

[1] Major version 3

- CMDB/SKMS capabilities in the Web interface.
- Embedded compliance reporting engine.
- Searchable reporting and analytics.
- A local customizable version of the Copernicus Knowledge Map is now fully integrated providing searchable on-line documentation, integrated with the policy browser.
- Virtualization (Libvirt model) support.
- Simplified management through Content-Driven Policies.
- Unique-ID support for mobile and dynamic IP support.
- FIPS validated encryption mode.

In addition to these features, Nova 2.0 is built around our extensive bug-reporting and self-diagnostic test suite that has allowed us to improve the reliability and functionality of the software for the most demanding environments.

1.3 About the CFEngine architecture

1.3.1 CFEngine is agent based software

CFEngine is agent based software. It resides on and runs processes on each individual computer under its management. That means you do not need to grant any security credentials for login to CFEngine. Instead, for normal operation, CFEngine runs in privileged 'root' or 'Administrator' mode to get access to system resources and makes these available safely to authorized inquiries.

A CFEngine installation is thus required on every machine you want to manage: client and server, desktop or blade. Typically, you will single out one machine to be a *policy server* or *hub*. In very large networks of many thousands of machines, you might need several policy servers, i.e. several hubs.

1.3.2 Single point of coordination

The default CFEngine Nova architecture uses a single hub or policy server to publish changes of policy and to aggregate knowledge about the environment, but you can set up as many as you like to manage different parts of your organization independently. The CFEngine technology is not centralized by nature. Most users choose to centralize updating of policy and report aggregation for convenience however.

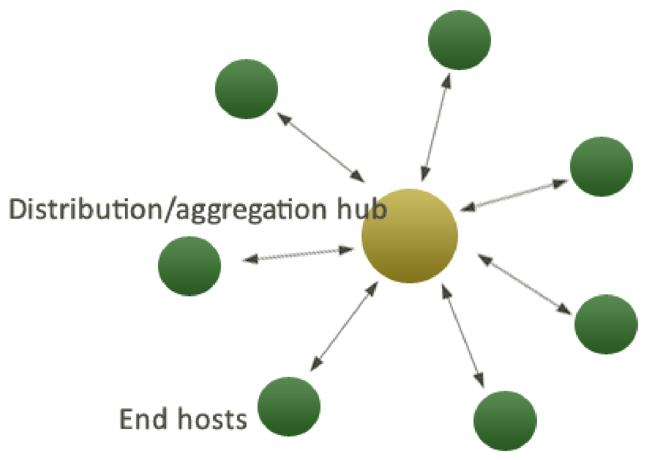


Figure: A policy server or 'hub' is implemented in CFEngine Nova as a simple solution that will scale for most sites 'out of the box'.

If you operate CFEngine Nova in its default mode, the hub acts as a server from which every other client machine can access policy updates. It also acts as a collector, aggregating summary information from each machine and weaving it into a knowledge map about the datacentre.

For a single hub configuration, the figure below shows a normal process approach to managing policy. Policy is edited and developed at a Policy Definition Point, outside of normal production environment. This can be done using the specialized editor embedded in CFEngine Nova, or it can be done using any text editor of your choice.

Edits are made in conjunction with a version control repository[2], which should be used to document the *reasons* for changes to policy[3]. When a change has been tested and approved, it will be copied manually to the policy dispatch point on one or more distribution servers. All other machines will then download policy updates from that single location according to their own schedule.

[2] CFEngine supports integration with Subversion through its Mission Portal, but any versioning system can of course be used.

[3] CFEngine and version control will document *what* the changes are, but what is usually missing from user documentation is an explanation of the reasoning behind the change. This is most valuable when trying to diagnose and debug changes later.

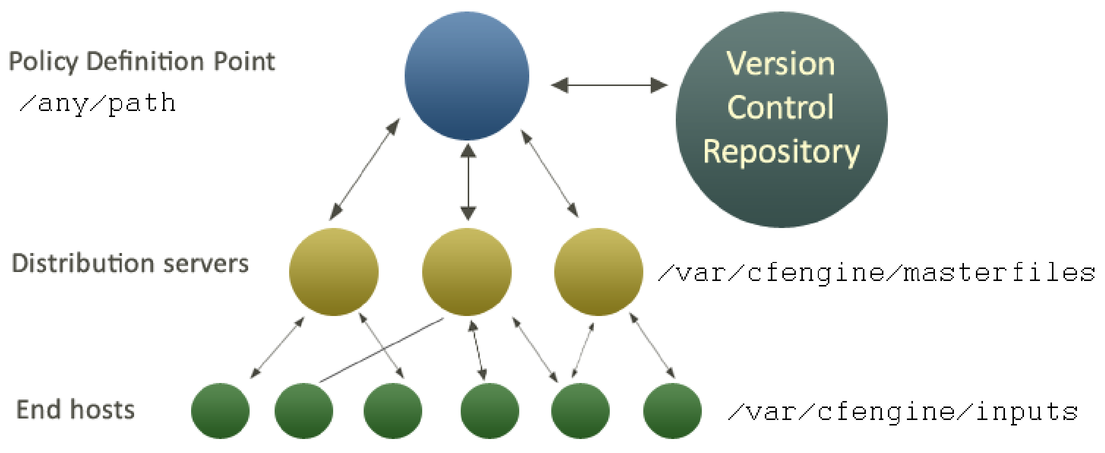


Figure: Policy coordinated from a central root location
is implemented in a distributed manner at every leaf node.

1.3.3 Requirements and scalability of CFEngine Nova

The default architecture and configuration skeleton is expected to scale to a few thousand hosts with a dedicated policy hub. Your hub machine should have at least 2 GB of memory and a modern processor. You will need about 2 MB of disk storage for every machine under CFEngine's management.

CFEngine will scale to very large systems, with tens of thousands of machines. This is only possible because of its decentralized agent-based operation. CFEngine encourages the federation of policy in complex environments, so that local domain experts manage what they know and baseline policies can be handled generally for all. We also encourage a practice of small adjustments, as opposed to large risky redesigns of operational policy.

CFEngine's recommended practice is to encourage small incremental changes to policy, not to save up changes into 'big project roll-outs'. This strategy lowers the risk of error and offers improved scalability.

CFEngine will operate autonomously (hands-free) in a network, under your guidance. If your site is large (thousands of servers) you should spend some time discussing with CFEngine experts how to tune this description to your environment as *scale* requires you to have more infrastructure, and a potentially more complicated configuration. The essence of any CFEngine deployment is the same.

1.3.4 Phases of the server life-cycle

The four phases in managing systems, summarized in the Build, Deploy, Manage, Audit (BDMA) framework originated with a model of system management based on transactional changes ('roll-out'). CFEngine's conception of management is some different, as transaction processing is not a good model for highly distributed systems, but we can use this template to see how CFEngine works differently.

Build CFEngine is often used together with other tools to build new machines, whether virtual or real. You integrate it into the 'kickstart' or 'jumpstart' process, using PXE network booting, etc, by including the CFEngine software in the initial build and executing CFEngine once at the end of the installation.

To complete the installation of your system, you create a set of promises about the system state, or policy-template of proposed promises. When systems keep these promises, they will function seamlessly as planned.

Deploy Deploying software or changes really means implementing the policy that has been decided. In CFEngine you simply publish your proposed policy and the machines can adjust accordingly. Each installed agent is capable of implementing policies and maintaining them over time without further assistance.

Manage Once a decision is made, unplanned events will occur. Such incidents usually set off alarms and humans rush to make new transactions to repair them. In CFEngine, the autonomous agent manages the system, and you only have to deal with rare events that cannot be dealt with automatically.

Audit

Promises are assured by design in CFEngine and maintained automatically, so the main worry when using CFEngine is whether or not you have coded conflicting intentions. The compliance of machines with the policy template is easily measured by CFEngine, using its model-oriented approach. This can be viewed from the web-browsable interface.

1.4 Nova Commercial Enhancements

The enhancement provide by CFEngine Nova fall into a number categories:

- Business transparency and reporting enhancements, providing insight into IT operations for both Business and IT. CFEngine Nova connects the dots between high level business goals and low level configuration, supporting Knowledge Management for ITIL processes.
- Productivity enhancements – easier getting started and making changes.
- Continuity and repair related enhancements.
- Full integrated Knowledge Management.
- Special support for operating system features and virtualization, such as Solaris zones and the libvirt interface.
- Native support for the Windows platform, with access to the configuration of the Windows registry, processes and file access control lists.

1.4.1 Productivity enhancements

Features that are designed to make it easier to work with CFEngine on a day to day basis.

- Simple automatic installation (one command per system)
- A template library of managed services for common platforms.
- Auto-analyzing system knowledge console
- Policy mining and semantic knowledge representation
- Auto-classification of systems
- Performance and Service Level reports
- On- and off-line syntax look-up.
- Integration with LDAP directory services.

The most important productivity tool is the CFEngine Mission Portal, which contains an interface for browsing and editing policy, and understanding it in the context of reports and a Knowledge Map of the system.

A library of managed services for common Linux operating systems is provided as part of the knowledge base. This makes setting up specialized service nodes a simple process of matching hosts to services from a required list.

The ability to look up syntax on the fly, from the web or the command line, can be save experienced users considerable time. Using the tools for database interaction, the process of setting up a CFEngine knowledge base at your organization is fully automated and you only have to think about local customizations you want to make.

LDAP query functions have been added for integration with LDAP services or Active Directory information, allowing a single point of definition for system lists and identity management.

1.4.2 Functional enhancements

- Database verification and editing including SQL (Mysql,Postgres)
- MS Registry management and repair
- File-system ACL security management (Linux, Windows, Solaris)
- Extended monitoring probes and system classification capabilities.

These include the ability to interact with and repair popular SQL databases (currently MySQL and PostgreSQL), as well as embedded system databases such as the Microsoft System Registry, defining, validating, scanning or repairing their tabular structure and the data within them.

- Access remote CFEngine variables (like a simple directory service)
- LDAP directory service integration
- Role based access control on policy execution by class.

Access Control List (ACL) support for Linux is now added for pinpoint accuracy in file permission security. CFEngine Nova's ACL support includes a completely new generic CFEngine model for ACLs that will translate across multiple platforms so that users can as closely as possible translate identical requirements across multiple platforms with different implementations. Native ACLs are also supported.

Fault tolerant features have been added in Nova to functions for retrieval of data from network connections. It is risky to rely of data from a network when configuring hosts. If the network connection could fail, erroneous data might be written to a local configuration. Nova adds local caching to network results and works opportunistically to provide the latest known values; however, look-up functions, such as LDAP retrieval of `remotescalar` data will not fail.

1.4.3 Reporting enhancements

Nova can provide a wider range of system reports about information collected by CFEngine on performance, security and state. This includes the ability to perform custom system discovery, and log the data into a variety of special reports.

Discovery and measurement promises are made through CFEngine's lightweight custom monitoring capabilities.

- HTML, XML, CSV report generation

- Service and performance level reports
- Reliability reports
- Deep history analysis and visualization, through knowledge base.
- Policy dependency and impact analysis reporting

Information can be extracted in HTML, XML and text formats (e.g. CSV) for easy integration with other presentation tools, or for direct viewing through the knowledge console.

As part of policy-writing, CFEngine Nova allows you to track dependencies on policy items. The knowledge agent can take this information, analyze it and present it as part of an overview of the system (see the chapter on Knowledge Management for more information). This enables virtual impact analysees to be inspected.

1.4.4 Documentation enhancements

Additional documentation is provided for Nova users through program built-in help, additional manuals and integration of syntax information into the CFEngine Knowledge Console.

An annoying aspect of any software is the need to browse through a manual to find quick answers to questions. Ideally one would only look at manuals during a learning phase; thereafter we want to see examples and summaries of what we basically already know but cannot keep in our heads.

System administrators often prefer to work in the command line and find the need to go to a manual a distraction. Indeed, if the network is non-functional or we are working off-line, instant command line help is a great bonus. Other users prefer the point and click of a familiar web interface. CFEngine Nova provides both these options to users to provide quick answers.

Of the two, a web interface is clearly the most flexible. Far more information can be browsed on the web than is practical with a simple text interface. However, of the two, the command line interface is by far the fastest way to get answers to simple questions of syntax. CFEngine's knowledge agent knows CFEngine's syntax tree and can summarize it at the keyword level.

```
atlas$ ./cf-know --syntax link_from
Constraint link_from (of promise type files) has possible values:

   link_from  ~ defined in a separate body, with elements

     source                ~ ()
     link_type             ~ (symlink,hardlink,relative,absolute,none)
     copy_patterns         ~ ()
     when_no_source        ~ (force,delete,nop)
     link_children         ~ (true,false,yes,no,on,off)
     when_linking_children ~ (override_file,if_no_such_file)

Description: A set of patterns that should be copied and synchronized instead of linked
```

1.4.5 Knowledge Management in Nova

The future of datacentre management lies in a more complete model of knowledge and transparency for processes and resources. CFEngine Nova provides a platform for this through:

- Auto-generated central knowledge console.
- Access to CFEngine's company support knowledge base.

- Tie-in knowledge base with local documentation.
- Dependency mapping between promises in CFEngine, with integration into the semantic
- Trace high level policies to low level policies

2 Installing CFEngine Nova

CFEngine Nova is designed to be simple to install in its default configuration. The installation process has three phases:

- Unpacking the software.
- Obtaining a license.
- Adapting policy.

2.1 Operating system support

CFEngine can be made to run on most operating systems. For efficiency CFEngine only supports a number of recent popular operating systems, which should be up to date with patches. If we don't have packages for your particular operating systems we can usually make packages by special arrangement.

The hub (or policy server) is only available for derivatives of the top GNU/Linux distributions (Debian, Red Hat, SuSE), as these make available software that the hub relies on for data storage and processing. Operating system choices for the hub are:

```
Debian 5,6
Ubuntu 8,9,10
RedHat 4,5
SuSE   11
```

No special software is required on other machines in your network. CFEngine bundles all dependencies in the Nova package.

Nova provides a version of CFEngine running natively on Windows, with support for registry management, Windows services and file security, See Chapter 12 [Windows-specific features in Nova], page 87. Support for Solaris zones has been added through automated zone detection and process model adaptation.

2.2 Installing the software on a new host

Nova is provided in two packages. The main software package (for each operating system) must be installed on every host. The second package is only installed on the *hub* or *policy server*. You should install the hub first.

On a new host, installation follows the following procedure. References to package managers assume that additional packages might need to be installed of the policy server, e.g. the Apache Web Server.

1. Verify that the machine's network connection is working and that it can resolve names. On the hub verify that package managers `yum`, `zypper` or `apt-get` are working.
2. Copy the Nova packages to the system. On the hub or policy server:

   ```
   CFEngine-nova-2.xxx.[rpm|deb]
   CFEngine-nova-expansion-2.xxx.[rpm|deb]
   ```

   On all other machines

   ```
   CFEngine-nova-2.xxx.[rpm|deb]
   ```

3. Unpack the software:

   Red Hat family

   ```
   host# rpm -ihv packages
   ```

Debian family

```
host# dpkg --install  packages
```

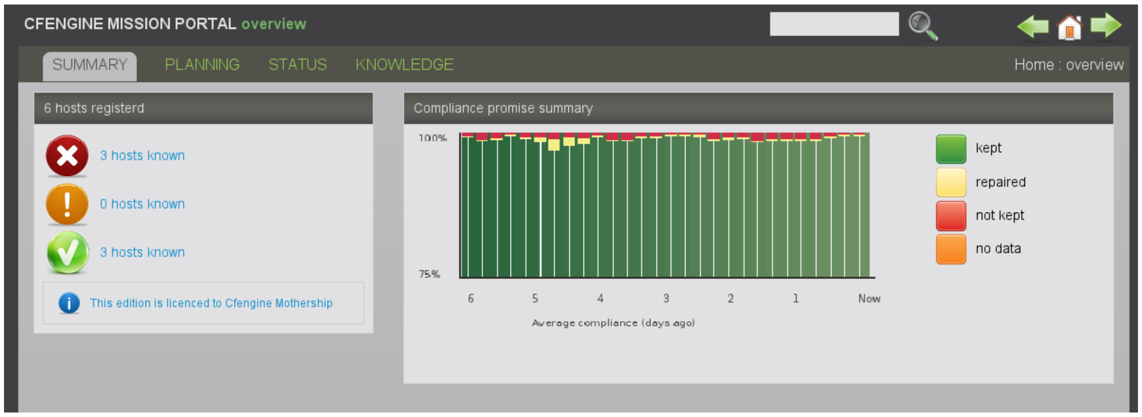
4. A public key has now been created in '`/var/cfengine/ppkeys/localhost.pub`' on the policy hub as part of the package installation. You should email this public key to CFEngine Support, as an attachment, to obtain a license file '`license.dat`'. Save the returned license file to '`/var/cfengine/masterfiles/license.dat`' on the hub before continuing.
5. The remaining steps apply to all hosts, but you should install the hub or policy server first. Find the name or IP address of the hub (policy server), here we assume '`123.456.789.123`' is the address.

   ```
   /var/cfengine/bin/cf-agent --bootstrap --policy-server 123.456.789.123
   ```

   Use the same command on all hosts, i.e. do not bootstrap the policy server with a localhost address.
6. The software should now be running.
7. To complete to licensing, you should make a promise to accept the license terms by editing the '`/var/cfengine/masterfiles/promises.cf`' on the policy server, and editing the '`host_licenses_paid => "100";`' line in '`body common control`' to reflect the correct number of licenses that you have subscribed to.

How to assess success in this procedure:

1. Look at the process list on the systems '`ps waux | grep cf`'or '`ps -ef | grep cf`'. You should be able to see `cf-execd` running, and eventually other processes from the CFEngine suite like `cf-monitord cf-serverd`.
2. Look for files in '`/var/cfengine/inputs`' (Unix) or '`X:\Program Files\CFEngine\inputs`' (Windows). The license file will be copied out from the policy server to the clients as part of the normal distribution of policy. Each Unix machine should get a copy of the '`license.dat`' file in '`/var/cfengine/inputs`' (Unix) or '`X:\Program Files\CFEngine\inputs`' (Windows).
3. The file '`/var/cfengine/promise_knowledge.cf`' should have been created, and should contain data.
4. Finally try connecting to the web server at port 80 on the hub / policy host. You should see a summary page like the one shown in the figure below.



The opening page of the CFEngine Nova Mission Portal.

2.3 Upgrading Nova

When upgrading the software from an earlier version, you should upgrade the policy server (hub) machine first. Any other hosts in your network that act as servers for encrypted copy operations would preferably be upgraded first. This is because a stronger form of encryption-hash is used in newer versions, which the older servers cannot understand.

CFEngine packages its software in operating system compatible package formats (RPM, PKG, MSI, etc). New packages are made available on the software.CFEngine.com website; these can be downloaded and installed in the standard way.

1. Go to the `software.CFEngine.com` website
2. Select the version of the software you require
3. Download the particular package for your operating systems to a correspondingly named subdirectory on your policy server under '`/var/cfengine/master_software_updates`', or simply mirror the entire file tree on your policy server under '`/var/cfengine/master_software_updates`'.

- You do not need to delete the contents of the work directory, e.g. '`/var/cfengine`' to upgrade. An upgrade only involves changing the binary code base.
- Users are encouraged to take advantage of changes to the Nova base policy, e.g. the '`update.cf`' policy. We have reduced the number of files distributed and managed by CFEngine in 2.0.0. To gain the full benefits of the software, you should use the latest versions of these files:

```
cdp_inputs/  CFEngine.cf         failsafe.cf     knowledge.cf  promises.cf   update.cf
cdp_lib/     CFEngine_stdlib.cf  file_change.cf  OrionCloud/
```

  Note that some log files have new names, and that the audit database is no longer used by default. If you do not upgrade the files above, make sure you rotate the log files and perform garbage collection as detailed in these files.
- You can upgrade systems manually by copying the relevant package to the local host and running the appropriate package update command (see summary below).
- To upgrade all of your systems automatically, you only need to copy a new package release to a specially designated location on your policy server, and CFEngine will do the rest. Each CFEngine client will detect the presence of the update, download it, install and restart CFEngine services locally. Thus you can deploy software using a single point of management.

2.3.1 Troubleshooting upgrade

You might experience some authentication errors during secure copy operations during upgrade from older versions of Nova. These will self-repair once the servers have been upgraded.

To avoid these altogether, you can temporarily disable copy verification, i.e. set

```
verify => "false";
```

in your `copy_from` bodies, should you wish. For standard installations, this setting is typically part of the definition of '`secure_cp`' defined in the '`CFEngine_stdlib.cf`' file.

2.4 What is the default configuration?

Following the above procedure, you should have a fully functional CFEngine on all clients. However, in the default configuration, CFEngine does nothing other look after itself, and look for possible policy updates from the master file policy distribution point on the policy server. The policy server is configured to collect data from non-policy server machines and generate reports that are integrated into the knowledge base.

To alter policies, you need to change the files on the policy hub, in the directory '/var/cfengine/masterfiles'. To begin with most of the policy examples are commented out in these files:

```
active_directory.cf   failsafe.cf           spp.cf                spp_registry.cf
win_emergency.cf      app_baseline.cf       file_change.cf        scheduling.cf
spp_commands.cf       spp_services.cf       win_registry.cf       CFEngine.cf
file_security.cf      OrionCloud            software_local.cf     spp_file_changes.cf
system.cf             win_services.cf       garbage_collection.cf process_matching.cf
spp_file_diffs.cf     update.cf             CFEngine_stdlib.cf    knowledge.cf
promises.cf          spp_acls.cf           spp_inputs
```

To change this, you can go to the main file 'promises.cf', and include additional pre-made bundles of promises. You should always verify the contents of the bundles you include before activating and deploying to new machines.

```
bundle agent main
{
methods:

  any::

   "general" usebundle => def;

#   "jobs" usebundle => system_scheduling;
#   "security" usebundle => change_management;
#   "security" usebundle => security_files;
#   "windows boxes" usebundle => active_directory;

#  windows::
#   "windows boxes" usebundle => software_local;
#   "windows boxes" usebundle => app_baseline;
#   "windows boxes" usebundle => win_services;
#   "windows boxes" usebundle => win_registry;
#   "windows boxes" usebundle => win_emergency;

#  !windows::
#   "security"    usebundle => system_xinetd;
#   "maintenance" usebundle => garbage_collection;
}
```

3 Mission Portal

Knowledge management has become a new focus in IT management, and it is a core focus at CFEngine. Comprehending the growing complexity of IT operations is one of the main challenges in IT today. CFEngine Nova has a number of key features for knowledge management, including automated documentation, report generation and associative inference. The CFEngine Mission Portal is the centerpiece of user interaction with CFEngine Nova.

The structure of the Mission Portal interface follows the concept of a Topic Map. A Topic Map is an ISO standard for indexing electronic information as a *semantic web*. In a semantic web, you are presented with links to documents about your chosen topic, as usual; in addition you are offered *leads* and possible pathways to topics that are known to be related. These leads don't just point you to more documents, but explain *how* neighboring issues are related. The aim is to help the user learn from the experience of browsing, by conveying the meaning of the current topic in relation to other issues in the system. This is how *knowledge transfer* occurs.

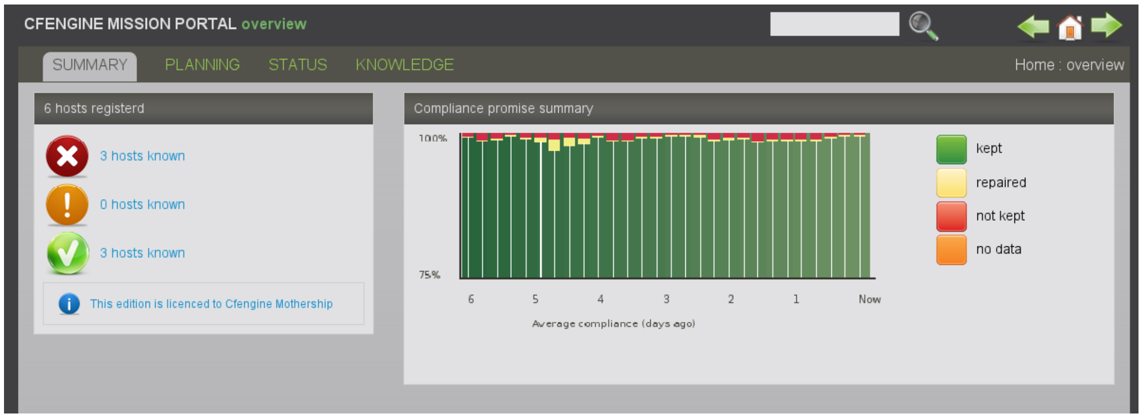


Figure: The mission portal

There are four main categories in the mission portal:

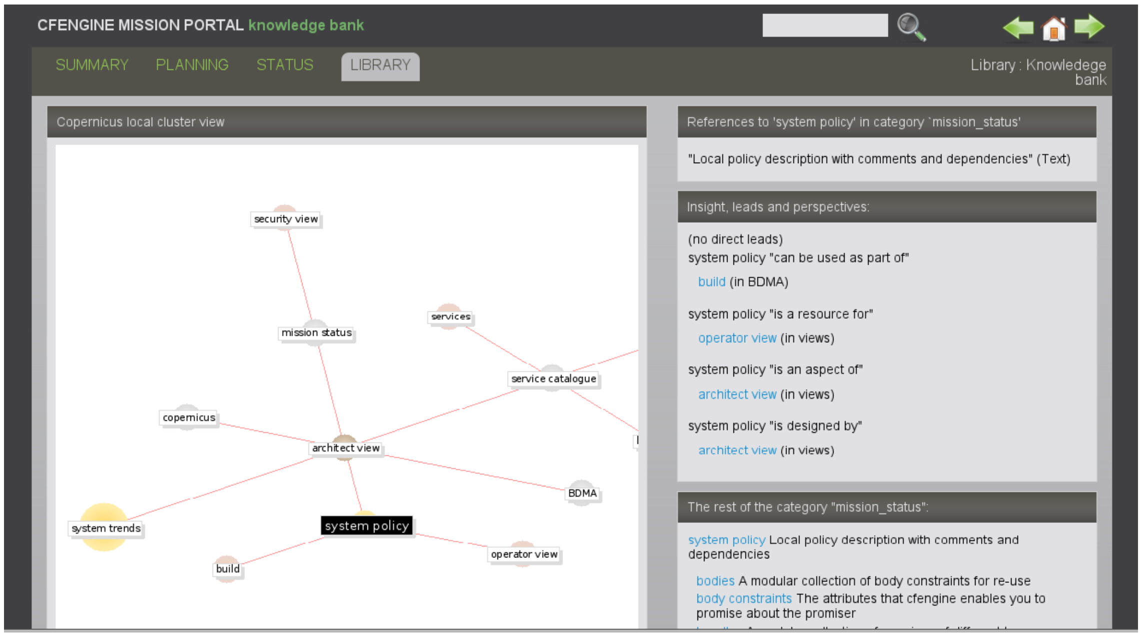
- Mission Summary - a top level overview of compliance status
- Mission Planning - a place to plan and make policy changes
- Mission Status - a place to see the current state of system repair
- Mission Library - a knowledge bank that connects information together

Each of these categories is a beginning from which you can refine your overview and search through information.

3.1 Mission Portal Views

The Mission Portal offers insight into three main areas:

- Operations and performance.
- Business and compliance.
- Organizational knowledge and library.



References to 'system policy' in category `mission_status'

"Local policy description with comments and dependencies" (Text)

Insight, leads and perspectives:

(no direct leads)
system policy "can be used as part of"
build (in BDMA)

system policy "is a resource for"
operator view (in views)

system policy "is an aspect of"
architect view (in views)

system policy "is designed by"
architect view (in views)

The rest of the category "mission_status":

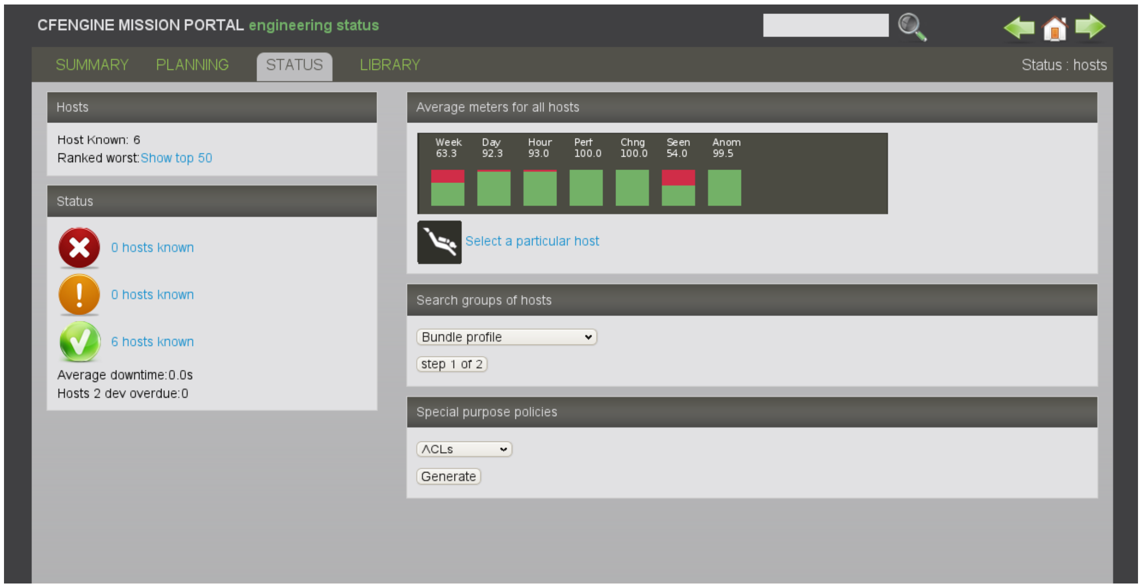
system policy Local policy description with comments and dependencies

bodies A modular collection of body constraints for re-use
body constraints The attributes that cfengine enables you to promise about the promiser

Figure: Topic Maps provide a variety of viewpoints on the information

Mission portal views link together a number of relevant resources for common system roles, like operator, security officer or system architect. This is an example of how the mission portal offers relationships that transcend simple categories. The same information can appear in many different contexts.

3.1.1 Mission status



Host Known: 6
Ranked worst: Show top 50

0 hosts known
0 hosts known
6 hosts known
Average downtime: 0.0s
Hosts 2 dev overdue: 0

Week	Day	Hour	Perf	Chng	Seen	Anom
63.3	92.3	93.0	100.0	100.0	54.0	99.5

Figure: The status of IT operations.

Mission status is a high level summary of how well the entire system is behaving. From this page you can dive down into details about managed hosts, by machine or by information type.

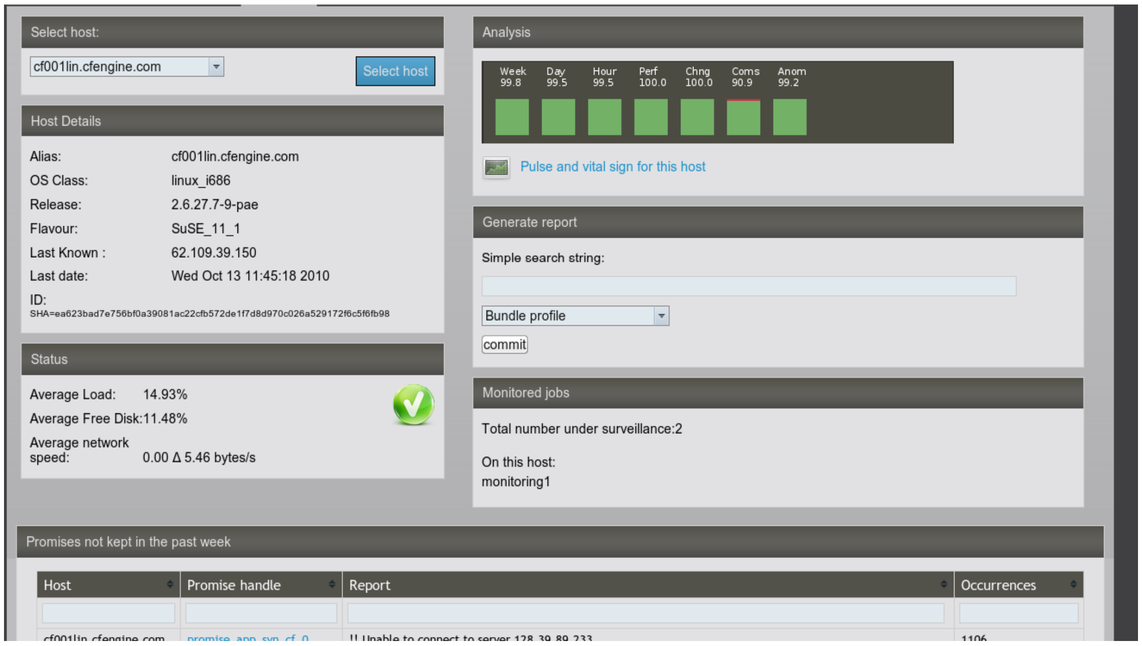
The row of bar meters shows the compliance of all registered hosts over the past week, the past day and the past hour. It also summarizes performance and anomalous behavior in a simple red, yellow green scale.

The hosts themselves are classified into red, yellow and green using a combination of these considerations. Clicking on the colored links produces a list of the hosts in that category.

3.1.2 Host view

Clicking on a host name or address leads to a specialized host page, which has the following broad characteristics:

- Operating system and host type.
- Host name, address and unique identity.
- A summary meter for compliance, performance, etc.
- Detailed reports.



Host Details

Alias:	cf001lin.cfengine.com
OS Class:	linux_i686
Release:	2.6.27.7-9-pae
Flavour:	SuSE_11_1
Last Known :	62.109.39.150
Last date:	Wed Oct 13 11:45:18 2010
ID: SHA=ea623bad7e756bf0a39081ac22cfb572de1f7d8d970c026a529172f6c5f6fb98	

Status

Average Load: 14.93%
Average Free Disk:11.48%
Average network speed: 0.00 Δ 5.46 bytes/s

Simple search string:

Monitored jobs

Total number under surveillance:2

On this host:
monitoring1

Promises not kept in the past week

Host	Promise handle	Report	Occurrences
cf001lin.cfengine.com	promise_app_svn_cf_0	!! Unable to connect to server 128.39.89.233	1106

Figure: A page specific for each host.

3.1.3 Status level meters

IT operations are referred to as *the mission* in the knowledge base. The mission depends on key resources and can be viewed from a number of perspectives, shown on the opening page.

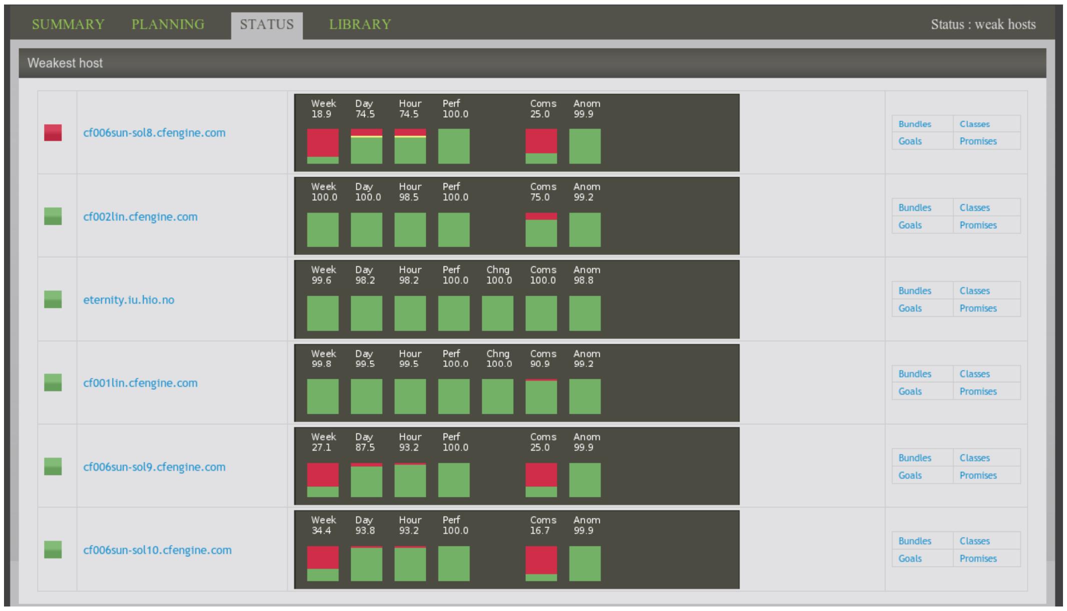
The bar meter shows the site-summary status of a number of key performance indicators. This is the highest level summary of mission status.

1. **Week**: The average level of promise-compliance over the whole past week.
2. **Day**: The average level of promise-compliance over the past day.
3. **Hour**: The average level of promise-compliance over the past hour.
4. **Perf**: The average performance status of the system, compared to the learned norm.
5. **Soft**: Software update status of the system.

6. **Lics**: Summary of licenses, i.e. number of active clients versus number of paid licenses promised.
7. **Anom**: Level of anomalous site-wide activity on the system.

3.1.4 Top 50 weakest hosts

One the status page, there is a link to the top 50 worst machines. Clicking on the bar meter graph takes you to the host directory, where a similar graph displays the status of the list of managed hosts.



	Host	Week	Day	Hour	Perf	Chng	Coms	Anom		
🟥	cf006sun-sol8.cfengine.com	18.9	74.5	74.5	100.0		25.0	99.9	Bundles / Goals	Classes / Promises
🟩	cf002lin.cfengine.com	100.0	100.0	98.5	100.0		75.0	99.2	Bundles / Goals	Classes / Promises
🟩	eternity.iu.hio.no	99.6	98.2	98.2	100.0	100.0	100.0	98.8	Bundles / Goals	Classes / Promises
🟩	cf001lin.cfengine.com	99.8	99.5	99.5	100.0	100.0	90.9	99.2	Bundles / Goals	Classes / Promises
🟩	cf006sun-sol9.cfengine.com	27.1	87.5	93.2	100.0		25.0	99.9	Bundles / Goals	Classes / Promises
🟩	cf006sun-sol10.cfengine.com	34.4	93.8	93.2	100.0		16.7	99.9	Bundles / Goals	Classes / Promises

Figure: The Top 50 weakest hosts list.

Clicking further on each host takes you do detailed performance data as measured by CFEngine's monitoring daemon.

We call this view pseudo-real time monitoring data, because there is always some delay between when data were measured and when they become available for you to view. Most tools simply hide this delay, but CFEngine tells you precisely when data were last updated, providing crucial information about system latencies.

3.1.5 Pulse and vital signs

For each host there is a 'pulse and vital signs' link. This shows an overview of monitoring data from each host and its current performance statistics. In order to see data in these graphs, each host in the CFEngine managed network must be running `cf-monitord` and `cf-serverd`. This is the default behavior for a Nova installation.

The main page of the pulse and vital signs shows up to 100 different measured values, for which some activity has been seen[1].

The graphs are arranged into three columns, representing three different time-scales. From the left:

[1] There is a known issue with the algorithm for determining whether or not there are actual data to be shown for a given metric. Noise and sporadic port connections might result in the Mission Portal showing strange looking graphs with blanks and holes. Alternative approaches to auto-detecting data are being investigated for future releases.

- Past four hours.
- Past weeks.
- Statistical complete history.

The past four hours graph is updated regularly and shows the current levels in 5 minute resolution. The weekly view shows data collected over many weeks, overlaid onto a ‘periodogram’, i.e. multiple weeks averaged over a single week view. This reveals the strongest trends of usage on the system.

The statistical history shows a spectral histogram, indicating the distribution of all measured values about the mean. This shows how much entropy or variation there is in the performance characteristic.

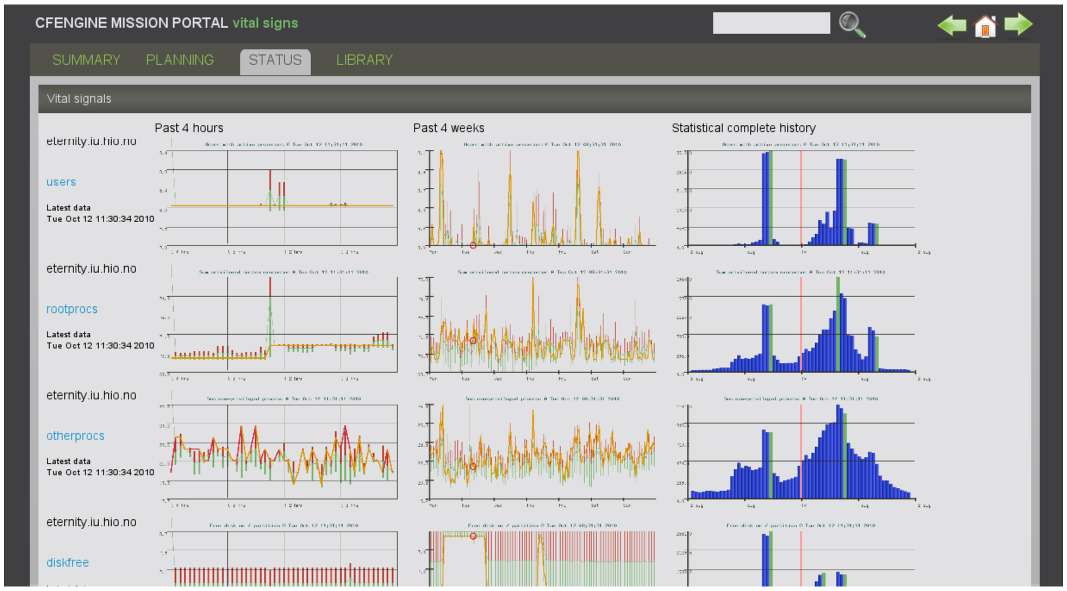
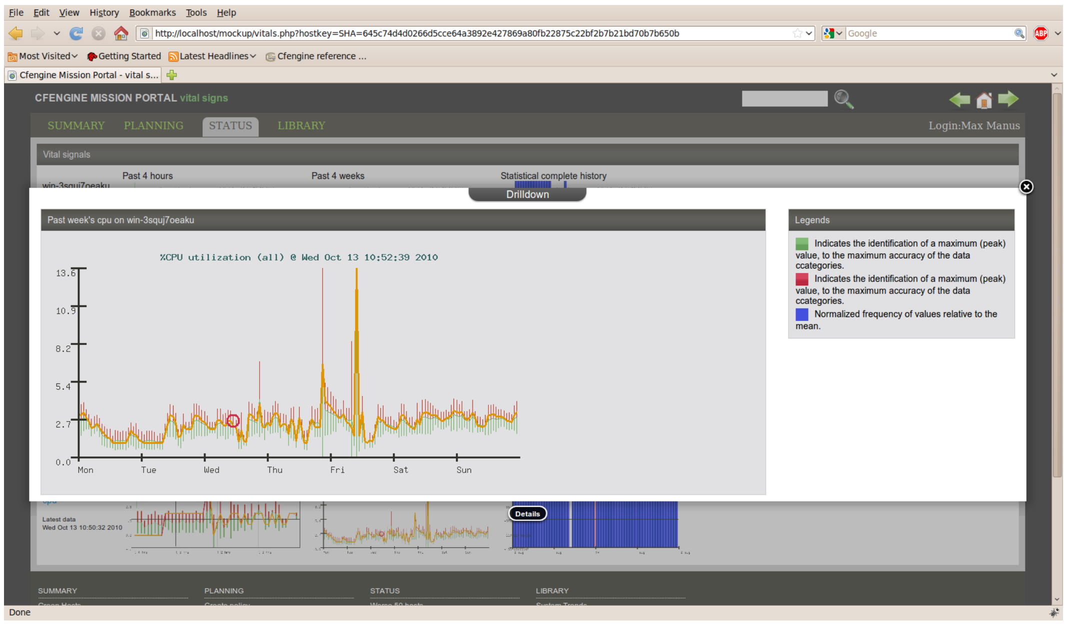


Figure: Overview of performance characteristics

3.1.6 Detail views

Clicking on the individual graphs in the vital signs provides a deeper analysis of each graph, with some basic statistics. These statistics are updated at different rates, depending on the time scales. ‘Past four hour’ graphs are updated every 5-10 minutes, while the others are updated every 6 hours. The numerical values are reanalyzed on a corresponding interval.


http://localhost/mockup/vitals.php?hostkey=SHA=645c74d4d0266d5cce64a3892e427869a80fb22875c22bf2b7b21bd70b7b650b
CFENGINE MISSION PORTAL vital signs
SUMMARY
PLANNING
STATUS
LIBRARY
Login:Max Manus
Vital signals
Past 4 hours
Past 4 weeks
Statistical complete history
Drilldown
Past week's cpu on win-3squj7oeaku
%CPU utilization (all) @ Wed Oct 13 10:52:39 2010
13.6
10.9
8.2
5.4
2.7
0.0
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Legends
Latest data Wed Oct 13 10:50:32 2010
Details
Done

Legends

🟩 Indicates the identification of a maximum (peak) value, to the maximum accuracy of the data ccategories.

🟥 Indicates the identification of a maximum (peak) value, to the maximum accuracy of the data ccategories.

🟦 Normalized frequency of values relative to the mean.

Figure: Details of the graph.

3.1.7 The policy editor

The CFEngine Nova Mission Portal provides an editor for working on CFEngine language. It does not support user management itself[2], however there is a tie-in for Subversion version control repositories.

Because there is no user management, each web-browser session will get its own working environment. Thus, there will be limited persistence, so remember to *commit your changes often*, especially before closing the browser.

For security reasons, there is no automatic link between the policy editor and the company policy in `masterfiles`. Subversion can be used to handle access control, and a manual review and copy step is required to deploy policy written in the editor.

The editor provides syntax high-lighting and look-up to make working with CFEngine's extensive language easier.

The main key commands in the editor Window are:

Autocompletion: Ctrl+Space
Shows a pop-up menu of possible items. This is context sensitive, e.g. it also works inside lists (e.g. `bsdflags`) to provide possible values.

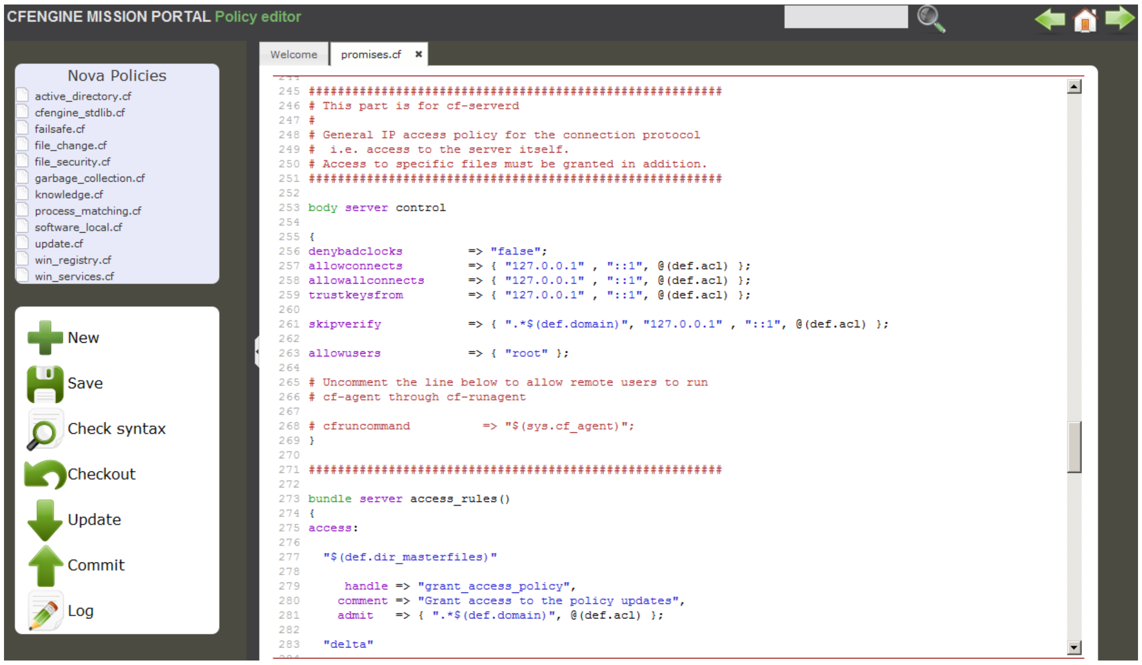
Undo: Ctrl+Z
In Safari, Ctrl-backspace may be used.

Redo: Ctrl+Y
Undo an undo operation, i.e. reverse the direction of transaction roll.

Indent: TAB
Format a file to a standard indentation.

Multiple documents appear as tabs along the top of the screen.

[2] A full multi-user version will be part of CFEngine Constellation



```
#########################################################
# This part is for cf-serverd
#
# General IP access policy for the connection protocol
#  i.e. access to the server itself.
# Access to specific files must be granted in addition.
#########################################################

body server control

{
denybadclocks         => "false";
allowconnects         => { "127.0.0.1" , "::1", @(def.acl) };
allowallconnects      => { "127.0.0.1" , "::1", @(def.acl) };
trustkeysfrom         => { "127.0.0.1" , "::1", @(def.acl) };

skipverify            => { ".*$(def.domain)", "127.0.0.1" , "::1", @(def.acl) };

allowusers            => { "root" };

# Uncomment the line below to allow remote users to run
# cf-agent through cf-runagent

# cfruncommand          => "$(sys.cf_agent)";
}

#########################################################

bundle server access_rules()
{
access:

  "$(def.dir_masterfiles)"

     handle => "grant_access_policy",
    comment => "Grant access to the policy updates",
    admit   => { ".*$(def.domain)", @(def.acl) };

  "delta"
```

Figure: The Policy Editor

The CFEngine Nova policy editor detects syntax errors and highlights these in red to avoid mistakes when editing. In addition, by using the `Check syntax` button, it is possible to pre-test the policy before committing changes to a repository. This will run `promises.cf` through the `cf-promises` parser.

The main menu on the left hand panel shows the main workflow items for policy editing. Clicking the arrow in the panel divider collapses the menu and gives full-screen editing.

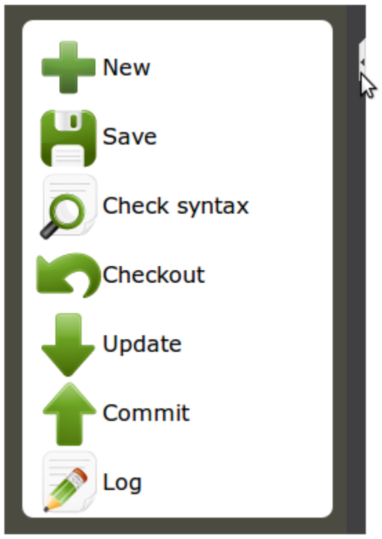


Figure: The policy editor main menu

3.1.8 Integration with subversion

The default architecture (page) proposes to work in close dialogue with a version repository. CFEngine recommends the use of such a repository and currently supports subversion.

Since the CFEngine Nova policy editor is a single user console, authentication is required for each interaction with the repository. Access control and authorization for policy are handled by subversion's authentication system so that edits made in the Mission Portal can be interleaved with edits from other sources. The subversion repository thus becomes the centerpiece of the distributed coordination, as is the intended function of version control systems.

When working with a subversion repository, you first do a "checkout". This allows you to select the repository you want to download and creates a working environment based on it. Note that the editor currently only views files in the checked out repository, so if you have a directory hierarchy, you need to check out multiple times. After doing a checkout, you can view the path to the current repository by clicking the link at the top.

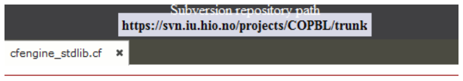
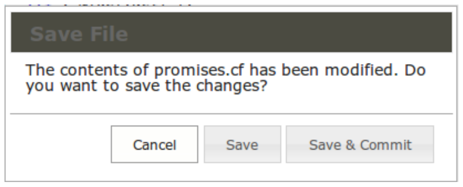

Subversion repository path
https://svn.iu.hio.no/projects/COPBL/trunk
cfengine_stdlib.cf

Figure: Path to the current repository.

The "update" command downloads the newest files from the already checked out subversion repository, while "commit" updates the repository with your working copy. You also have the ability to see the repository log of the 20 last changes. This includes changed files, users and log messages. For more detailed information on working with subversion, see http://svnbook.red-bean.com.

After editing you may or may not commit changes. When closing an edited file, you are also given the option to commit at once.



The contents of promises.cf has been modified. Do you want to save the changes?

Figure: Save file with repository commit

The menu choice to commit a current version commits all files in the configuration repository. Commits require a mandatory comment. CFEngine recommends you use this to explain why a change was made, or relate it to an incident number, etc. The details of *what* was changed are already documented by the version control logs.

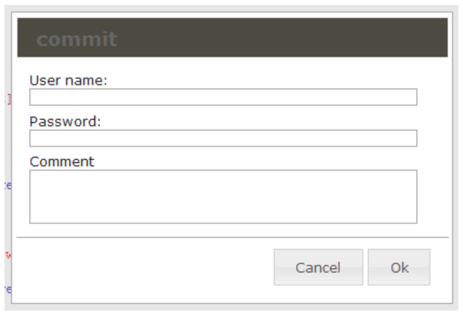


Figure: Commit dialogue

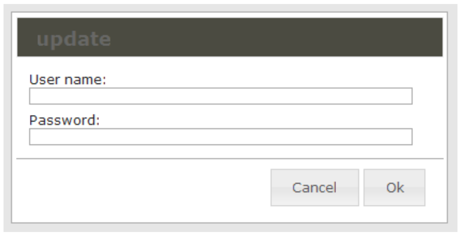


Figure: Updating the repository contents.

3.2 Reports in the Mission Portal

A significant capability of CFEngine Nova over previous versions of CFEngine is the existence of automated system reporting. CFEngine collects history, state and change data about computers and ties them together. The CFEngine strategy is to replace conventional CMDBs with a more scalable and flexible approach to information mining over the coming years. Commercial versions of CFEngine are designed to bring state of the art methods to the problem of information management for IT operations.

A report is a tabular summary of CFEngine's internal information, tailored to a particular purpose. Reports in CFEngine Nova are searchable, and describe attributes and qualities of managed hosts.

In CFEngine Nova 2 reports are summarized and collected automatically by the `cf-hub` program. This is a daemon which runs only on the system hub or policy server. Its sole function is to connect to all registered systems and build a searchable summary of its running state, in as efficient a manner as possible.

In addition to the automated hub, reports may be generated on each individual host using the `cf-report` program. This generates reports in a variety of formats, either for direct viewing (in HTML) or for import/export to other software systems, using XML or CSV output formats.

3.3 Should monitoring and configuration be separate?

The traditional view of IT operations is that configuration, monitoring and reporting a three different things that should not be joined. Traditionally, all three have been independent centralized processes. This view has emerged historically, but it has a major problem.

Monitoring as an independent activity is inherently non-scalable. When numbers of hosts grow beyond a few thousands, centralized monitoring schemes fail to manage the information. Tying configuration (and therefore repair) to monitoring at the host level is essential for the effective management of large and distributed data facilities. CFEngine foresaw this need in 1998, with its Computer Immunology initiative, and continues to develop this strategy.

CFEngine's approach is to focus on scalability. The commercial editions of CFEngine provide what meaningful information they can in a manner that can be scaled to tens of thousands of machines.

3.4 What is a CMDB?

A Configuration Management Database (CMDB), sometimes called a Change and Configuration Management Database (CCMDB), is a repository of information about hardware and its expected state. CMDBs are basically an outgrowth of *inventory databases*. They were embraced by many IT companies as a plausible solution to configuration management, in the absence of an alternative.

A CMDB records so-called *configuration items* (CI), which record technical, ownership and proposed state data. A CMDB does not generally record actual state – this is left for monitoring software.

The term CMDB originates from the IT Infrastructure Library (ITIL) framework, where it is given a prominent role in system planning and verification. The CMDB is by definition a centralized repository.

3.5 CMDB and CFEngine

CFEngine deviates from the classical view of a CMDB to modernize the concept for modern scaling requirements. Configuration management requires something more sophisticated than a database to describe system patterns. The CMDB applies a *brute force* approach to collection and searching of system data that does not scale well, and requires large and expensive resources to manage.

CFEngine works with a highly compressed description of system properties that is based on category classification rather than an exhaustive inventory of computers. Being distributed in operation, CFEngine can also enforce policy on state, without brute force imaging.

3.6 Decentralized data collection in CFEngine

CFEngine is designed to scale to vast numbers of machines. It does so because it is fundamentally decentralized. Data about system state are recorded *in situ* and are not transmitted over the network directly. CFEngine summarizes and compresses system information before making it available for central aggregation and analysis. This means that high resolution data are available where they count, processing is decentralized, and inexpensive summaries may be compared and mined for correlations.

CFEngine's distributed architecture means that no data are lost if the network fails temporarily. CFEngine uses the network *opportunistically*, but it is not reliant on it for real-time operations.

3.7 Reporting in CFEngine

If you have regular reporting needs, we recommend using our commercially supported version of CFEngine (CFEngine Nova or above), as you will save considerable time and resources in programming, and you will have access to the latest developments through the software subscription.

No promises made in CFEngine imply automatic aggregation of data to a central location. In commercial CFEngine versions, e.g. CFEngine Nova, an optimized aggregation of standardized reports is provided, but the ultimate decision to aggregate must be yours.

Monitoring and reporting capabilities in CFEngine depend on the software version include:

- **Community Edition:** Basic output to file or logs may be customized on a per-promise basis. Users can design their own log and report formats, but data processing and extraction from CFEngine's embedded databases must be scripted by the user.
- **Nova:** In addition to community features, Nova provides automated extraction of data from CFEngine's self-learning agents, and the generation of a standard set of reports in text, HTML or XML formats. Nova summarizes distributed data and provides simple compression and aggregation of these summaries. Finally summaries are tied into a knowledge map or semantic index for browsing by IT operations. At this level CFEngine exceeds other industry CMDB solutions for configuration.
- **Constellation:** In addition to Nova features, Constellation performs additional data extraction from the collected reports. It analyses correlations and provides reverse look up of system attributes

based on searchable expressions. At this level, CFEngine exceeds other industry CMDB solutions in both reporting and configuration.

3.8 Standard reports in CFEngine Nova

Available patches report
: Patches already installed on system if available.

Classes report
: User defined classes observed on the system – inventory data.

Compliance report
: Total summary of host compliance, all promises aggregated over time.

File_changes report
: Latest observed changes to system files with time discovered.

File_diffs report
: Latest observed differences to system files, in a simple diff format.

Hashes report
: File hash values measured (change detection).

Installed patches report
: Patches not yet installed, but published by vendor if available.

Installed software report
: Software already installed on system if available.

Last-seen report
: Time and frequency of communications with peers, host reliability.

Micro-audit report
: Generated by CFEngine self-auditing. This report is not aggregated.

Monitor summary report
: Pseudo-real-time measurement of time series data.

Performance report
: Time cost of verifying system promises.

Promise report
: Per-promise average compliance report over time.

Promises not kept report
: Promises that were recently un-kept.

Promises repaired report
: Promises that were recently kept by repairing system state.

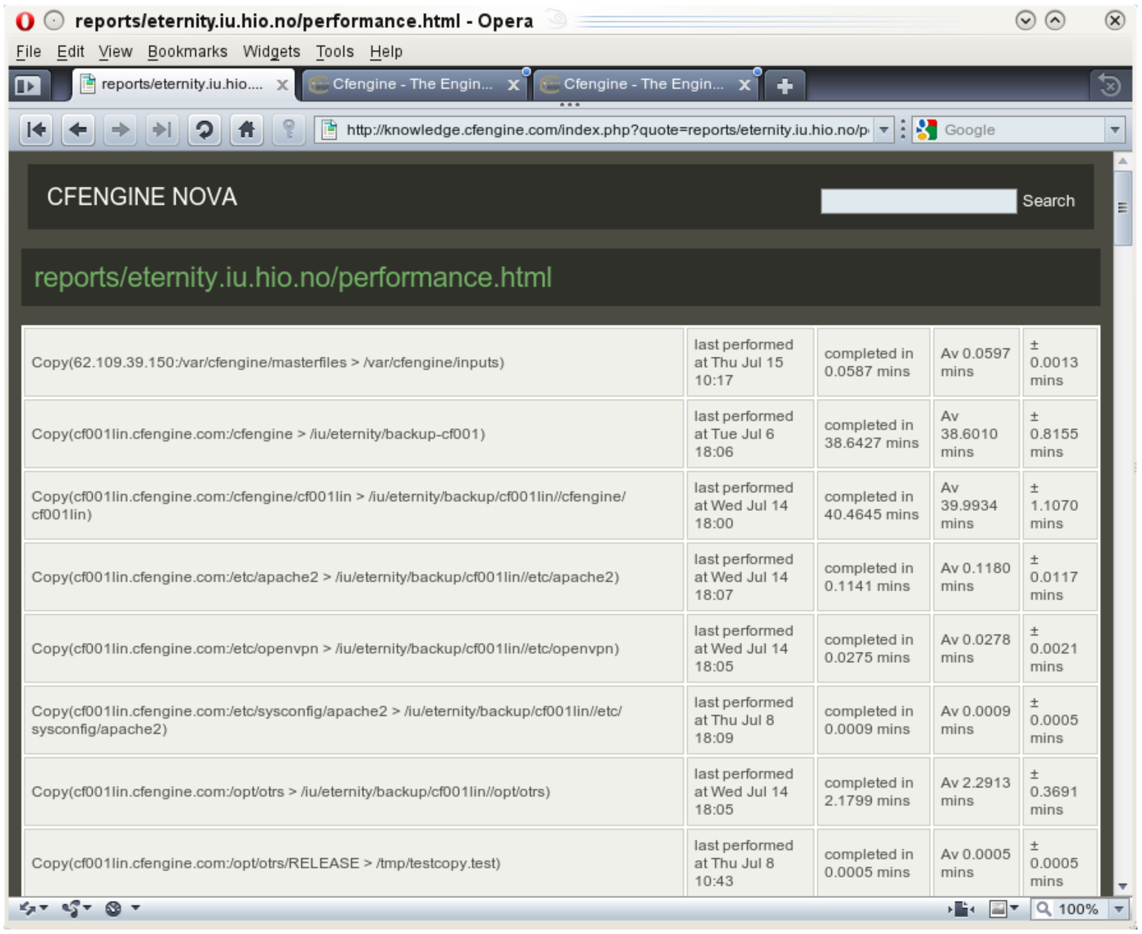
Setuid report
: Known setuid programs found on system.

Variables report
: Current variable values expanded on different hosts.

3.8.1 Performance logs

Performance logs show the last measured and average times taken to complete a particular promise from start to finish. The standard deviation of the time is quoted to show the major variations.

- Promises that have anomalous performance, especially long jobs might be worthy of attention if they do not have a predictable execution time.
- The frequency of long jobs can be restricted
- Jobs that are especially long can be backgrounded for improved throughput.

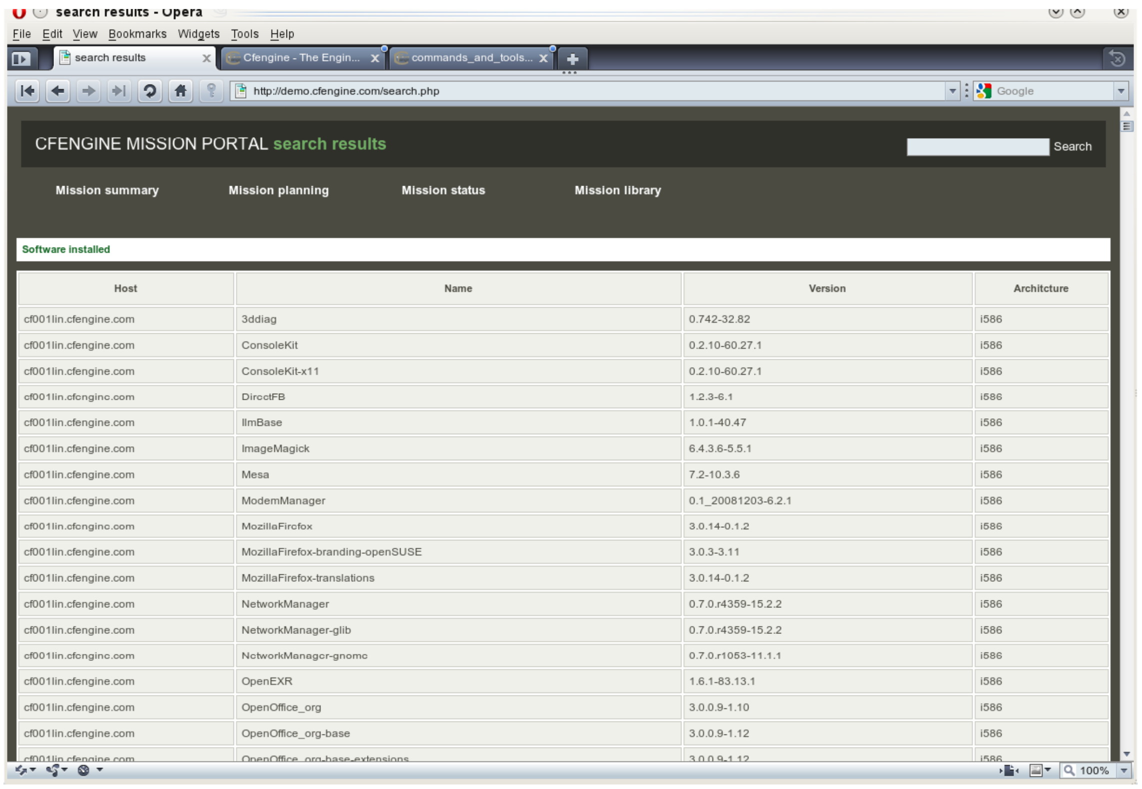

reports/eternity.iu.hio.no/performance.html - Opera
File Edit View Bookmarks Widgets Tools Help
reports/eternity.iu.hio....
Cfengine - The Engin...
Cfengine - The Engin...
http://knowledge.cfengine.com/index.php?quote=reports/eternity.iu.hio.no/p
Google
CFENGINE NOVA
Search
100%

reports/eternity.iu.hio.no/performance.html

Copy(62.109.39.150:/var/cfengine/masterfiles > /var/cfengine/inputs)	last performed at Thu Jul 15 10:17	completed in 0.0587 mins	Av 0.0597 mins	± 0.0013 mins
Copy(cf001lin.cfengine.com:/cfengine > /iu/eternity/backup-cf001)	last performed at Tue Jul 6 18:06	completed in 38.6427 mins	Av 38.6010 mins	± 0.8155 mins
Copy(cf001lin.cfengine.com:/cfengine/cf001lin > /iu/eternity/backup/cf001lin//cfengine/cf001lin)	last performed at Wed Jul 14 18:00	completed in 40.4645 mins	Av 39.9934 mins	± 1.1070 mins
Copy(cf001lin.cfengine.com:/etc/apache2 > /iu/eternity/backup/cf001lin//etc/apache2)	last performed at Wed Jul 14 18:07	completed in 0.1141 mins	Av 0.1180 mins	± 0.0117 mins
Copy(cf001lin.cfengine.com:/etc/openvpn > /iu/eternity/backup/cf001lin//etc/openvpn)	last performed at Wed Jul 14 18:05	completed in 0.0275 mins	Av 0.0278 mins	± 0.0021 mins
Copy(cf001lin.cfengine.com:/etc/sysconfig/apache2 > /iu/eternity/backup/cf001lin//etc/sysconfig/apache2)	last performed at Thu Jul 8 18:09	completed in 0.0009 mins	Av 0.0009 mins	± 0.0005 mins
Copy(cf001lin.cfengine.com:/opt/otrs > /iu/eternity/backup/cf001lin//opt/otrs)	last performed at Wed Jul 14 18:05	completed in 2.1799 mins	Av 2.2913 mins	± 0.3691 mins
Copy(cf001lin.cfengine.com:/opt/otrs/RELEASE > /tmp/testcopy.test)	last performed at Thu Jul 8 10:43	completed in 0.0005 mins	Av 0.0005 mins	± 0.0005 mins

3.8.2 Installed software packages

An overview of installed software as viewed through the eyes of the locally installed package manager. These data are collected as a byproduct of the `packages` promises. You must check for the existence of at least one package to generate this report. Moreover, the report can only be generated by sufficiently smart package managers or by custom scripts (see the ‘`packages`’ promise in the reference manual).



Software installed

Host	Name	Version	Architcture
cf001lin.cfengine.com	3ddiag	0.742-32.82	i586
cf001lin.cfengine.com	ConsoleKit	0.2.10-60.27.1	i586
cf001lin.cfengine.com	ConsoleKit-x11	0.2.10-60.27.1	i586
cf001lin.cfengine.com	DirectFB	1.2.3-6.1	i586
cf001lin.cfengine.com	IlmBase	1.0.1-40.47	i586
cf001lin.cfengine.com	ImageMagick	6.4.3.6-5.5.1	i586
cf001lin.cfengine.com	Mesa	7.2-10.3.6	i586
cf001lin.cfengine.com	ModemManager	0.1_20081203-6.2.1	i586
cf001lin.cfengine.com	MozillaFirefox	3.0.14-0.1.2	i586
cf001lin.cfengine.com	MozillaFirefox-branding-openSUSE	3.0.3-3.11	i586
cf001lin.cfengine.com	MozillaFirefox-translations	3.0.14-0.1.2	i586
cf001lin.cfengine.com	NetworkManager	0.7.0.r4359-15.2.2	i586
cf001lin.cfengine.com	NetworkManager-glib	0.7.0.r4359-15.2.2	i586
cf001lin.cfengine.com	NetworkManager-gnome	0.7.0.r1053-11.1.1	i586
cf001lin.cfengine.com	OpenEXR	1.6.1-83.13.1	i586
cf001lin.cfengine.com	OpenOffice_org	3.0.0.9-1.10	i586
cf001lin.cfengine.com	OpenOffice_org-base	3.0.0.9-1.12	i586
cf001lin.cfengine.com	OpenOffice_org-base-extensions	3.0.0.9-1.12	i586

3.8.3 A promise compliance report

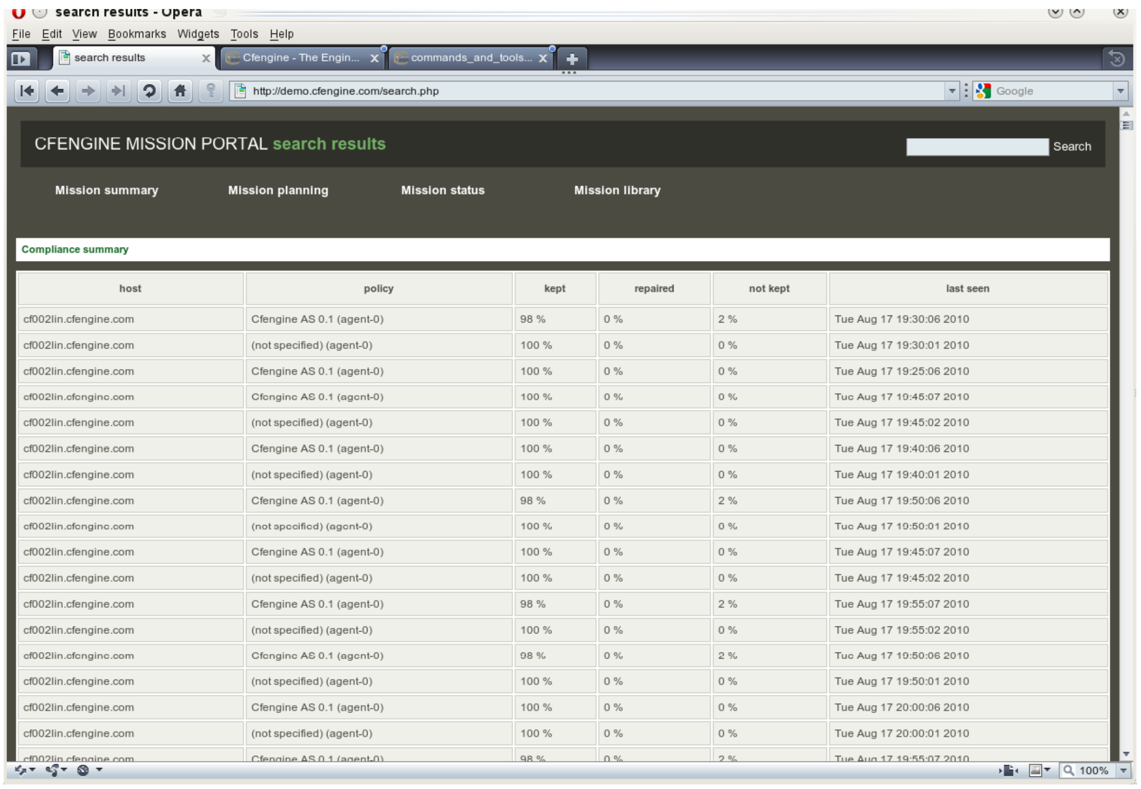
Each time `cf-agent` completes a scan of whether local promises are kept it computes a simple high level summary of the state of the system in terms of whether promises were kept before and after execution.

The agent distinguished three categories

Promise kept
: The promise was already kept, and no action was required on the system.

Promise repaired
: The promise was not properly kept but maintenance was performed and the problem was healed. The system is now in compliance.

Promise not kept
: The promise was not kept, and no remedial action was taken to correct it. (This is usually a result of an explicit policy decision to not repair a problem, e.g. selecting action ‘warn’ instead of ‘fix’.)



Compliance summary

host	policy	kept	repaired	not kept	last seen
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	98 %	0 %	2 %	Tue Aug 17 19:30:06 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:30:01 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:25:06 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:45:07 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:45:02 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:40:06 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:40:01 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	98 %	0 %	2 %	Tue Aug 17 19:50:06 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:50:01 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:45:07 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:45:02 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	98 %	0 %	2 %	Tue Aug 17 19:55:07 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:55:02 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	98 %	0 %	2 %	Tue Aug 17 19:50:06 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 19:50:01 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	100 %	0 %	0 %	Tue Aug 17 20:00:06 2010
cf002lin.cfengine.com	(not specified) (agent-0)	100 %	0 %	0 %	Tue Aug 17 20:00:01 2010
cf002lin.cfengine.com	Cfengine AS 0.1 (agent-0)	98 %	0 %	2 %	Tue Aug 17 19:55:07 2010

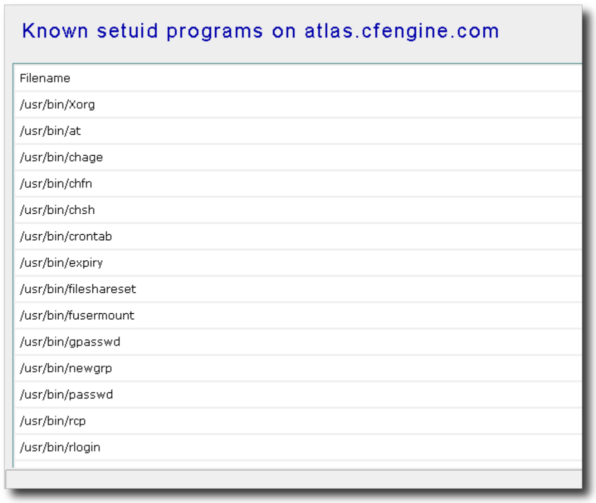
3.8.4 A file content change report

Hash scans of files are a common way of detecting any small changes in file integrity. This log shows the specific times that change events were detected. The resolution of this log depends on the frequency of scanning, which is the purview of a `files` promise.

If you enable the `report_diffs` feature in Nova, on individually named files, a second 'change tracking' report will be aggregated, summarizing the major changes. On text-only files this reports major changed in the spirit of the Unix `diff` command, though comparatively simplified for a readability. Users can then use other more specialized difference tools for alternative view for forensic detail on the host concerned.

3.8.5 Installed setuid program report

Setuid root programs (Unix systems) are programs given special privileges to carry out actions normally restricted to Administrative or root users. Attention to programs with setuid permissions is a matter of security, and thus CFEngine watches out for changes in the known list of these programs as a matter of course. This report summarizes the known programs.



Known setuid programs on atlas.cfengine.com

Filename
/usr/bin/Xorg
/usr/bin/at
/usr/bin/chage
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/crontab
/usr/bin/expiry
/usr/bin/fileshareset
/usr/bin/fusermount
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/rcp
/usr/bin/rlogin

3.9 Knowledge Map

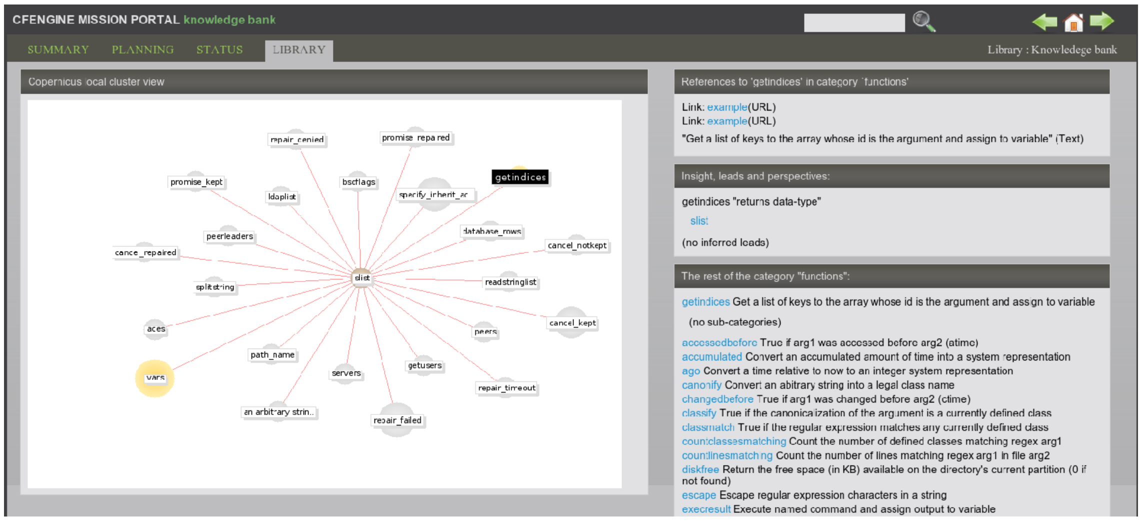
The Knowledge Map is a tool that may be used as:

- An index of the information in the Knowledge Map
- A tool for brainstorming
- A tool for self-learning

It is designed to show relationships between subjects of importance. The relationships are shown graphically and in text format.

The text relationships on the right hand side of the screen show direct relationships between the current subject and neighboring subjects, but with semantic explanations. The graphical renditions attempt to show greater depth if possible. Due to the limited size of the screen, and limitations of human cognition, only about twenty of the closest relationships are depicted to create a sense of 'where we are' in the landscape of knowledge. The actual map might contain many thousands of entries.

There are up to two graphical representations on each page. The Copernicus view is a depiction of the current topic in relation to its most important neighbors. The nodes in the graph are shown as 'planetary balls'. The highest ranking (most connected) topic is shown in yellow, and each node is sized according to its rank. The current topic has a black label, whereas all others have a white label.



References to 'getindices' in category 'functions'

Link: example(URL)
Link: example(URL)
"Get a list of keys to the array whose id is the argument and assign to variable" (Text)

Insight, leads and perspectives:

getindices "returns data-type"
slist
(no inferred leads)

The rest of the category "functions":

getindices Get a list of keys to the array whose id is the argument and assign to variable
(no sub-categories)
accessedbefore True if arg1 was accessed before arg2 (atime)
accumulated Convert an accumulated amount of time into a system representation
ago Convert a time relative to now to an integer system representation
canonify Convert an abitrary string into a legal class name
changedbefore True if arg1 was changed before arg2 (ctime)
classify True if the canonicalization of the argument is a currently defined class
classmatch True if the regular expression matches any currently defined class
countclassesmatching Count the number of defined classes matching regex arg1
countlinesmatching Count the number of lines matching regex arg1 in file arg2
diskfree Return the free space (in KB) available on the directory's current partition (0 if not found)
escape Escape regular expression characters in a string
execresult Execute named command and assign output to variable

Figure: Copernicus view of the Knowledge Map.

On selected promise pages, two images will appear. The lower image called 'influence channel', shows a causation or dependency graph between promises. This graph offers a simple form of impact analysis. The links between subjects show only dependency relationships.

"What am I looking for in a knowledge base?"

There are many reasons to browse knowledge. You browse an encyclopedia because you are curious. You look up in a dictionary because you want an answer. You look at wind-dials and thermometers to gauge the environment. You look at dependencies and diagnostics to answer: what is the *cause* of some effect?

The helm screen is divided into several parts. On the right hand side there are two images on each page.

3.9.1 The knowledge map pages

Each page of the console represents a single topic in the documentation. The topic name is described in the title of the page. Topics fall into categories, and these are reflected in the name

```
Category::Topic
```

Categories are also topics that can be looked up. Clicking on one of the links centers your page view on that particular topic.

3.9.2 The knowledge base maps

On each page you will find one or two graphics that illustrate the current topic and its relationship to neighboring leads and perspectives. This neighborhood shows you up to 30 of the closest related topics, but it does not show you *how* they are related. This picture is useful for seeing the 'bigger picture', but for a more detailed understanding you should look at the *Insight, leads and Perspectives* panel. That panel tells you the exact nature of the relationship to the closest topics in the map: in other words, what should you expect to learn by following that link?

The graphical representation of the knowledge map shows the local neighborhood you are currently exploring. Clicking on a panel to the right will enlarge it. The current topic will be shown in yellow (with red text). You will only see a maximum of thirty topics at any one time.

The relative importance of the topics is shown by their size, and size is determined by how many references a topic has to nearby topics. The most important topic is placed as the star at the center of this 'solar system' view. Other nodes are colors pink, brown, grey depending on their distance from it.

The current topic might not be the most important in the neighborhood you are exploring. The graphs are designed to guide you towards what is important in your system and its work-flows.

3.9.3 Results for this topic

This panel shows document references and information pertaining directly to the current topic (if any). The red links point to URL documents. The text to the left explains what kind of information you should expect to find about the current topic.

3.9.4 Topics that have Category::Topic as a subject-header

If the current topic is a category with several sub-themes, this panel contains a list of possible sub-topics that lie under this category. In other words, the list shown here is a depth search in the knowledge base.

3.9.5 Other topics found under Category

This panel shows any other topics in the same category as the current topic. In other words, the list shown here is a breadth search of the knowledge base.

3.10 Using the knowledge map

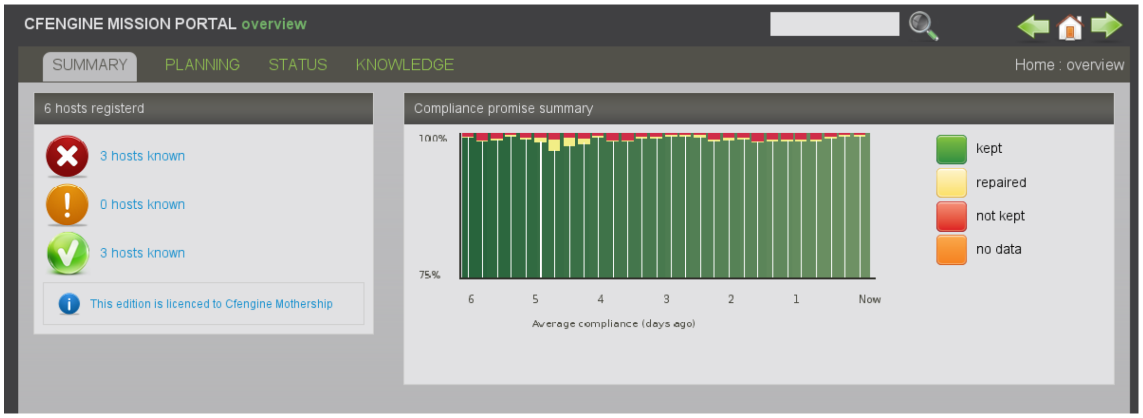
Go to the URL set up for the policy server, using a web browser of your choice.

```
http://policy_server.example.com
```

When you look at the knowledge base for the first time, you can either explore using the links, or type in a search word into the top field to jump to the closest matching topic.

No matter where you begin your search, you will end up on a page which has the same basic structure (described above).

The front page of the knowledge map is shown in the figure below. This has a number of links and a number of graphic images which are also links.

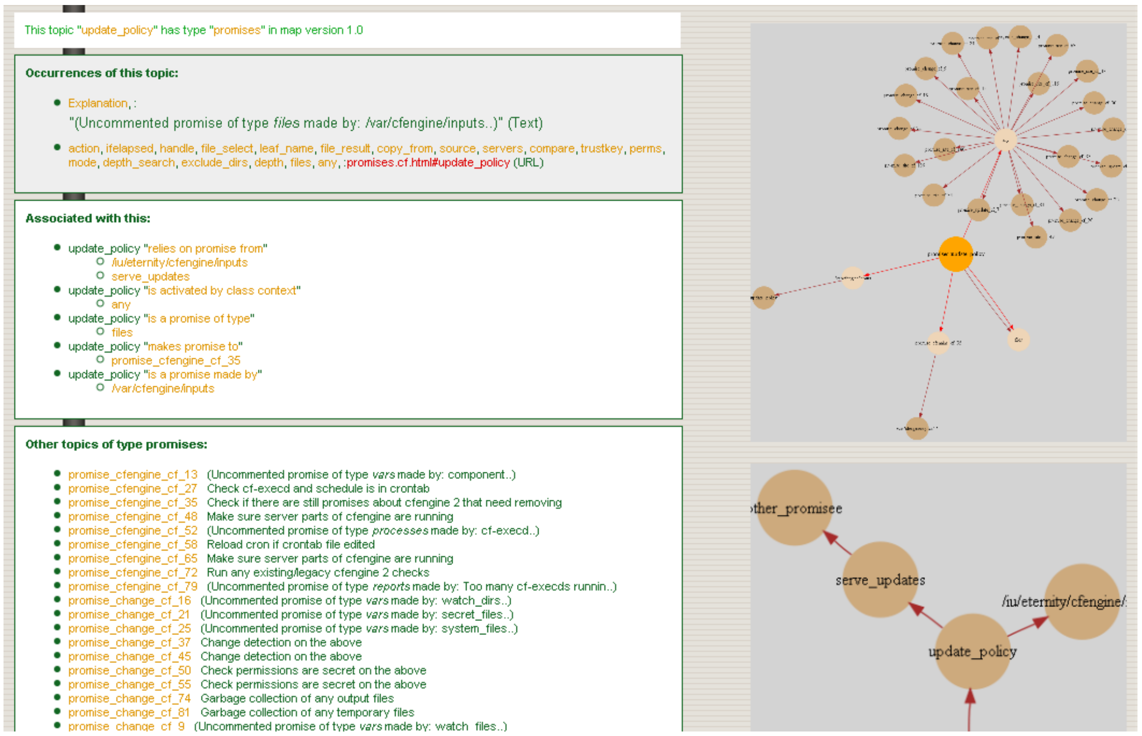


3.10.1 Policy analysis

CFEngine helps you to make sense of the knowledge implicit in your current CFEngine policy, by turning it into an annotated map. You can extend the automatically generated map with your own comments, topics and documents, in order to tie in with your own enterprise information. The knowledge map will normally run on a single machine, such as the CFEngine policy server, but this is not a requirement.

The example figure below shows typical page generated by the knowledge agent. It describes a CFEngine promise from the policy files. One such page is generated for each policy promise, and pages may be generated for reports from different computers. You can also create 'topic pages' for any local (enterprise) information that you have. In this case the promise has been given the promise-handle `update_policy`, and the associations and the lower graph shows how this promise relates to other promises through its documented dependencies (these are documented from the promisees and `depends_on` attributes of other promises.).

The example page shows two figures, one above the other. The upper figure shows the thirty nearest topics (of any kind) that are related to this one. Here the relationships are unspecific. This diagram can reveal pathways to related information that are often unexpected, and illustrates relationships that broaden one's understanding of the place the current promise occupies within the whole.


This topic "update_policy" has type "promises" in map version 1.0

Occurrences of this topic:

- Explanation, :
  "(Uncommented promise of type *files* made by: /var/cfengine/inputs..)" (Text)
- action, ifelapsed, handle, file_select, leaf_name, file_result, copy_from, source, servers, compare, trustkey, perms, mode, depth_search, exclude_dirs, depth, files, any, :promises.cf.html#update_policy (URL)

Associated with this:

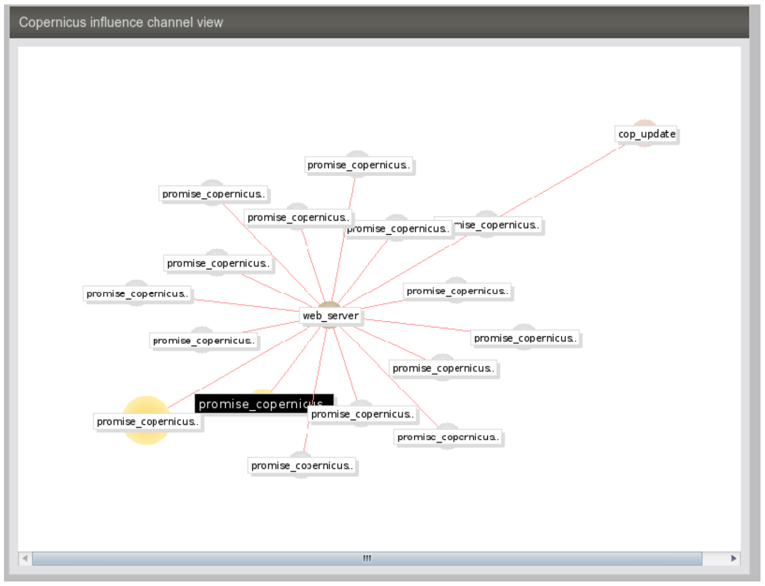
- update_policy "relies on promise from"
  - /iu/eternity/cfengine/inputs
  - serve_updates
- update_policy "is activated by class context"
  - any
- update_policy "is a promise of type"
  - files
- update_policy "makes promise to"
  - promise_cfengine_cf_35
- update_policy "is a promise made by"
  - /var/cfengine/inputs

Other topics of type promises:

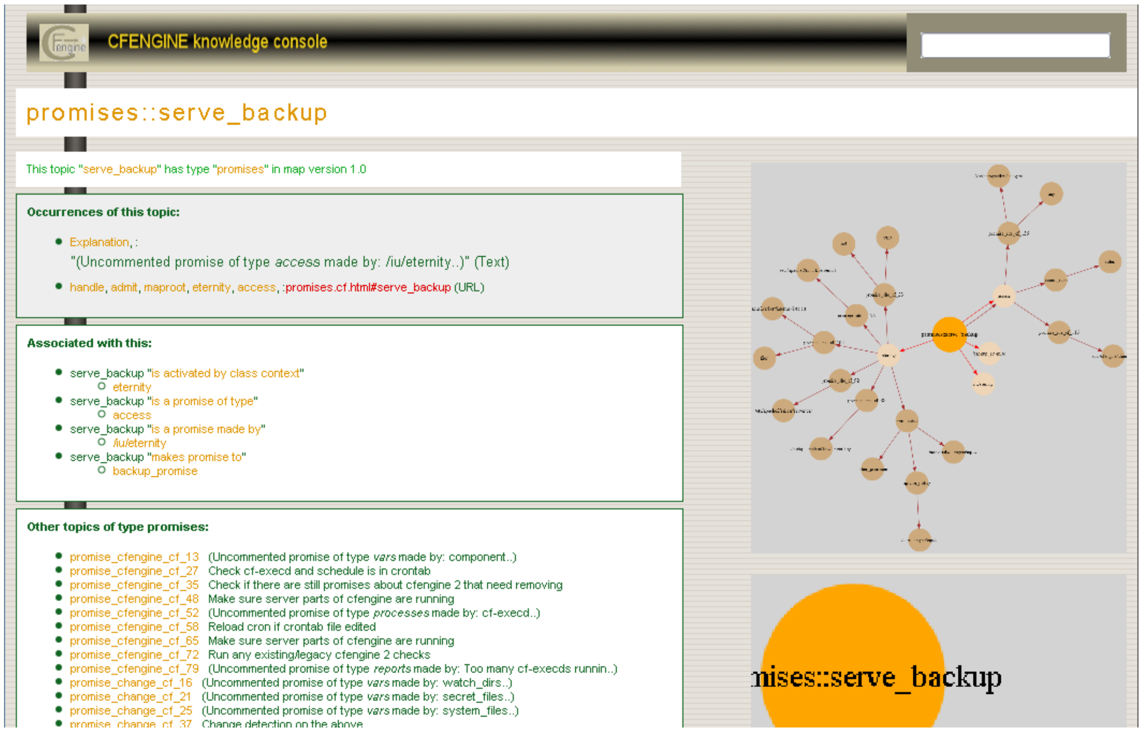
- promise_cfengine_cf_13 (Uncommented promise of type *vars* made by: component..)
- promise_cfengine_cf_27 Check cf-execd and schedule is in crontab
- promise_cfengine_cf_35 Check if there are still promises about cfengine 2 that need removing
- promise_cfengine_cf_48 Make sure server parts of cfengine are running
- promise_cfengine_cf_52 (Uncommented promise of type *processes* made by: cf-execd..)
- promise_cfengine_cf_58 Reload cron if crontab file edited
- promise_cfengine_cf_65 Make sure server parts of cfengine are running
- promise_cfengine_cf_72 Run any existing/legacy cfengine 2 checks
- promise_cfengine_cf_79 (Uncommented promise of type *reports* made by: Too many cf-execds runnin..)
- promise_change_cf_16 (Uncommented promise of type *vars* made by: watch_dirs..)
- promise_change_cf_21 (Uncommented promise of type *vars* made by: secret_files..)
- promise_change_cf_25 (Uncommented promise of type *vars* made by: system_files..)
- promise_change_cf_37 Change detection on the above
- promise_change_cf_45 Change detection on the above
- promise_change_cf_50 Check permissions are secret on the above
- promise_change_cf_55 Check permissions are secret on the above
- promise_change_cf_74 Garbage collection of any output files
- promise_change_cf_81 Garbage collection of any temporary files
- promise_change_cf_9 (Uncommented promise of type *vars* made by: watch_files..)

The second figure shows a blow up of the dependency map illustration. It is a subset of the larger picture, choosing only those topics that are linked by impact-related associations.

Although these illustrations are just renderings of the 'leads and perspectives' associations shown in text, they are useful for visualizing several levels of depth in the associative network. In particular, one can see the other promises that could be affected if we were to make a change to the current promise. Such impact analyses can be crucial to planning change and release management of policy. ITIL best practices recommend thorough pre-analyses of release changes. CFEngine Nova combined with good practice of documentation makes this straightforward.



Another example topic page.



promises::serve_backup

This topic "serve_backup" has type "promises" in map version 1.0

Occurrences of this topic:

- Explanation, :
  "(Uncommented promise of type *access* made by: /iu/eternity..)" (Text)
- handle, admit, maproot, eternity, access, :promises.cf.html#serve_backup (URL)

Associated with this:

- serve_backup "is activated by class context"
  - eternity
- serve_backup "is a promise of type"
  - access
- serve_backup "is a promise made by"
  - /iu/eternity
- serve_backup "makes promise to"
  - backup_promise

Other topics of type promises:

- promise_cfengine_cf_13 (Uncommented promise of type *vars* made by: component..)
- promise_cfengine_cf_27 Check cf-execd and schedule is in crontab
- promise_cfengine_cf_35 Check if there are still promises about cfengine 2 that need removing
- promise_cfengine_cf_48 Make sure server parts of cfengine are running
- promise_cfengine_cf_52 (Uncommented promise of type *processes* made by: cf-execd..)
- promise_cfengine_cf_58 Reload cron if crontab file edited
- promise_cfengine_cf_65 Make sure server parts of cfengine are running
- promise_cfengine_cf_72 Run any existing/legacy cfengine 2 checks
- promise_cfengine_cf_79 (Uncommented promise of type *reports* made by: Too many cf-execds runnin..)
- promise_change_cf_16 (Uncommented promise of type *vars* made by: watch_dirs..)
- promise_change_cf_21 (Uncommented promise of type *vars* made by: secret_files..)
- promise_change_cf_25 (Uncommented promise of type *vars* made by: system_files..)
- promise_change_cf_37 Change detection on the above

A CFEngine knowledge base is not a new document format, it is an overlay map that joins ideas and resources together, and displays relationships.

3.10.2 Understanding dependencies

Nova adds automatically instrumentation to the policy compilation that allows you to trace dependencies, provided you follow the best-practice when constructing your promise rules.

When writing promises, get into the habit of giving every promise a comment that explains its intention.

Also, give related promises *handles*, or labels that can be used to refer to them by.

```
files:

  "/var/cfengine/inputs"

    handle => "update_policy",

    perms => system("600"),
    copy_from => mycopy("$(master_location)","$(policy_server)"),
    depth_search => recurse("inf"),
    file_select => input_files,
    action => immediate;
```

If a promise affects another promise in some way, you can make the affected promise one of the promisees, like this:

```
access:

  "/master/CFEngine/inputs" -> { "update_policy", "other_promisee" },

    handle  => "serve_updates",

    admit   => { "217.77.34.*" };
```

Conversely, if a promise might depend on another in some (even indirect) way, document this too.

```
files:

  "/var/cfengine/inputs"

    handle     => "update_policy",
    depends_on => {"serve_updates"},

    perms => system("600"),
    copy_from => my-copy("$(master_location)","$(policy_server)"),
    depth_search => recurse("inf"),
    file_select => input_files,
    action => immediate;
```

The lower of the two diagrams on the knowledge page (called the influence channel) shows graphically how dependency trails flow. The text links show the specific of each relationship.

- When a bundle is passed a parameter by another
- In method calls, a passed variable value is considered a causal influence.
- When a variable is used in a shell command
- When a variable is set by a function call that retrieves from a database/file

3.10.3 A quick tour of the knowledge console

1. Click on the top bar to go back to the start page. This takes you to the mission status summary.
2. Look at the different panels
   - References to this topic - showing immediate results and documents
   - Topic (current focus) includes - showing sub categories of this major heading
   - Insight, Leads and perspectives - show intelligent hints and tips
   - Other topics found under (major heading) - similar topics to the current
3. Click on ‘Manager view’, and look at the new panels. Explore at will.
4. Click on the upper network image to the right, showing topics that are closely related through ‘leads and perspectives’. Click back to the mission status page.
5. Click on the bar-meter ‘site summary’. This takes you to the monitoring portal for individual machines. Here you see all managed hosts.
6. Click on a new bar meter to drill down into performance statistics, and keep clicking until you can go no further to see detail.
7. Go back to main page and find the search field at the top right. Enter ‘BDMA’ to search for this topic.

3.10.4 Why knowledge management?

At CFEngine, we believe that Knowledge Management is the challenge of our times. As futurist Alvin Toffler wrote in his seminal 1970 study *Future Shock* about the rapidity of industrial changes of the ’60s:

> "*Rising novelty renders irrelevant the traditional goals of our chief institutions...Acceleration produces a faster turnover of goals. Diversity or fragmentation leads to a relentless multiplication... Caught in this churning, goal-cluttered environment, we stagger...from crisis to crisis, pursuing a welter of conflicting and self-cancelling purposes.*"

We see it as a key strategy to provide simple tools that explain *intentions* and *meaning* amongst the welter of system documentation and information.

3.11 cf-hub and data collection

The ‘hub’ is both a machine and a process in a default centralized reporting architecture. For most purposes, CFEngine avoids centralization of effort in order to scale efficiently. For reporting, however, one needs a common location and standard of data collection to provide accurate information, so we make an exception. To make hub aggregation scale efficiently, Nova 2 uses an extremely lightweight data encoding that is several hundred times faster than the approach used in Nova 1. Using a hub, the process can be parallelized and kept up to date more often than before.

```
host# /var/cfengine/bin/cf-hub
```

The CFEngine hub runs in the background and connects to all known registered hosts and builds a summary model of their state in a database. This database is searchable and may be browsed using the Mission Portal. Updates may be collected manually also:

```
host# cf-runagent -H hostname -q delta
host# cf-runagent -H hostname -q full
```

3.11.1 CMDB or SKMS

Last generation systems have used the concept of the Configuration Management Database (CMDB) as a central repository of information. The CMDB concept was primitive and data-intensive; it had no concept of intended versus un-intended state. Today the term SKMS or Service Knowledge Management System is coming into use, based in ITIL version 3. The idea here is to go beyond mere data-mining to a knowledge oriented system.

CFEngine Nova provides a ‘next generation’ knowledge-oriented overview of IT operations in its Mission Portal. It combines intended state maintained through policy, with actual state measured by observation, and relates these for the user in a semantic web. This enables users to relate measured data to actual strategic goals, and ‘connect the dots between Business and IT’.

3.11.2 The reporting agent

A control body for the reporter looks like this:

```
body reporter control
{
reports => { "performance", "last_seen", "monitor_history" };
build_directory => "/var/cfengine/reports";
report_output => "html";
}
```

- Knowledge management leads to better IT decisions
- Advance from knowledge hunter-gathers to a knowledge-sharing society
- Knowledge yields improved response to business needs
- Sharing the lessons learned results in best practices across the organization.
- Knowledge gives the accuracy and precision to succeed

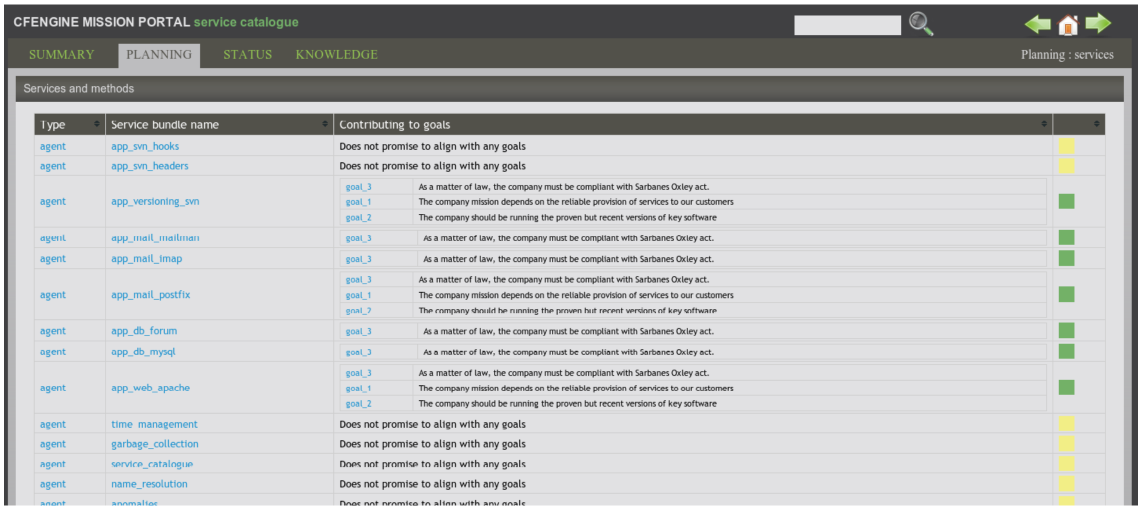
4 Business Integration

4.1 Business goals and the Service Catalogue

Trust and confidence form just one of the pillars of business; agility and creativity make up the others. CFEngine Nova brings features that make it possible to connect the dots meaningfully between high level business goals and low level configuration implementation. This enables different layers of an organization to gain insight into how the system is being managed, in relation to the strategic goals.

- CFEngine Nova's rich repertoire of configuration capabilities allows model-based integration of systems, attending to the unique requirements of each business service. Off-the-shelf does not have to mean bland and generic. Customization is CFEngine's forte. It is the glue holding applications together.
- Nova's speed and use of patterns allows many small changes to be made very quickly.

Nova provides appropriate insight into IT operations through explanation, reporting and visualization of policy, allowing business and IT to gather around information of mutual interest. Of course, CFEngine is not going to make IT experts of business staff, or vice versa. However, business and IT work best together when then can communicate effectively. Our aim with Nova is to give business heads just the right level of insight into technical IT operations to be assured that their needs are being met[1]. Similarly, Nova makes business goals available to IT staff so that they can feel connected and responsible for the strategic goals of the organization.



Type	Service bundle name	Contributing to goals	
agent	app_svn_hooks	Does not promise to align with any goals	🟨
agent	app_svn_headers	Does not promise to align with any goals	🟨
agent	app_versioning_svn	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act. goal_1 The company mission depends on the reliable provision of services to our customers goal_2 The company should be running the proven but recent versions of key software	🟩
agent	app_mail_mailman	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act.	🟩
agent	app_mail_imap	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act.	🟩
agent	app_mail_postfix	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act. goal_1 The company mission depends on the reliable provision of services to our customers goal_2 The company should be running the proven but recent versions of key software	🟩
agent	app_db_forum	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act.	🟩
agent	app_db_mysql	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act.	🟩
agent	app_web_apache	goal_3 As a matter of law, the company must be compliant with Sarbanes Oxley act. goal_1 The company mission depends on the reliable provision of services to our customers goal_2 The company should be running the proven but recent versions of key software	🟩
agent	time_management	Does not promise to align with any goals	🟨
agent	garbage_collection	Does not promise to align with any goals	🟨
agent	service_catalogue	Does not promise to align with any goals	🟨
agent	name_resolution	Does not promise to align with any goals	🟨
agent	anomalies	Does not promise to align with any goals	🟨

A human-readable Service Catalogue generated from technical specifications shows what goals are being attended to automatically

The Service Catalogue is a concept that has been promoted as part of the IT Infrastructure Library (ITIL). Nova generates a service catalogue directly from a model of the knowledge about policy.

[1] Often it is enough to be able to ask a relevant question and feel that one has a voice, while experts are taking care of the details.

```
bundle agent service_catalogue
{
methods:

   "everyone" usebundle => time_management,
                comment => "Ensure clocks are synchronized";

   "everyone" usebundle => garbage_collection,
                comment => "Keep disks free of junk and rotate logs";

   "everyone" -> goal_3,

              usebundle => change_management,
                 comment=> "Monitor change";

   "managed hosts" -> goal_1,

                   usebundle => CFEngine_management,
                     comment => "Management of CFEngine internals";

   "mail server"  -> { "goal_3", "goal_1", "goal_2" }

                  usebundle => app_mail_postfix,
                    comment => "The mail delivery agent";
}
```

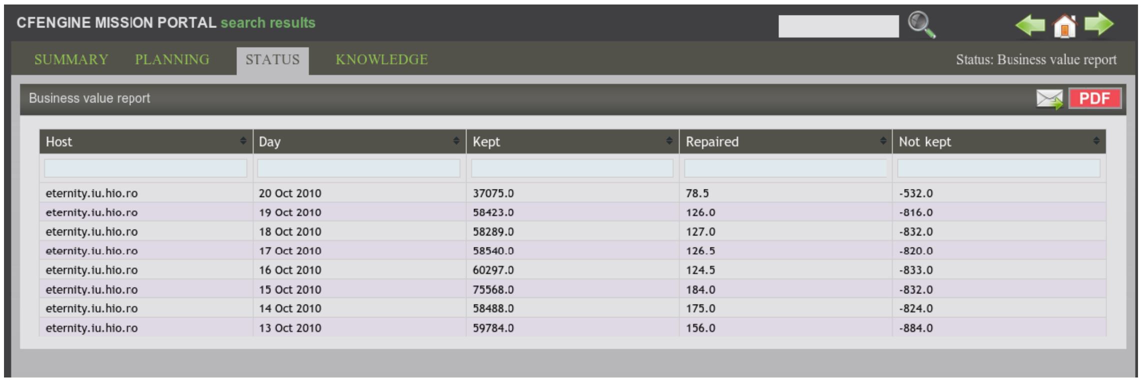
Technical IT administrators can document which promises or bundles of promises are directed towards specific goals by adding a simple reference The Nova Knowledge Map will then build a narrative around these references and tie the goals to the services in the catalogue (See the Planning tab).

4.2 Business value reports

One of the capabilities of CFEngine is to add business or organizational value to the configuration model. The notion of business value is not a clear concept, but a very simple approach to measuring value is to attach a monetary value to the outcome of promises.

The `value_kept` (default value 1), `value_repaired` (default value 0.5), `value_notkept` (default value -1) settings fall under CFEngine transaction logging and allow administrators to attach actual monetary (or other) values to promises kept, or issues repaired, or conversely measure the loss of non-compliance in dollar terms (choose your currency). This value is summed and recorded for each execution of CFEngine, and can be turned into graphs for your management reports.

An example of this report is shown below. The results are summed for each day and presented in column form.

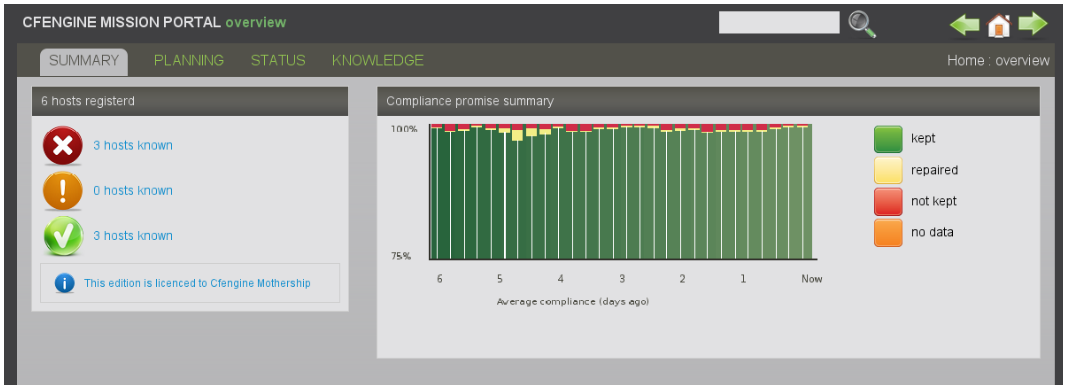

CFENGINE MISSION PORTAL search results
SUMMARY
PLANNING
STATUS
KNOWLEDGE
Status: Business value report
Business value report
PDF

Host	Day	Kept	Repaired	Not kept
eternity.iu.hio.ro	20 Oct 2010	37075.0	78.5	-532.0
eternity.iu.hio.ro	19 Oct 2010	58423.0	126.0	-816.0
eternity.iu.hio.ro	18 Oct 2010	58289.0	127.0	-832.0
eternity.iu.hio.ro	17 Oct 2010	58540.0	126.5	-820.0
eternity.iu.hio.ro	16 Oct 2010	60297.0	124.5	-833.0
eternity.iu.hio.ro	15 Oct 2010	75568.0	184.0	-832.0
eternity.iu.hio.ro	14 Oct 2010	58488.0	175.0	-824.0
eternity.iu.hio.ro	13 Oct 2010	59784.0	156.0	-884.0

Business value accounting can be entered into the model.

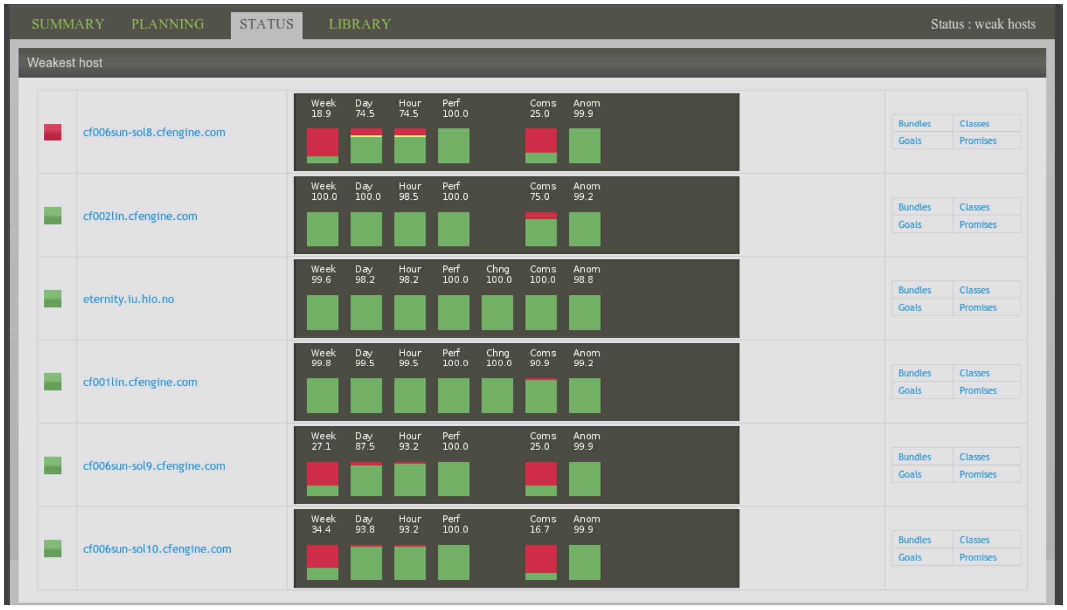
4.3 Simple insight into IT compliance

While business and IT can come together to decide what promises IT services should keep, a simple confirmation of that compliance helps business units to be assured



An overview of compliance

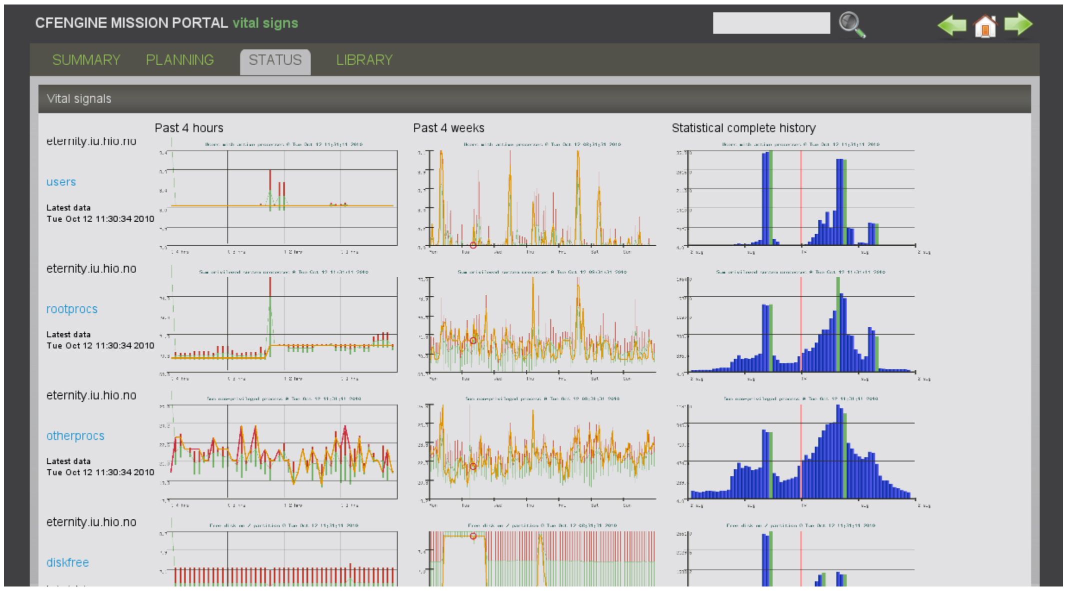
Increasing and graded levels of detail can be obtained to match the level of expertise of the viewer. Bar meters can represent more specific details.



	Host	Week	Day	Hour	Perf	Chng	Coms	Anom
🟥	cf006sun-sol8.cfengine.com	18.9	74.5	74.5	100.0		25.0	99.9
🟩	cf002lin.cfengine.com	100.0	100.0	98.5	100.0		75.0	99.2
🟩	eternity.iu.hio.no	99.6	98.2	98.2	100.0	100.0	100.0	98.8
🟩	cf001lin.cfengine.com	99.8	99.5	99.5	100.0	100.0	90.9	99.2
🟩	cf006sun-sol9.cfengine.com	27.1	87.5	93.2	100.0		25.0	99.9
🟩	cf006sun-sol10.cfengine.com	34.4	93.8	93.2	100.0		16.7	99.9

Weakest hosts broken down by compliance, performance and change.

For system analysts, Nova can recognize and highlight trends and patterns of behavior that inform further strategic decision making.



eternity.iu.hio.no
users
Latest data
Tue Oct 12 11:30:34 2010

eternity.iu.hio.no
rootprocs
Latest data
Tue Oct 12 11:30:34 2010

eternity.iu.hio.no
otherprocs
Latest data
Tue Oct 12 11:30:34 2010

eternity.iu.hio.no
diskfree

Analysis of system performance over multiple time-scales
can aid with resource deployment and future planning.

5 Monitoring extensions

CFEngine Nova incorporates a lightweight monitoring agent, whose aim is to provide meaningful performance data about systems, in a scalable fashion. CFEngine Nova does not aim to replace specialized rapid-update monitoring and alarm systems; it provides a context-aware summary of current state that is always displayed in relation to previous system behavior, for comparison. The aim is to offer useful analytics rather than jump-to alarms.

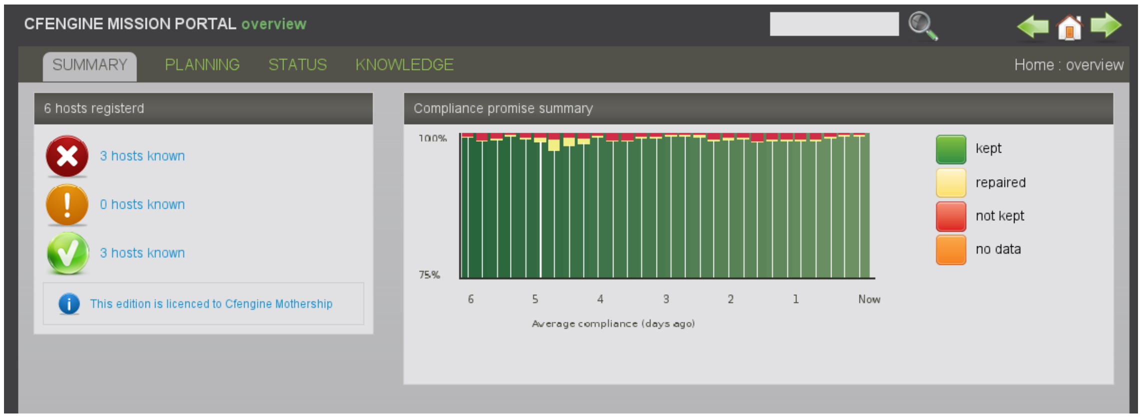
CFEngine's monitoring component `cf-monitord` records a number of performance data about the system by default. These include process counts, service traffic, load average and CPU utilization and temperature when available. In the Community Edition, data are only collected and stored for personal use, but users have to work to see results. CFEngine Nova improves on this in three ways.

- Data collected from the monitoring system are integrated into the aggregate knowledge console.
- It adds a three year life-cycle trend summary, based any 'shift'-averages.
- It add customizable promises to monitor or log specific highly specific user data through the generic promise interface.

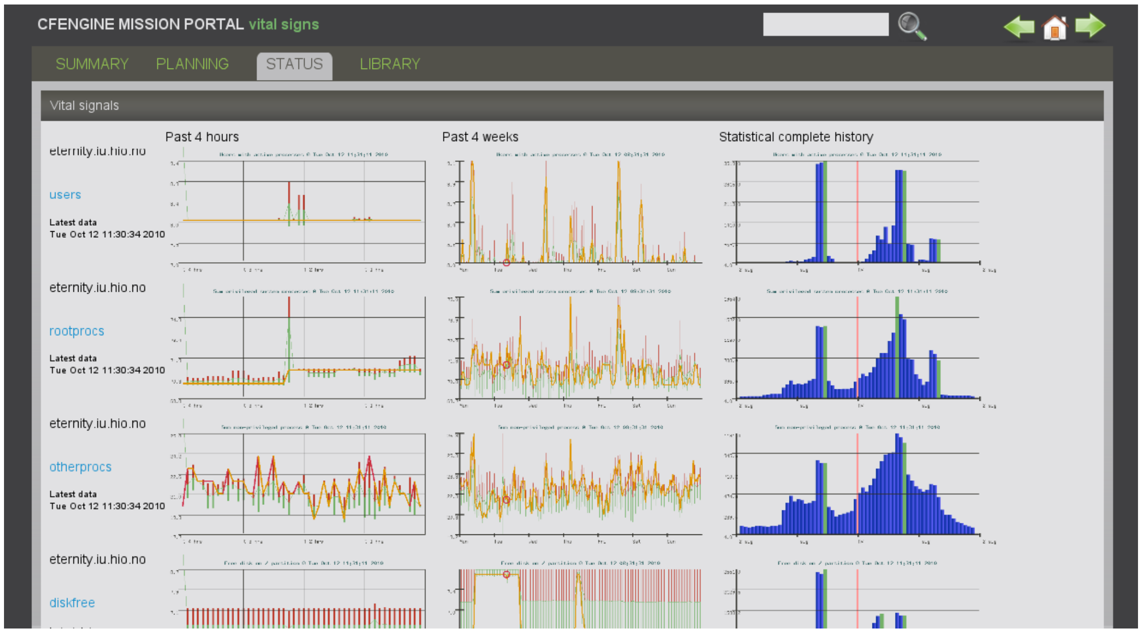
The end result is to display time series traces of system performance, like the above mentioned values, and customized logs feeding custom-defined reports.

5.1 Integration of monitoring with knowledge base

CFEngine Nova integrates monitoring reports with the automated base knowledge to provided self-analysis and simple summary reporting. A 'graphic equalizer' view shows important status summaries, and the semantic links allow users to drill down to specific reporting data.



Detailed time-series views can be collected and collated, providing honest and accurate data that allow you to gauge your own confidence level in system performance. Unlike most monitoring solutions, CFEngine shows you its own confidence in the measurements taken. It takes a finite amount of time to measure and transport data from systems to the knowledge console. That time also provides information about system performance. CFEngine always promises to tell you how old data are and how confident it is in the values.



5.2 Long term trends

CFEngine normally operates with time-series data represented in two forms:

- A weekly average, plotted on a periodogram, showing performance now in relation to the same time of week in previous weeks. After about a month data are forgotten to ensure a sufficient rate of adaptation to new patterns.
- The past four hours in high resolution.

CFEngine Nova adds quarter day averages of recorded time-series which go back three years in time. Three years is considered to be the lifetime of a computer. Summaries of the detailed performance are summarized by flat averages for a four-shift day:

- **Night shift**: from midnight 00:00 to 06:00
- **Morning shift**: from 06:00 to 12:00
- **Afternoon shift**: from 12:00 to 18:00
- **Evening shift**: from 18:00 to 00:00

5.3 Custom promises to measure

CFEngine Nova adds a new promise type in bundles for the monitoring agent. These are written just like all other promises within a bundle destined for the agent concerned. In this case:

```
bundle monitor watch

{
measurements:
  # promises ...

}
```

5.3.1 Extraction strings and logging

Let's take a generic example. Suppose we have a file of nonsense '`/tmp/testmeasure`' and we want to extract some information that we call a 'blonk' from the file. A blonk is always on the second line of this file following a prefix 'Blonk blonk '. We would get the value like this:

```
  "/tmp/testmeasure"

      handle => "blonk_watch",
      stream_type => "file",
      data_type => "string",
      history_type => "log",
      units => "blonks",
      match_value => find_blonks,
      action => sample_min("10");
```

This promise body has several attributes.

`handle` It is essential to give measurement promises handles, as these are used to label the log files that will store the values.

`stream_type`
Tells us that we are reading from what the system considered to be a regular file on the file-system.

`data_type` This says that data are to be treated as text with no other meaning.

`history_type`
This tells us that we want to log the values with a time stamp.

`units` This string is used in documentation to explain the measurement units of this result.

`match_value`
This is a body reference that represents the algorithm by which we extract data from the file.

`action` This is the generic action parameter that may be added to all promises. We use it here to limit the sample rate of this promise; `cf-monitord` samples by default at a rate of once per 2.5 minutes.

The matching body uses a method for selecting the correct line, and a way for extracting a pattern from the line. In every case the value extracted is described by using a regular expression *back-reference*, i.e. a parenthesized expression within a regular expression. The expression should match the entire line and should contain exactly one parenthesis.

```
body match_value find_blonks
{
select_line_number => "2";
extraction_regex => "Blonk blonk ([blonk]+).*";
}
```

The sampling rate is controlled by using the generic `action` constraint.

```
body action sample_min(x)
{
ifelapsed => "$(x)";
expireafter => "$(x)";
}
```

5.3.2 Extracting one-off numerical data

In this example we extract an integer value from an existing file. Notice that CFEngine samples the process table during `processes` promises so you might be able to save a new execution of a shell command and use the cached data, depending on your need for immediacy. It is always good practice to limit the system load incurred by monitoring.

```
  # Test 2 - follow a special process over time
  # using CFEngine's process cache to avoid resampling

   "/var/cfengine/state/cf_rootprocs"

      handle => "monitor_self_watch",
      stream_type => "file",
      data_type => "int",
      history_type => "static",
      units => "kB",
      match_value => proc_value(".*cf-monitord.*",

         "root\s+[0-9.]+\s+[0-9.]+\s+[0-9.]+\s+[0-9.]+\s+([0-9]+).*");
```

This match body selects a line matching a particular regular expression and extracts the 6th column of the process table. The regular expression skips first the root string and then five numerical values. The value is extracted into a one-off value

```
body match_value proc_value(x,y)
{
select_line_matching => "$(x)";
extraction_regex => "$(y)";
}
```

5.3.3 Extraction to list variable

In this example we discover a list of disks attached to the system.

```
  # Test 3, discover disk device information

  "/bin/df"

      stream_type => "pipe",
      data_type => "slist",
      history_type => "static",
      units => "device",
      match_value => file_system,
      action => sample_min("480"); # this is not changing much!


body match_value file_system
{
select_line_matching => "/.*";
extraction_regex => "(.*)";
}
```

5.4 Uses for custom monitoring

Unlike most other monitoring tools that use heavy-weight scripting languages to exact data, often running many processes for each measurement, CFEngine is a lightweight probe, using file interfaces and regular expressions to extract data. Thus its impact on the system is minimal. The possibilities for using this are therefore extremely broad:

- Extracting accounting data from systems for charge-back. This could be useful in cloud scenarios.
- Discovering memory leaks.
- Looking for zombie processes relating to specific software.
- Logging up-time.
- System class-dependent discovery and extraction of any kind of text for insertion into a CMDB.

6 Database Promises

CFEngine Nova can interact with commonly used database servers to keep promises about the structure and content of data within them.

CFEngine is, of course, not the tool of choice for performing complex database transactions, but it is an appropriate choice of tool for managing policy about the existence and structure of databases and correcting discrepancies.

6.1 How to manage databases

There are two main cases of database management to address: small embedded databases and large centralized databases.

CFEngine is a tool whose strength lies distributed management of computers. Databases are often centralized entities that have single point of management, so a large monolithic database is more easily managed with other tools. However, CFEngine can still monitor changes and discrepancies, and it can manage smaller embedded databases that are distributed in nature, whether they are SQL, registry or future types.

So creating 100 new databases for test purposes is a task for CFEngine, but adding a new item to an important production database is not a task that we recommend using CFEngine for.

There are three kinds of database supported by Nova:

LDAP - The Lightweight Directory Access Protocol
A hierarchical network database primarily for reading simple schema.

SQL - Structured Query Language
A number of relational databases (currently supported: MySQL, Postgres) for reading and writing complex data.

Registry - Microsoft Registry
An embedded database for interfacing with system values in Microsoft Windows.

In addition, CFEngine uses a variety of embedded databases for its own internals.

6.2 Database access rights

CFEngine's ability to make promises about databases depends on the good grace of the database server. Embedded databases are directly part of the system and promises can be made directly. However, databases running through a server process (either on the same host or on a different host) are independent agents and CFEngine cannot make promises on their behalf, unless they promise (grant) permission for CFEngine to make the changes. Thus the pre-requisite for making SQL database promises is to grant a point of access on the server.

6.3 Creating a point of contact on a server

The default behavior is for databases to deny connection access to external parties. However, it is possible to create a connection database that can be used as an anchor point. Once CFEngine is connected to any database, it can create new ones.

6.4 Creating SQL databases

Although the Structured Query Language (SQL) is a standard, database servers diverge wildly in their approaches to interaction. CFEngine currently supports two open source database implementations: MySQL and PostgreSQL and these two could not be more different.

Although it would be desirable to perform operations like create and destroy databases from within CFEngine, these operations can be difficult to set up due to security design of the database servers. This is where CFEngine's idea of *voluntary cooperation* helps us to understand why. The database service (even running on localhost) is an independent entity that we must interact with through its service portal. It promises nothing a priori, so we must arrange for the necessary promises to be kept so that the database will respond to CFEngine's manipulations. In other words, we have to grant access.

6.4.1 Creating a database manually

```
body common control
{
bundlesequence => { "databases" };
}

bundle agent databases

{
# Create database first - for postgres we could use a command

commands:

  "/usr/bin/createdb cf_topic_maps",

        contain => as_user("postgres");

# Create a table in the new database

databases:

  "cf_topic_maps/topics"

    database_operation => "create",
    database_type => "sql",
    database_columns => {
                        "topic_name,varchar,256",
                        "topic_comment,varchar,1024",
                        "topic_id,varchar,256",
                        "topic_type,varchar,256",
                        "topic_extra,varchar,26"
                        },

    database_server => myserver;
}

################################################

body database_server myserver
{
db_server_owner => "postgres";
db_server_password => "";
db_server_host => "localhost";
db_server_type => "postgres";
}

#

body contain as_user(x)
{
exec_owner => "$(x)";
}
```

6.4.2 Creating a database directly

Both MySQL and PostgreSQL provide default databases of the same name to connect to as an anchor point, allowing further databases to be created thereafter. In fact any database will do, you must just make sure that CFEngine has access rights to connect to the database.

```
body common control
{
bundlesequence => { "databases" };
}

bundle agent databases

{
databases:

  # Create table topics in database cf_topic_maps

  "cf_topic_maps"

    database_operation => "create",
    database_type => "sql",
    database_server => myserver("mysql");

}
```

```
################################################

body database_server myserver(x)
{
db_server_owner => "$(x)";
db_server_password => "";
db_server_host => "localhost";
db_server_type => "$(mysql)";
db_server_connection_db => "$(x)";
}

body contain as_user(x)
{
exec_owner => "$(x)";
}
```

6.5 Database table promises

To refer to tables in an SQL database, you treat the tables as if they were sub-directories of the database. In this example, we create one of the topic map tables for the knowledge base.

Table existence is determined by the `database_operation` attribute. We specify columns in a table as doublets or triplets with SQL type-names. In this example CFEngine will assume the promise below to be a complete specification of the table. Thus

- It will create non-existing columns (in no guaranteed order).
- It will report about columns that exist but are not promised, i.e. indicating that the table has been edited inappropriately.
- It will verify that the existing column names match their specification.

```
bundle agent databases

{
databases:

  "cf_topic_maps/topics"

    database_operation => "create",
    database_type => "sql",
    database_columns => {
                        "topic_name,varchar,256",
                        "topic_comment,varchar,1024",
                        "topic_id,varchar,256",
                        "topic_type,varchar,256",
                        "topic_extra,varchar,26"
                        },

    database_server => myserver;
}

################################################

body database_server myserver
{
any::
 db_server_owner => "postgres";
 db_server_password => "";
 db_server_host => "localhost";
 db_server_type => "postgres";
 db_server_connection_db => "postgres";
none::
 db_server_owner => "root";
 db_server_password => "";
 db_server_host => "localhost";
 db_server_type => "mysql";
 db_server_connection_db => "mysql";
}

body contain as_user(x)
{
exec_owner => "$(x)";
}
```

6.6 MS Registry functions

Another important database for Windows systems is the MS registry. The registry looks like a file-system, with files inside directories. in MS nomenclature, the registry consists of *(value,data)* pairs inside "keys" which are like directories. Keys may contain sub-keys in a hierarchical manner.

Managing data in the registry is a key part of Windows administration.

- Important system settings are located in and hence controlled by the registry.
- Security and compliance guidelines often specify values for the registry.
- If registry data become corrupted, software can cease to function. Compliance with government standards such as the U.S. FDCC regulations involves registry compliance checking.
- Automated repair (self-healing) of Windows machines can involve registry repair.

CFEngine Nova has functions for

- Creating and deleting keys, and *(value,data)* pairs.
- Scanning values and performing change detection.
- Restore or repair damaged registry data its cache in real-time.

6.6.1 Creating a registry key

```
body common control
```

```
{
bundlesequence => { "registry" };
}

bundle agent registry

{
databases:

 windows::

  "HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS"

    database_operation => "create",
    database_type => "ms_registry";

}
```

In this example we create a sub-key called 'CFEngine AS' inside the software key. Directory-like notation is used, as in the Windows registry editor. To create a sub-key of this new key, we simply add another link:

```
bundle agent registry
{
databases:

 windows::

  "HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS\CFEngine"

    database_operation => "create",
    database_type => "ms_registry";

}
```

6.6.2 Creating a value-data pair

```
body common control
{
bundlesequence => { "registry" };
}

bundle agent registry

{
databases:
 windows::

  "HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS\CFEngine"

    database_operation => "create",
    database_rows => { "value1,REG_SZ,new value 1", "value2,REG_SZ,new val 2"} ,
    database_type => "ms_registry";

}
```

Value-data pairs are in-fact value-type-data triplets, though the most common registry type is REG_SZ. Users are referred to the Microsoft Developer Network documentation on the registry for details (search for 'msdn registry').

6.6.3 Deleting registry keys

The `drop` or `delete` option is used to delete keys from a database. CFEngine does not (presently) perform recursive deletions of keys. It is debatable whether this feature would be potentially too destructive an operation to make simple.

In order to be allowed to delete a key or database, you must define a comment. This acts as a confirmation of intent to avoid accidents.

```
  "HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS\CFEngine"

    database_operation => "delete",
    database_type => "ms_registry";
```

6.6.4 Deleting registry values

If a registry deletion contains `columns`, these can be selected for deletion. It is not possible to delete rows in the registry (which corresponds to the just the data in a value-data pair.

```
  "HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS\CFEngine"

    database_operation => "delete",
    database_columns => { "value1", "value2" } ,
    database_type => "ms_registry";
```

Note that, unlike the case of adding values, there should be no comma separated doublets in this list of values, as one cannot delete by data, only by value.

6.6.5 Scanning and restoring the registry

CFEngine has the ability to scan a portion of the registry and cache its values for change detection and even later restoration. Since this is a resource consuming operation it should be performed only rarely, i.e. one should limit the policy to scan to special occasions.

The key question to ask is: when do I expect the current state of the registry to be a proper and true reflection of the 'correct state' of the system. This is not a trivial question to answer and it should be given careful thought before any talk of restoring the state of the registry based on such a scan.

Some values in the registry change often and one would like to ignore them entirely. The `registry_exclude` list allows you to make a list of regular expressions to match keys and value-names which are neither scanned nor restored.

```
bundle agent registry

{
databases:

 windows:: # add a time qualifier to this class

 "HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS\CFEngine"

    database_operation => "cache",   # cache,restore

    registry_exclude => {
                         ".*Windows.*CurrentVersion.*",
                         ".*Touchpad.*",
                         ".*Capabilities.FileAssociations.*",
                         ".*Rfc1766.*" ,
                         ".*Synaptics.SynTP.*",
                         ".*SupportedDevices.*8086",
                         ".*Microsoft.*ErrorThresholds"
                         },

    database_type     => "ms_registry";
```

Once one believes and trusts in a scan of the registry as a matter of policy, the values can be compared to this golden standard and restored if in error with the following kind of promise:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\CFEngine AS"

   database_operation => "restore",
   database_type      => "ms_registry";
```

Deletion or alteration of a key or value will now be swiftly repaired without the need to reboot the system.

6.7 LDAP integration

The Lightweight Directory Access Protocol (LDAP) is the basis of enterprise data centralization in many organizations using a variety of products from multiple vendors, including the MS Active Directory. CFEngine Nova supports special inbuilt functions for extracting data from LDAP repositories. The functions follow the general pattern of Open LDAP search functions.

6.7.1 Function `ldapvalue`

```
(string) ldapvalue(uri,dn,filter,name,scope,security)
```

This function retrieves a single field from a single LDAP record identified by the search parameters. The first matching value it taken.

ARGUMENTS:

'`uri`'	String value of the ldap server. e.g. `"ldap://ldap.CFEngine.com.no"`
'`dn`'	Distinguished name, an ldap formatted name built from components, e.g. "dc=CFEngine,dc=com".
'`filter`'	String filter criterion, in ldap search, e.g. "(sn=User)".
'`name`'	String value, the name of a single record to be retrieved, e.g. `uid`.
'`scope`'	Menu option, the type of ldap search, from the specified root. May take values: `subtree` `onelevel` `base`
'`security`'	Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities. `none` `ssl` `sasl`

6.7.2 Function `ldaplist`

```
(slist) ldaplist(uri,dn,filter,name,scope,security)
```

This function retrieves a single field from all matching LDAP records identified by the search parameters.

ARGUMENTS:

'`uri`'	String value of the ldap server. e.g. `"ldap://ldap.CFEngine.com.no"`
'`dn`'	Distinguished name, an ldap formatted name built from components, e.g. "dc=CFEngine,dc=com".
'`filter`'	String filter criterion, in ldap search, e.g. "(sn=User)".
'`name`'	String value, the name of a single record to be retrieved, e.g. `uid`.
'`scope`'	Menu option, the type of ldap search, from the specified root. May take values: `subtree` `onelevel` `base`
'`security`'	Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities.

```
none
ssl
sasl
```

6.7.3 Function `ldaparray`

```
(class) ldaparray (array,uri,dn,filter,scope,security)
```

This function retrieves an entire record with all elements and populates an associative array with the entries. It returns a class which is true if there was a match for the search and false if nothing was retrieved.

ARGUMENTS:

‘`array`’	String name of the array to populate with the result of the search
‘`uri`’	String value of the ldap server. e.g. "`ldap://ldap.CFEngine.com.no`"
‘`dn`’	Distinguished name, an ldap formatted name built from components, e.g. "dc=CFEngine,dc=com".
‘`filter`’	String filter criterion, in ldap search, e.g. "(sn=User)".
‘`scope`’	Menu option, the type of ldap search, from `subtree` `onelevel` `base`
‘`security`’	Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilites. `none` `ssl` `sasl`

6.7.4 Function `regldap`

```
(class) regldap(uri,dn,filter,name,scope,regex,security)
```

This function retrieves a single field from all matching LDAP records identified by the search parameters and compares it to a regular expression. If there is a match, true is returned else false.

ARGUMENTS:

‘`uri`’	String value of the ldap server. e.g. "`ldap://ldap.CFEngine.com.no`"
‘`dn`’	Distinguished name, an ldap formatted name built from components, e.g. "dc=CFEngine,dc=com".
‘`filter`’	String filter criterion, in ldap search, e.g. "(sn=User)".
‘`name`’	String value, the name of a single record to be retrieved, e.g. `uid`.
‘`scope`’	Menu option, the type of ldap search, from the specified root. May take values:

```
subtree
onelevel
base
```

‘regex’ A regular expression string to match to the results of an LDAP search. If any item matches, the result will be true.

‘security’ Menu option indicating the encryption and authentication settings for communication with the LDAP server. These features might be subject to machine and server capabilities.

```
none
ssl
sasl
```

6.7.5 LDAP function examples

```
# LDAP example function usage

body common control
{
bundlesequence => { "ldap" };
}

bundle agent ldapbundle
{
vars:

   # Get the first matching value for "uid"

  "value" string => ldapvalue(
                             "ldap://ldap.CFEngine.com.no",
                             "dc=CFEngine,dc=com",
                             "(sn=User)",
                             "uid",
                             "subtree",
                             "none"
                             );

   # Get a all matching values for "uid"

  "list" slist =>  ldaplist(
                           "ldap://ldap.CFEngine.com.no",
                           "dc=CFEngine,dc=com",
                           "(sn=User)",
                           "uid",
                           "subtree",
                           "none"
                           );

classes:
   # Get first matching value with all fields

   "gotdata" expression => ldaparray(
                                    "myarray",
                                    "ldap://ldap.CFEngine.com.no",
                                    "dc=CFEngine,dc=com",
                                    "(sn=User)",
                                    "subtree",
                                    "none"
                                    );

   # Look for a regex match within the ldap match

   "found" expression => regldap(
                                "ldap://ldap.CFEngine.com.no",
                                "dc=CFEngine,dc=com",
                                "(sn=User)",
                                "uid",
                                "subtree",
                                "jon.*",
                                "none");


reports:

 linux::

   "LDAP VALUE $(value) found";
   "LDAP LIST VALUE $(list)";

 gotdata::

   "found data $(ldapbundle.myarray[uid])";

  found::

    "Matched regex";

}
```

6.7.6 Best practice for LDAP integration

Relying on LDAP or other network data without validation is a potentially dangerous policy. One could destroy a system by assuming that the service will respond with a sensible result. CFEngine does not recommend reliance on any network services in a configuration policy.

The failure to find an item in LDAP does not imply its non-existence. Policies should work opportunistically. If something is found, it may be used. If something is not found, it is difficult to conclude the reason for this.

If users are removed from a system, a best practice would be to add their names to a list of users who should not exist, rather than assuming that the failure of a search for the user implies his/her absence from the system.

7 File Access Control Lists

7.1 Introduction

Access Control Lists (ACL) allow for a more fine-grained access control on file system objects than standard Unix permissions. In spite of the success of the POSIX model's simplicity the functionality is limited.

File permission security is a subtle topic. Users should take care when experimenting with ACLs as the results can often be counter-intuitive. In some cases the functioning of a system can be compromised by changes of access rights.

Not all file systems support ACLs. In Unix systems there is a plethora of different file system types, which have different models of ACLs. Be aware that the mount-options for a file-system may affect the ACL semantics.

Note that when adding a user to a group, this will not have any effect until the next time the user logs in on many operating systems.

As CFEngine works across multiple platforms and needs to support ACLs with different APIs, a common ACL syntax has been abstracted to add a layer of portability. This is a specific feature of CFEngine, not of the host systems. A generic syntax ensures that the ACLs that are commonly needed can be coded in a portable fashion. CFEngine Nova's ACL model is translated into native permissions for implementation; CFEngine does not interfere with native access mechanisms in any way. The CFEngine ACL syntax is similar to the POSIX ACL syntax, which is supported by BSD, Linux, HP-UX and Solaris.

CFEngine also allows you to specify platform-dependent ACLs. Of course, these ACLs will only work on the given platform, and must therefore be shielded with classes that select the appropriate model within the promise body.

Currently, CFEngine Nova supports the following ACL APIs and operating systems.

ACL type	**Operating system**
NTFS	Windows Server 2003, 2008
POSIX	Linux

7.2 File ACL example

The form of a CFEngine files promise that uses ACLs is as follows:

```
#
# test_acl.cf
#

body common control
{
bundlesequence => { "acls" };
}
```

```
#########################################

bundle agent acls

{
files:

  "/office/shared"

    acl => template;
}

#########################################

body acl template

{
acl_method => "overwrite";
acl_directory_inherit => "parent";

 linux|solaris::

  acl_type => "posix";

  aces => {
          "user:*:rw",
          "user:root:rw",
          "group:*:r",
          "mask:rwx",
          "all:r"
          };

 windows::

  acl_type => "ntfs";

  aces => {
          "user:Administrator:rw(po)",
          "all:r"
          };
}
```

7.2.1 Concepts

As mentioned, there are many different ACL APIs. For example, the POSIX draft standard, NTFS and NFSv4 have different and incompatible ACL APIs. As CFEngine is cross-platform, these differences should, for the most usual cases, be transparent to the user. However, some distinctions are impossible to make transparent, and thus the user needs to know about them.

We will explore the different concepts of ACL implementations that are critical to understanding how permissions are defined and enforced in the different file systems. As a running example, we will consider NTFS ACLs and POSIX ACLs, because the distinction between these ACL APIs is strong.

7.2.2 Entity types

All ACL APIs support three basic entity types: user, group and all. User and group are simply users and groups of the system, where a group may contain multiple users. all is all users of the system, this type is called "other" in POSIX and "Everyone" in NTFS.

7.2.3 Owners

All file system objects have an owner, which by default is the entity that created the object. The owner can always be a user. However, in some file systems, groups can also be owners (e.g. the "Administrators" group in NTFS). In some ACL APIs (e.g POSIX), permissions can be set explicitly for the owner, i.e. the owner has an independent ACL entry.

7.2.4 Changing owner

It is generally not possible for user A to set user B as the owner of a file system object, even if A owns the object. The superuser ("root" in POSIX, "Administrator" in NTFS) can however always set itself as the owner of an object. In POSIX, the superuser may in fact set any user as the owner, but it NTFS it can only take ownership.

7.2.5 Permissions

An entity can be given a set of permissions on a file system object (e.g. read and write). The data structure holding the permissions for one entity is called an "Access Control Entry" (ACE). As many users and groups may have sets of permissions on a given object, multiple Access Control Entries are combined to an Access Control List, which is associated with the object.

The set of available permissions differ with ACL APIs. For example, the "Take Ownership" permission in NTFS has no equivalent in POSIX. However, for the most common situations, it is possible to get equivalent security properties by mapping a set of permissions in one API to another set in a second API.

There are however different rules for the access to the contents of a directory with no access. In POSIX, no sub-objects of a directory with no access can be accessed. However, in NTFS, sub-objects that the entity has access rights to can be accessed, regardless of whether the entity has access rights to the containing directory.

7.2.6 Deny permissions

If no permissions are given to a particular entity, the entity will be denied any access to the object. But in some file systems, like NTFS, it is also possible to explicitly deny specific permissions to entities. Thus, two types of permissions exist in these systems: allow and deny.

It is generally good practice to design the ACLs to specify who is allowed to do some operations, in contrary to who is not allowed to do some operations, if possible. The reason for this is that describing who is not allowed to do things tend to lead to more complex rules and could therefore more easily lead to mis-configurations and security holes. A good rule is to only define that users should not be able to access a resource in the following two scenarios:

- Denying access to a subset of a group which is allowed access
- Denying a specific permission when a user or a group has full access

If you think about it, this is the same principle that applies to firewall configuration: it is easier to white-list, specify who should have access, than to blacklist, specify who should not have access. In addition, since CFEngine is designed to be cross-platform and some ACL permissions are not available on all platforms, we should strive to keep the ACLs as simple as possible. This helps us avoid surprises when the ACLs are enforced by different platforms.

7.2.7 Changing permissions

Generally, only the owner may change permissions on a file system object. However, superusers can also indirectly change permissions by taking ownership first. In POSIX, superusers can change permissions without taking ownership. In NTFS, either ownership or a special permission ("Change Permissions") is needed to alter permissions.

7.2.8 Effective permissions

Unfortunately, even though two ACL APIs support all the permissions listed in an ACL, the ACL may be interpreted differently. For a given entity and object with ACL, there are two conceptually different ways to interpret which permissions the entity obtains: ACE precedence and cumulative ACL.

For example, let '`alice`' be a user of the group '`staff`'. There is an ACL on the file '`schedule`', giving '`alice`' write permission, and the group '`staff`' read permission. We will consider two ways to determine the effective permissions of '`alice`' to '`schedule`'.

Firstly, by taking the most precise match in the ACL, '`alice`' will be granted write permission only. This is because an ACE describing '`alice`' is more precise than an ACE describing a group '`alice`' is member of. However, note that some ACEs may have the same precedence, like two ACEs describing permissions for groups '`alice`' is member of. Then, cumulative matching will be done on these ACEs (explained next). This is how POSIX does it.

Secondly, we can take the cumulative permissions, which yields a user permissions from all the ACE entries with his user name, groups he is member of or the ACE entry specifying all users. In this case, '`alice`' would get read and write on '`schedule`'. NTFS computes the effective permissions in this way.

7.2.9 Inheritance

Directories have ACLs associated with them, but they also have the ability to inherit an ACL to sub-objects created within them. POSIX calls the former ACL type "access ACL" and the latter "default ACL", and we will use the same terminology.

7.3 CFEngine 3 Generic ACL Syntax

The CFEngine 3 ACL syntax is divided into two main parts, a generic and an API specific (native). The generic syntax can be used on any file system for which CFEngine supports ACLs, while the native syntax gives access to all the permissions available under a particular ACL API.

An ACL can contain both generic and native syntax. However, if it contains native syntax, it can only be enforced on systems supporting the given ACL API. Thus, only the generic syntax is portable.

Note that even though the same generic ACL is set on two systems with different ACL APIs, it may be enforced differently because the ACE matching algorithms differ. For instance, as discussed earlier, NTFS uses cumulative matching, while POSIX uses precedence matching. CFEngine cannot alter the matching algorithms, and simulating one or the other by changing ACL definitions is not possible in all cases, and would probably lead to confusion. Thus, if an ACL is to be used on two systems with different ACL APIs, the user is encouraged to check that any differences in matching algorithms do not lead to mis-configurations.

The CFEngine generic ACL syntax explained next, and native syntax is described in following sections.

```
body acl acl_alias:
{
acl_type               => "generic"/"ntfs"/"posix";
acl_method             => "append"/"overwrite";
acl_directory_inherit => "nochange"/"clear"/"parent"/"specify";
aces                   => {
                           "user:uid:mode[:perm_type]", ...,
                           "group:gid:mode[:perm_type]", ...,
                           "all:mode[:perm_type]"
                          };
specify_inherit_aces  => {
                           "user:uid:mode[:perm_type]", ...,
                           "group:gid:mode[:perm_type]", ...,
                           "all:mode[:perm_type]"
                          };
}
```

- `acl_alias` is the name of the specified ACL. It can be any identifier containing alphanumeric characters and underscores. We will use this name when referring to the ACL.
- `acl_type` (optional) specifies the ACL API used to describe the ACL. It defaults to `generic`, which allows only CFEngine generic ACL syntax, but is valid on all supported systems. `acl_type` only needs to be specified if native permissions are being used in the ACL (see `nperms` below). If `acl_type` is set to anything other than `generic`, the system on which it is enforced must support this ACL API.
- `acl_method` (optional) can be set to either `append` or `overwrite`, and defaults to `append`. Setting it to `append` only adds or modifies the ACEs that are specified in `aces` (and `specify_inherit_aces`, see below). If set to `overwrite`, the specified ACL will completely replace the currently set ACL. All required fields must then be set in the specified ACL (e.g. `all` in POSIX), see the following sections describing the supported native APIs.
- `acl_directory_inherit` (optional) specifies the ACL of newly created sub-objects. Only valid if the ACL is set on a directory. On directories, `nochange` is the default and indicates that the ACL that is currently given to newly created child objects is left unchanged. If set to `clear`, no ACL will be inherited, but the file system specifies a default ACL, which varies with the file system (see the following sections on the supported ACL APIs). `parent` indicates that the ACL set in `aces` (see below) should be inherited to sub-objects. If set to `specify`, `specify_inherit_aces` specifies the inherited ACL, and `acl_method` applies for `specify_inherit_aces` too.
- `aces` is a list of access control entries. It is parsed from left to right, and multiple entries with the same entity-type and id is allowed. This is necessary to specify permissions with different `perm_type` for the same entity (e.g. to allow read permission but explicitly deny write).
- `specify_inherit_aces` (optional) is a list of access control entries that are set on child objects. It is also parsed from left to right and allows multiple entries with same entity-type and id. Only valid if `acl_directory_inherit` is set to `specify`.

- `user` indicates that the line applies to a user specified by the user identifier `uid`. `mode` is the permission mode string.
- `group` indicates that the line applies to a group specified by the group identifier `gid`. `mode` is the permission mode string.
- `all` indicates that the line applies to every user. `mode` is the permission mode string.
- `uid` is a valid user name for the system and cannot be empty. However, if `acl_type` is `posix`, `uid` can be set to * to indicate the user that owns the file system object.
- `gid` is a valid group name for the system and cannot be empty. However, if `acl_type` is `posix`, `gid` can be set to * to indicate the file group.
- `mode` is one or more strings `op|gperms|(nperms)`; a concatenation of `op`, `gperms` and optionally (`nperms`), see below, separated with commas (e.g. +rx,-w(s)). `mode` is parsed from left to right.
- `op` specifies the operation on any existing permissions, if the specified ACE already exists. `op` can be =, empty, + or -. = or empty sets the permissions to the ACE as stated, + adds and - removes the permissions from any existing ACE.
- `nperms` (optional) specifies ACL API specific (native) permissions. Only valid if `acl_type` is not `generic`. Valid values for `nperms` varies with different ACL types, and are specified in subsequent sections.
- `perm_type` (optional) can be set to either `allow` or `deny`, and defaults to `allow`. `deny` is only valid if `acl_type` is set to an ACL type that support deny permissions.
- `gperms` (generic permissions) is a concatenation of zero or more of the characters shown in the table below. If left empty, none of the permissions are set.

Flag	**Description**	**Semantics on file**	**Semantics on directory**
`r`	Read	Read data, permissions, attributes	Read directory contents, permissions, attributes
`w`	Write	Write data	Create, delete, rename sub-objects
`x`	Execute	Execute file	Access sub-objects

Note that the `r` permission is not necessary to read an object's permissions and attributes in all file systems (e.g. in POSIX, having `x` on its containing directory is sufficient).

7.3.1 Generic syntax examples

```
body common control
{
bundlesequence => { "acls" };
}

bundle agent acls
{
files:
  "/office/schedule"
    acl => small;

  "/office/audit_dir"
```

```
    acl => dirinherit;
}

body acl small
{
aces => {"user:alice:w", "group:staff:r", "all:"};
}

body acl dirinherit
{
acl_directory_inherit => "parent";
aces => {"user:alice:+w,-x", "user:bob:+r,-w", "group:staff:=rx", "all:-w"};
}
```

See the following sections on native ACL types for more examples.

7.4 POSIX ACL type

7.4.1 POSIX-specific ACL syntax

Native permissions The valid values for `nperms` in POSIX are `r`,`w`, and `x`. These are in fact the same as the generic permissions, so specifying them as generic or native gives the same effect.

File owner and group A user-ACE with uid set to * indicates file object owner. A group-ACE with gid set to * indicates file group.

mask mask can be specified as a normal ACE, as `mask:mode`. mask specifies the maximum permissions that are granted to named users (not owning user), file group and named groups. mask is optional, if it is left unspecified it will will be computed as the union of the permissions granted to named users, file group and named groups (see acl_calc_mask(3)).

Required ACEs POSIX requires existence of an ACE for the file owner, (`user:*:mode`), the file group (`group:*:mode`), other (`all:mode`) and mask (`mask:mode`). As mentioned, CFEngine automatically creates a mask-ACE, if missing. However, if `method` is set to `overwrite`, the user must ensure that the rest of the required entries are specified.

7.4.2 Generic syntax mapping

Entity types All entity types in the generic syntax are mapped to the corresponding entity types with the same name in POSIX, except `all` which corresponds to `other` in POSIX.

Generic permissions

As shown in the table below, `gperms` is mapped straightforward from generic to POSIX permission flags.

Generic flag	**POSIX flag**
`r`	`r`
`w`	`w`
`x`	`x`

Inheritance POSIX supports `acl_directory_inherit` set to `specify`. The `specify_inherit_aces` list is then set as the default ACL in POSIX (see acl(5)).

If `acl_directory_inherit` is set to `parent`, CFEngine copies the access ACL to the default ACL. Thus, newly created objects get the same access ACL as the containing directory.

`acl_directory_inherit` set to `clear` corresponds to no POSIX default ACL. This results in that newly created objects get ACEs for owning user, group and other. The permissions are set in accordance with the mode parameter to the creating function and the umask (usually results in 644 for files and 755 for directories).

Further reading The manual page `acl(5)` contains much information on POSIX ACLs, including the access check algorithm. In particular, this shows that POSIX uses ACE precedence matching, and exactly how it is done. Operating systems usually bundle tools for manipulating ACLs, for example `getfacl(1)` and `setfacl(1)`.

7.4.3 POSIX ACL examples

```
body common control
{
bundlesequence => { "acls" };
}

bundle agent acls
{
files:

  "/office/timetable"
    acl => nativeperms;

  "/office/user_dir"
    acl => specifyinherit;
}

body acl nativeperms
{
acl_type => "posix";
aces => {"user:alice:r(w)", "user:root:=(rwx)",
         "group:staff:-r(x)", "all:-(w)", "mask:(rx)"};
}

body acl specifyinherit
{
acl_type => "posix";
acl_method => "overwrite";
acl_directory_inherit => "specify";
aces => {"user:*:rwx", "group:*:rx", "user:alice:rwx",
         "user:root:rx", "group:staff:r", "all:rx"};
specify_inherit_aces => {"user:*:", "group:*:", "all:"};
}
```

7.5 NT ACL type

7.5.1 NT-specific ACL syntax

Native permissions NTFS supports fourteen so-called special file permissions. However, we do not consider the `Synchronize` permission because it is used for a different purpose than the other permissions. In order to give access to the thirteen relevant permissions, CFEngine defines a native permission flag for each of them. This one-to-one mapping is as follows.

NTFS Special Permission	CFEngine nperm
Execute File / Traverse Folder	x
Read Data / List Folder	r
Read Attributes	t
Read Extended Attributes	T
Write Data / Create Files	w
Append Data / Create Folders	a
Write Attributes	b
Write Extended Attributes	B
Delete Sub-folders and Files	D
Delete	d
Read Permissions	p
Change Permissions	c
Take Ownership	o

The semantics of these special permissions can be found in the references for further reading below.

Denying permissions NTFS supports setting `perm_type` to `deny` in addition to `allow`, which is the default. This can for instance be used to denying a user a particular permission that a group membership grants him. It is important to note that the precedence of allow and deny permissions is as follows:

1. Explicit Deny
2. Explicit Allow
3. Inherited Deny from parent
4. Inherited Allow from parent
5. Inherited Deny from grandparent
6. Inherited Allow from grandparent
7. ...

Thus, the closer the permission is to the object in the directory path, the greater precedence it is given.

An important point here is that even though a user is denied access in a parent directory and this permission is inherited, but one of the groups he is member of is explicitly allowed access to a file in that directory, he is actually allowed to access the file.

Ownership In NTFS, the default owner is the user who is currently logged on. The only exceptions occur when the user is a member of either the '`Administrators`' group or the '`Domain Admins`' group.

Owners of NTFS objects can allow another user to take ownership by giving that user Take Ownership permission. Owners can also give other users the Change Permissions flag. In addition, members of the '`Administrator`' group can always take ownership. It is never possible to give ownership of an object to a user, but members of the '`Administrator`' group can give ownership to that group.

7.5.2 Generic syntax mapping

Entity types The three entity types of NTFS are called user, group and Everyone. The user and group entity types in NTFS are mapped to the user and group entity types in CFEngine. Everyone is mapped to all in CFEngine.

Generic permissions For NTFS, CFEngine maps the `gperms` to `nperms` as follows.

Generic flag	**Native flags**
`r`	`rtTp`
`w`	`wabB`
`x`	`x`

The rationale for this mapping is discussed next.

NTFS groups the thirteen special permissions to create five sets of permissions:

- Read
- Read & Execute
- Write
- Modify
- Full Control

In addition, we have the List Folder Contents set, which is equivalent to the Read & Execute set but is only available to- and inherited by directories. The Full Control set is unsurprisingly all the thirteen special permissions. An overview of the NTFS mapping of special permissions to sets is given in the references stated as further reading below. The NTFS permission sets can be expressed in CFEngine syntax as follows.

NTFS sets	**CFEngine gperms|(nperms)**
`Read`	`r`
`Write`	`w`
`Read & Execute`	`rx`
`Modify`	`rwx(d)`
`Full Control`	`rwx(dDco)`

Inheritance `acl_directory_inherit` set to `clear` disables inheritance, such that child objects get a default ACL specified by the operating system, namely Full control for the file object creator and `SYSTEM` accounts.

POSIX compatibility Be aware that setting `gperms` to '`rwx`' on directories is more restrictive in NTFS than in POSIX ACLs. This is because NTFS does not allow deletion of objects within a directory without a Delete Sub-folders and Files permission on the directory (or a Delete permission on the object itself), while in POSIX, '`rwx`' on the directory is sufficient to delete any file or directory within it (except when the sticky-flag is set on the directory). Thus, on directories, the NTFS-equivalent to POSIX `gperms` set to '`rwx`' is '`rwx(D)`'. However, for files, '`rwx`' is equivalent in POSIX and NTFS semantics.

In POSIX ACLs, there is no explicit delete permission, but the execute, write and sticky permissions on the containing directory determines if a user has privileges to delete. In POSIX, the owner and root can change permissions, while usually only the root may change the ownership, so there is no direct equivalent to the Change Permission and Take Ownership in POSIX.

Further reading A description of the fourteen NTFS permission and the mapping of these into sets is given at http://support.microsoft.com/kb/308419.

7.5.3 NT ACL examples

```
body common control
{
bundlesequence => { "acls" };
}

bundle agent acls
{
files:

  "C:\Program Files\Secret Program"
    acl => restrictive;

  "D:\Shared"
    acl => sharespace;
}

body acl restrictive
{
acl_type => "ntfs";
acl_method => "overwrite";
acl_directory_inherit => "parent";
aces => {"user:Administrator:r"};
}

body acl sharespace
{
acl_type => "ntfs";
acl_method => "overwrite";
acl_directory_inherit => "specify";
aces => { "user:Administrator:rwx(dDco)",
          "group:Hackers:rwx(dDco):deny",
          "all:rw" };
specify_inherit_aces => {"user:Administrator:r"};
}
```

8 Server extensions

CFEngine Nova adds a simple server extension to the Community Edition server, namely the ability to encode data directly in policy. This feature is useful for distributing password hashes to systems.

8.1 Server access resource type

By default, access to resources granted by the server are files. However, sometimes it is useful to cache `literal` strings, hints and data in the server, e.g. the contents of variables, hashed passwords etc for easy access. In the case of literal data, the promise handle serves as the reference identifier for queries. Queries are instigated by function calls by any agent.

```
access:

  "This is a string with a $(localvar) for remote collection"

        handle  => "test_scalar",
  resource_type => "literal",
        admit   => { "127.0.0.1" };
```

The promise looks exactly like a promise for a file object, but the data are literal and entered into the policy. This is a useful approach for distributing password hashes on a need-to-know basis from a central location. The server configuration file need not be distributed to any client, thus only authorized systems will have access to the hashes.

8.2 Function `remotescalar`

The client side of the literal look up function is:

```
(string) remotescalar(resource handle,host/IP address,encrypt);
```

This function downloads a string from a remote server, using the promise handle as a variable identifier.

ARGUMENTS:

‘`resource handle`’
The name of the promise on the server side

‘`host or IP address`’
The location of the server on which the resource resides.

‘`encrypt`’ Whether to encrypt the connection to the server.

```
true
yes
false
no
```

8.3 Example remote scalar lookup

```
########################################################
#
# Remote value from server connection to cf-serverd
#
########################################################

body common control

{
bundlesequence  => { "testbundle"  };

version => "1.2.3";
}

########################################################

bundle agent testbundle

{
vars:

 "remote" string => remotescalar("test_scalar","127.0.0.1","yes");

reports:

  linux::

    "Receive value $(remote)";
}

#########################################################
# Server config
#########################################################

body server control

{
allowconnects         => { "127.0.0.1" , "::1" };
allowallconnects      => { "127.0.0.1" , "::1" };
trustkeysfrom         => { "127.0.0.1" , "::1" };
allowusers            => { "mark" };
}

#########################################################
```

```
bundle server access_rules()

{
vars:

  "localvar" string => "literal string";

access:

  "This is a $(localvar) for remote access"

        handle  => "test_scalar",
  resource_type => "literal",
        admit   => { "127.0.0.1" };
}
```

9 Environments and workflows

9.1 Environments in Nova

Nova supports notion of ‘environments’ – named groups of hosts, like ‘development’, 'QA' and 'production' hosts.

Each environment gets

- Separate subdirectory ‘environment_<NAME>’ for storing promises
- Separate subdirectory ‘environment_<NAME>/cdp_inputs’ for content-based policies
- grouping of hosts belonging to environment in GUI

Default Nova policies group hosts into environments in bundle ‘common environments’. This bundle defines

- ‘environments.active’ variable. This variable holds the name of environment for current host, and is used by other promises in Nova templates. This variable does not affect grouping in GUI.
- global ‘environment_<NAME>’ class. This class should be defined if given host belongs to the ‘<NAME>’ environment. GUI groups hosts using this class.

Administrator may customize this bundle as needed, adding or removing environments defined (Nova does not have any built-in environment names), changing rules for hosts selection and making any other modifications, the only requirement for environments feature to work is keeping aforementioned variables and classes defined. Note that the environment name 'any' is reserved.

Default rules expect file ‘environment_$(promises.active)/promises.cf’ to exist and to contain bundle ‘agent main’, this bundle is included last from main promises.cf and should implement/include/use any environment-specific rules.

9.2 Implementing workflows in Nova

Workflows are implemented easily with Nova, administrator starts with defining necessary set of environments (usually those are ‘development’, ‘testing’ and 'production'), then develops new configuration in one enviroment, and promotes rules first to testing and then to production environment.

Promotion of rules is performed by copying rules from one environment’s subdirectory to next one.

10 Virtualization

10.1 What are virtualization and cloud computing?

Virtualization refers to the ability to run multiple host instances on a single physical node. Cloud computing typically refers to what is called 'platform as a service', or deployment of virtual machines on demand, often as an on-line service.

In this document, virtualization support refers specifically to hypervisor technologies supported by the open source library layer *libvirt* project, which includes interfaces for Xen, KVM, Vmware-ESX, and more. CFEngine thus integrates freely with other tools based on this library, such as *virsh* and the *Virtual Manager* graphical user interface.

10.2 Why build virtualization support into CFEngine?

Virtualization engines (usually called supervisors or hypervisors) are seeing an explosion of development. They exist as a number of projects in various stages of maturity. The libvirt project was designed as an integration layer based on an XML specification.

The tools for management are still quite primitive and require much manual work. CFEngine has a unique role to play in maintaining desired state in virtual machine systems.

In the cloud, virtual machines may be rented from remote commercial providers, and managed as disposable resources. Convergent or 'self-healing' maintenance is an essential method for managing machines that are geographically remote and awkward to access, e.g. machines in other time-zones that it is impractical to monitor by legacy methods.

10.3 What can CFEngine do with virtual machines?

The simple answer is: anything that *libvirt* can do, with added convergence to a desired state: that means, creating, destroying and starting and stopping machines. By starting virtual machines through CFEngine, you can be sure that a given 'virtual guest' is running on one and only one physical host, thus avoiding conflicts that are difficult to detect with centralized systems.

CFEngine does not support everything that libvirt does – it offers a simplified interface that is meant for robustness, stability and hands-free repeatability.

CFEngine does not use libvirt's TLS based web communication layer. It manages every host as an independent entity, in typical CFEngine fashion, using CFEngine's own distributed cooperation to provide the implicit communication. CFEngine does not currently support so-called 'live migration' of virtual machines.

10.4 Guest environments promises

A virtual machine is one example of what CFEngine calls an ‘guest environment’. You can promise to create (and host) an environment with certain attributes, just as you can promise to host a file or a process. Here is a simple example:

```
body common control
{
bundlesequence  => { "my_vm_cloud" };
}

#######################################################

bundle agent my_vm_cloud
{
guest_environments:

   "myUbuntu" # the running instance name, defined in XML

       environment_resources => virt_xml,
       environment_type      => "xen",
       environment_host      => "my_physical_computer", # ipv4_10_1_2_3
       environment_state     => "create";
}

#######################################################

body environment_resources virt_xml
{
env_spec_file => "/srv/xen/centos5-libvirt-create.xml";
}
```

- The promiser (in this case ‘`myUbuntu`’) is the name of the virtual machine. This should be a unique identifier, as we need to be able to refer to machines uniquely.
- The guest environment host is the name of the computer that is the host for the virtual machine.
- Normally when we want to ensure something on a machine, we use classes to decide where the promise will be made. For guest environments, however, we need to make promises about the uniqueness of the machine. When you make a machine instance you normally want it to be running on one and only one host. So you want *every* machine to make a promise. On the guest environment's host, you want to promise that the guest environment is running, and on every other machine you want to promise that it is not. In CFEngine, you simply include a unique class belonging to host in the promise using `environment_host` and CFEngine assumes that rest. Unique classes might include
  - Hostname class e.g. `myhost_CFEngine_com`
  - IP address class e.g. `ipv4_123_456_789_123`

An alternative way to write this example is to quote the XML specification in CFEngine directly. This has a few advantages: you can re-use the data and use it as a template, filling in CFEngine-variables. You can thus adapt the configuration using CFEngine's classes.

```
bundle agent my_vm_cloud
{
guest_environments:

   "myUbuntu" # the running instance name, defined in XML
       environment_resources => virt_xml("$(this.promiser)"),
       environment_type      => "xen",
       environment_host      => "myphysicalcomputer";
       environment_state     => "create"
}

#######################################################

body environment_resources virt_xml(host)
{
env_spec_file =>

"<domain type='xen'>
  <name>$(host)</name>
  <os>
    <type>linux</type>
    <kernel>/var/lib/xen/install/vmlinuz-ubuntu10.4-x86_64</kernel>
    <initrd>/var/lib/xen/install/initrd-vmlinuz-ubuntu10.4-x86_64</initrd>
    <cmdline> kickstart=http://example.com/myguest.ks </cmdline>
  </os>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <devices>
    <disk type='file'>
      <source file='/var/lib/xen/images/$(host).img'/>
      <target dev='sda1'/>
    </disk>
    <interface type='bridge'>
      <source bridge='xenbr0'/>
      <mac address='aa:00:00:00:00:11'/>
      <script path='/etc/xen/scripts/vif-bridge'/>
    </interface>
    <graphics type='vnc' port='-1'/>
    <console tty='/dev/pts/5'/>
  </devices>
</domain>
";
}
```

You should consult the libvirt documentation for the details of the XML specification.

10.5 Virtualization types supported

CFEngine currently supports virtualization only through libvirt, so it supports those technologies that libvirt supports. Currently this includes most popular technologies. You must choose the type of monitor that is to be responsible for keeping the guest environment promise. In CFEngine, you should choose between a machine environment or network environment of the following types:

`xen`	A Xen hypervisor virtual domain.
`kvm`	A KVM hypervisor virtual domain.
`esx`	A VMware hypervisor virtual domain.
`test`	The libvirt test-hypervisor virtual domain.
`xen_net`	A Xen hypervisor virtual network.
`kvm_net`	A KVM hypervisor virtual network
`esx_net`	An ESX/VMWare hypervisor virtual network.
`test_net`	The test hypervisor virtual network.
`zone`	A Solaris zone (future development)
`ec2`	An Amazon EC2 instance (future development)
`eucalyptus`	A Eucalyptus instance (future development)

Once again, you must consult the libvirt documentation for details.

10.6 Distinct states

Libvirt recognizes a number of distinct states are transliterated into CFEngine as

`create`	Build and start an guest environment.
`delete`	Halt and remove runtime resources associated with an guest environment.
`running`	An existing guest environment is in a running state.
`suspended`	An existing guest environment is in a 'paused' state.
`down`	An existing guest environment is in a halted state.

The default promised state is for a machine to be running wherever the `environment_host` class is true, and suspended or down elsewhere.

10.7 Example deployment

Prerequisites: you need to make a 'disk image' for the machine, or a virtual disk of blocks that can be allocated. This image does not have to contain any data, it will simply as a block device for the VM. You can then install it by booting the machine from a network image, like a PXE/kickstart installation.

If you want to allocate disk blocks as the file grows, you can create a file with a hole. The following command will creates a file of 2048MB, but the actual data blocks are allocated in a lazy fashion:

```
# dd if=/dev/zero of=/srv/xen/my.img oflag=direct bs=1M seek=2047 count=1
```

To reserve all the data blocks right away:

```
# dd if=/dev/zero of=/srv/xen/my.img oflag=direct bs=1M count=2048
```

Libvirt uses an XML file format that cannot be circumvented. CFEngine promises to honor the promises that are expressed in this file, as in the examples above. You need to find out about this file format from the libvirt website. To get CFEngine to Honor these promises, you point it to the specification that it should promise using `spec_file`.

You need to set up a network for virtual machines to communicate with the outside world. This can also be done with CFEngine, using the network promise types to build a bridge into a virtual network.

Then just run CFEngine to start, stop or manage the guest environments on each localhost. Run in verbose mode to see how CFEngine maintains the states convergently.

```
# cf-agent -v
```

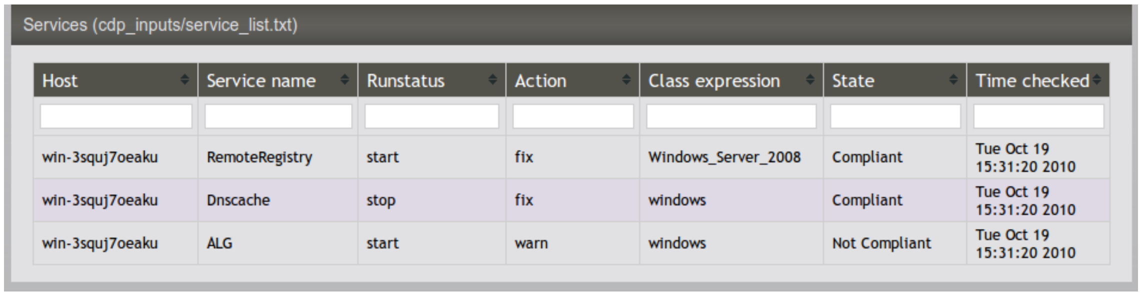
11 Content-Driven Policies

In Nova 2.0, Content-Driven Policies (CDP) were introduced to make policy management easier. In contrast to policies written in the CFEngine language, these are semi-colon separated fields in a text file that the user just fills with content. All the underlying enforcement and reporting are taken care of automatically by Nova.

For example, to manage three Windows services, the following services-CDP will suffice.

```
# masterfiles/cdp_inputs/service_list.txt

Dnscache;stop;fix;windows
ALG;start;warn;windows
RemoteRegistry;start;fix;Windows_Server_2008
```

The meanings of the fields are different depending of the CDP-type, but explained in the file header. With these three lines, we ensure the correct status of three services on all our Windows machines and are given specialized reports on the outcome. The Content-Driven Policy services report is shown below.



Services (cdp_inputs/service_list.txt)

Host	Service name	Runstatus	Action	Class expression	State	Time checked
win-3squj7oeaku	RemoteRegistry	start	fix	Windows_Server_2008	Compliant	Tue Oct 19 15:31:20 2010
win-3squj7oeaku	Dnscache	stop	fix	windows	Compliant	Tue Oct 19 15:31:20 2010
win-3squj7oeaku	ALG	start	warn	windows	Not Compliant	Tue Oct 19 15:31:20 2010

11.1 Benefits of Content-Driven Policies

As seen in the example above, Content-Driven Policies are easy to write and maintain, especially for users unfamiliar with the CFEngine language. They are designed to capture the essence of a specific, popular use of CFEngine, and make it easier. For example, the services Content-Driven Policy above has the following equivalent in the CFEngine language.

```
bundle agent service_example
{
services:

  "Dnscache"
    comment             => "Check services status of Dnscache",
    handle              => "srv_Dnscache_windows",
    service_policy      => "stop",
    service_method      => force_deps,
    action              => policy("fix"),
    ifvarclass          => "windows";

  "ALG"
    comment             => "Check services status of ALG",
    handle              => "srv_ALG_windows",
    service_policy      => "start",
    service_method      => force_deps,
    action             => policy("warn"),
    ifvarclass         => "windows";

  "RemoteRegistry"
    comment            => "Check services status of ALG",
    handle             => "srv_ALG_windows",
    service_policy     => "start",
    service_method     => force_deps,
    action             => policy("fix"),
    ifvarclass         => "Windows_Server_2008";

}
```

Writing this policy is clearly more time-consuming and error-prone. On the other hand, it allows for much more flexibility than Content-Driven Policies, when that is needed.

CFEngine provides Content-Driven Policies to cover mainstream management tasks like the following.

- File change/difference management
- Service management
- Database management
- Application / script management

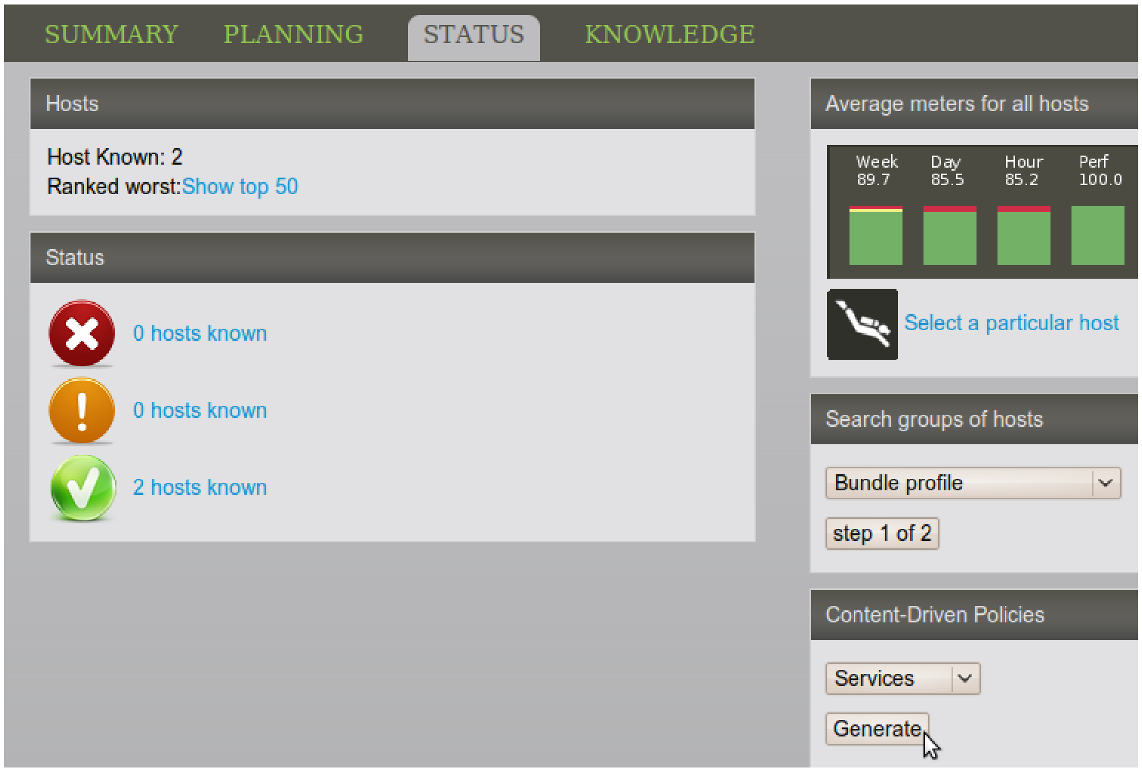
11.2 Getting started

All the CDP input files are located in `/var/cfengine/masterfiles/cdp_inputs` on your policy hub. Nova is bundled with some examples, but ensure to *edit the examples before enabling them*.

To enable all the CDPs located in `cdp_inputs/`, open `promises.cf` in your masterfiles directory. Remove the comment (#) in front of the cdp entries in `bundlesequence` and `inputs`, to make it look like the following.

```
bundlesequence => {
                  "def",
                  "cfengine_management",
                  "service_catalogue",
                  "cdp",
...

        inputs => {
...
                  "cdp_lib/cdp.cf",
                  "cdp_lib/cdp_acls.cf",
                  "cdp_lib/cdp_registry.cf",
                  "cdp_lib/cdp_file_changes.cf",
                  "cdp_lib/cdp_file_diffs.cf",
                  "cdp_lib/cdp_services.cf",
                  "cdp_lib/cdp_commands.cf",
...
```

When the hosts start reporting back on the outcome of these policies (usually within 10 minutes), you can view the reports from your Knowledge Map on the policy hub. Go to the `Status` tab, find the Content-Driven Policies drop-down, select the CDP type (e.g. `Services`) and click `Generate`.



Host Known: 2
Ranked worst:Show top 50

0 hosts known
0 hosts known
2 hosts known

Week	Day	Hour	Perf
89.7	85.5	85.2	100.0

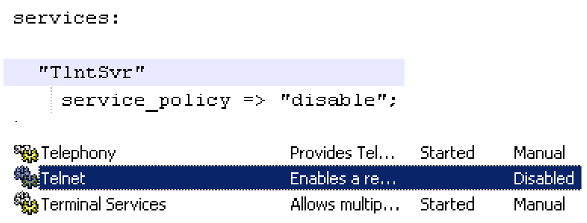
12 Windows-specific features in Nova

In this section, we will explore the Windows-specific features of the native Windows version of CFEngine Nova, and how it integrates with Windows. We will also consider features that are more interesting or popular on Windows than on other platforms.

Feature highlights include Windows service management and integration, event logging, Windows registry repair, and fine-tuned file security support through access control lists. See the sections on databases and ACLs to find information on Windows registry repair and NTFS ACLs, respectively. We will look at some of the other added features next.

12.1 Windows service management

CFEngine Nova can maintain complete control of the state of all Windows services, in addition to Unix daemons. Services prone to security issues or errors can easily be given a disabled state.



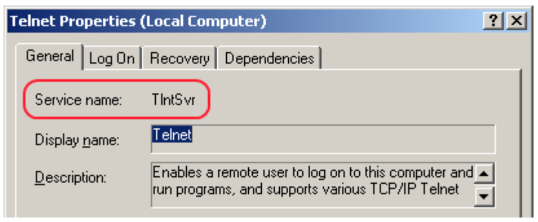
```
services:

  "TlntSvr"
    service_policy => "disable";
```

```
Telephony            Provides Tel...   Started   Manual
Telnet               Enables a re...             Disabled
Terminal Services    Allows multip...  Started   Manual
```

A service can also be given a running state, in which case CFEngine Nova ensures that it is running, and starts it if it is not, with parameters if desired. More advanced policy options are also available, including support for starting and stopping dependencies, and configuring when the services should be started (e.g. only when they are being used).

Furthermore, the CFEngine executor in Nova now runs as a Windows service itself. This means it runs in the background an starts with Windows, before any user logs in. It can be configured, started and stopped from the "Services" listing in Windows.

Note that the name of a service in Windows may be different from its "Display name". CFEngine Nova policies use the name, not the display name, due to the need of uniqueness.



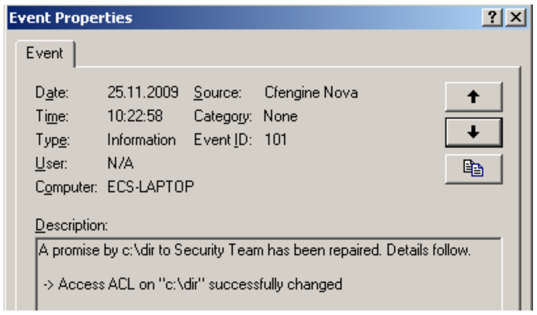
Service name: TlntSvr

Display name: Telnet

Description: Enables a remote user to log on to this computer and run programs, and supports various TCP/IP Telnet

12.2 Windows event logging

Event logs are the Windows counterpart to syslog from Unix. The main difference is that event logs aim to group similar log messages, giving each group an event id.



Date:	25.11.2009	Source:	Cfengine Nova
Time:	10:22:58	Category:	None
Type:	Information	Event ID:	101
User:	N/A		
Computer:	ECS-LAPTOP		

Description:

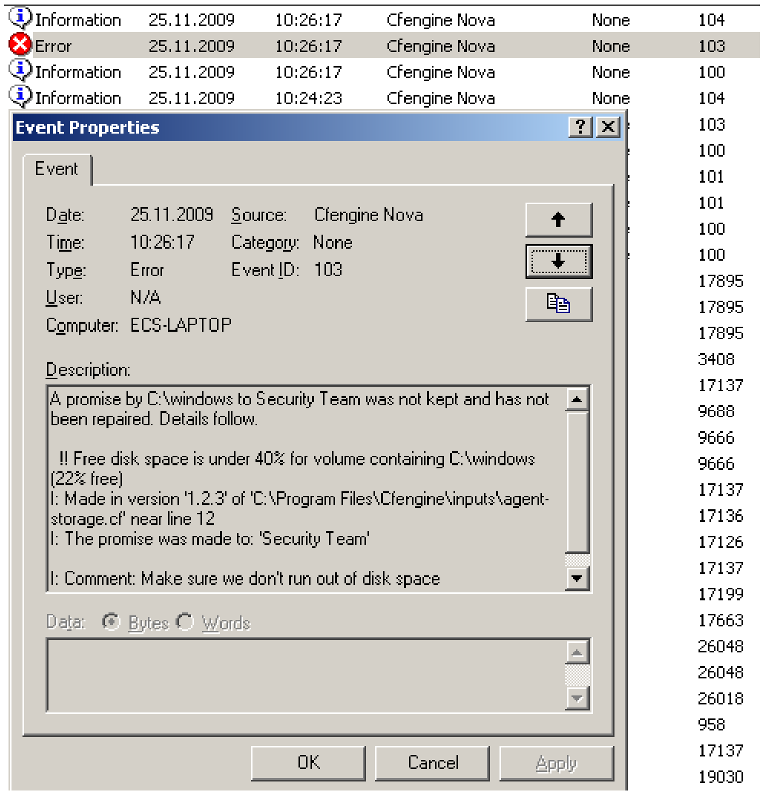
A promise by c:\dir to Security Team has been repaired. Details follow.

-> Access ACL on "c:\dir" successfully changed

A program that creates logs, such as CFEngine Nova, must define the possible event IDs, and their meaning. In many applications, only one event id is defined, a generic log message. However, CFEngine Nova defines the following range of event IDs, which allows for automatic handling of log messages.

Description	**Event ID**	**Type**
Promise kept	100	Information
Promise repaired	101	Information
Promise not repaired due warn only policy	102	Error
Promise not repaired due to error	103	Error
Report promise	104	Information
Generic information	105	Information
Generic verbose	106	Information
Generic warning	107	Warning

Generic error	108	Error



```
Information  25.11.2009  10:26:17  Cfengine Nova  None  104
Error        25.11.2009  10:26:17  Cfengine Nova  None  103
Information  25.11.2009  10:26:17  Cfengine Nova  None  100
Information  25.11.2009  10:24:23  Cfengine Nova  None  104
                                                        103
                                                        100
                                                        101
                                                        101
                                                        100
                                                        100
                                                        17895
                                                        17895
                                                        17895
                                                        3408
                                                        17137
                                                        9688
                                                        9666
                                                        9666
                                                        17137
                                                        17136
                                                        17126
                                                        17137
                                                        17199
                                                        17663
                                                        26048
                                                        26048
                                                        26018
                                                        958
                                                        17137
                                                        19030
```

Event Properties

Event

Date: 25.11.2009 Source: Cfengine Nova
Time: 10:26:17 Category: None
Type: Error Event ID: 103
User: N/A
Computer: ECS-LAPTOP

Description:

```
A promise by C:\windows to Security Team was not kept and has not
been repaired. Details follow.

 !! Free disk space is under 40% for volume containing C:\windows
(22% free)
I: Made in version '1.2.3' of 'C:\Program Files\Cfengine\inputs\agent-
storage.cf' near line 12
I: The promise was made to: 'Security Team'

I: Comment: Make sure we don't run out of disk space
```

The Nova event logs can be found under the "System" logs. Almost all monitoring products for Windows supports reading event logs, and they can thus monitor logs from CFEngine Nova as well. This makes it possible to do more advanced querying on the status of a machine running CFEngine Nova, e.g. to show all promises that have not been kept in a certain time interval. However, we recommend using the Knowledge Map to do more advanced things, as it is specifically made for this purpose and supports all operating systems that CFEngine runs on.

12.3 Windows special variables

Three new special variables have been added to the Windows version of CFEngine Nova.

- `sys.windir` contains the Windows directory, e.g. "C:\WINDOWS".
- `sys.winsysdir` contains the Windows system directory, e.g. "C:\WINDOWS\system32".
- `sys.winprogdir` contains the program files directory, e.g. "C:\Program Files".

Note that these variables are not statically coded, but retrieved from the current system. For example, `sys.winprogdir` is often different on Windows versions in distinct languages.

12.4 Windows hard classes

The Windows version of CFEngine Nova defines hard classes to pinpoint the exact version of Windows that it is running on, the service pack version and if it's a server or workstation.

First of all, the class `windows` is defined on all Windows platforms. For Windows workstations, such as Windows XP, `WinWorkstation` is defined. On Windows servers, such as Windows Server 2003, WinServer is defined. In addition, if the server is a domain controller, DomainController is defined. Note that if DomainController is defined, then WinServer is also defined, for natural reasons.

The class Service_Pack_X_Y is defined according to the service pack version. For example, at the time of writing, Service_Pack_3_0 is set on an updated Windows XP operating system.

To allow taking specific actions on different Windows versions, one of the following hard classes is defined.

- Windows_7
- Windows_Server_2008_R2
- Windows_Server_2008
- Windows_Vista
- Windows_Server_2003_R2
- Windows_Home_Server
- Windows_Server_2003
- Windows_XP_Professional_x64_Edition
- Windows_XP
- Windows_2000

Note that all defined hard classes for a given system is shown by running `cf-promises -v`.

12.5 Notes on windows policies

A potential problem source when writing policies for windows is that paths to executables often contain spaces. This makes it impossible for CFEngine to know where the executable ends and the parameters to it starts. To solve this, we place escaped quotes around the executable.

Additionally, Windows does not support that processes start themselves in in the background (i.e. fork off a child process in the Unix world). The result is that CFEngine is always waiting for the commands to finish execution before checking the next promise. To avoid this, use the background attribute in the action body-part.

Both these things are demonstrated in the following example.

```
body common control
{
bundlesequence  => { "main" };
}

bundle agent main
{
commands:

"\"C:\Program Files\Some Dir\program name.bat\" --silent --batch"
  action => background;
}

body action background
{
```

```
background => "true";
}
```

Finally, one should note that Windows lacks support for certain features that are utilized in Unix versions of CFEngine. These include symbolic links, file groups, user and group identifiers.

Thus, the parts of promises containing these features will be ignored. For example, the `getgid()` function does not return anything on Windows. The reference manual documents exactly which promises are ignored and not. Also, `cf-agent` from CFEngine Nova prints warning messages on ignored attributes when run in verbose mode.

CFEngine®