CFEngine 3 Nova Evaluation Guide

CFEngine Enterprise Documentation
Updated 7. May 2012

CFEngine AS

Table of Contents

1 Introduction

CFEngine comes in two software editions: CFEngine Community and CFEngine 3 Nova. CFEngine 3 Nova is a commercial subscription offering, with simple productivity enhancements and reporting extensions. Features include compliance management, reporting and business integration, and tools for handling necessary complexity. CFEngine 3 Nova has features to support Cloud Computing for public and private clouds, as well as greater integration facilities with database resources.

The evaluation program aims to familiarize the user with CFEngine 3 Nova through examples and practical use of the Nova Mission Portal, a web based graphical user interface. You do not need experience with CFEngine 3, the current document provides a step-by-step guide to help you find and use the product's major features. Of course, the deeper knowledge of system administration and CFEngine 3 language that you possess, the more you will get out of this evaluation.

The Mission Portal is the centerpiece of user interaction with CFEngine 3 Nova, designed to suit the needs of the next generation of system administrators and IT managers alike. The evaluation program will allow the user to quickly:

- View the status of all servers (e.g. availability, compliance, performance)
- Consult out-of-the-box reports, such as:
    - Software installed (by machine)
    - Compliance
    - File diffs
    - Promises not kept
- Understand and write a simple CFEngine Policy file

For the purpose of this evaluation program, CFEngine 3 Nova has been set up in a network comprising several nodes. One node serves as a central hub to distribute CFEngine configuration policies and collect reports from the other nodes (clients). They run on different operating systems, comprising several linux flavors and windows servers.

If you have not already done so, you may request access to the CFEngine Nova Evaluation Program by entering your information at `http://cfengine.com/evaluation`.

2 Get started

2.1 Accessing CFEngine 3 Nova

You should have received an email with account information after making an evaluation agreement with CFEngine. In it, you will find information on how to access your dedicated CFEngine 3 Nova instance, the CFEngine support ticketing system, and other online resources. Go to the url indicated in this email to access the Nova Mission Portal, use your credentials to log in.

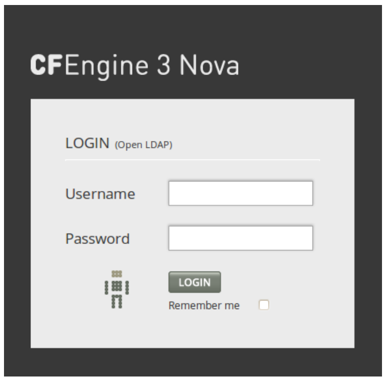


Figure: Nova Mission Portal login screen

The normal Nova Mission Portal is divided into four main sections called *rooms*. Here we have added a fifth room which is specific to the evaluation environment (i.e. is not shipped with the actual product): shell access to machines. Each *room* offers insight into different aspects of operations and is a beginning from which you can refine your overview and search through information.

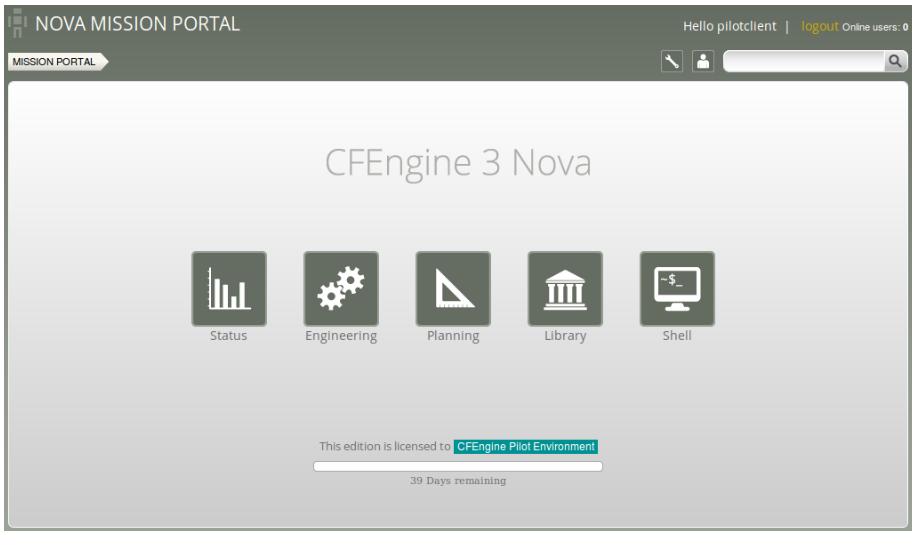


Figure: Nova Mission Portal

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

- Shell: web based ssh access to the machines in the network (only available in evaluation environment).

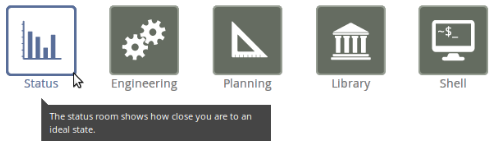
You can always use the breadcrumb in the top left corner to navigate and see where you are in the Mission Portal.

2.2 CFEngine terms used in this document

CFEngine uses a declarative language that describes the desired state of a system. Individual statements are called *promises*. Promises can be *kept* (CFEngine was able to keep the promise about the desired state), *not kept* (CFEngine was not able to keep the promise about the desired state) or *repaired* (promise was initially not kept, CFEngine fixed this). The desired state of your system is thus described in collections of promises, assembled in CFEngine policy files with the extension '`.cf`'.

2.3 Status room

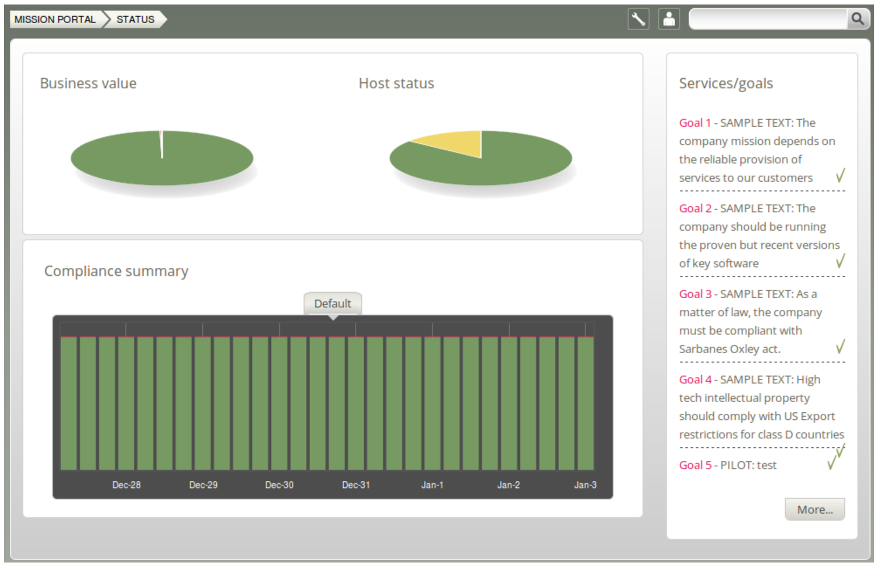
Action item: Click the **Status** icon in the main page of the Mission Portal to enter the '`Status`' room.



The status room shows how close you are to an ideal state.

Figure: Go to Status room

The status room gives an overview of IT goals and how close you are to an ideal state. Each section in the room is described below.



Goal 1 - SAMPLE TEXT: The company mission depends on the reliable provision of services to our customers ✓

Goal 2 - SAMPLE TEXT: The company should be running the proven but recent versions of key software ✓

Goal 3 - SAMPLE TEXT: As a matter of law, the company must be compliant with Sarbanes Oxley act. ✓

Goal 4 - SAMPLE TEXT: High tech intellectual property should comply with US Export restrictions for class D countries ✓

Goal 5 - PILOT: test ✓

Figure: Status room

2.3.1 Business Value and Host Status

The two pie charts show the business value of the promises kept/not kept and well as host status, respectively. Business value is associated with the value of promises as defined in policy files. In the Host Status chart, each node in the network represents a slice of the pie and is classified into red, yellow, green and blue according to the level of their compliance:

- Red: more than 20% of the promises were not kept
- Yellow: 20% or more of the promises were repaired and total compliance is above 80%
- Green: more than 80% of the promises were kept
- Blue: there is no contact between the hub and the client host (host unreachable)

2.3.2 Compliance Summary

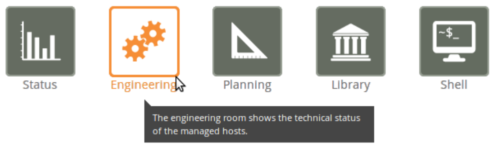
The row of bar meters shows the compliance (average percentage of promises kept, repaired or not kept) of all registered hosts in blocks of 6 hours for the past week. It summarizes performance and anomalous behavior in a simple red (promises not kept), yellow (promises repaired) and green (promises kept) scale. Click on a bar to see which promises were kept/not kept.

2.3.3 Services/Goals

A summary of Mission goals as defined in user policy files (these examples are from '`company_knowledge.cf`'). You can look at editing policy files in Appendix A [Policy editor (beta)], page 23.

2.4 Engineering room

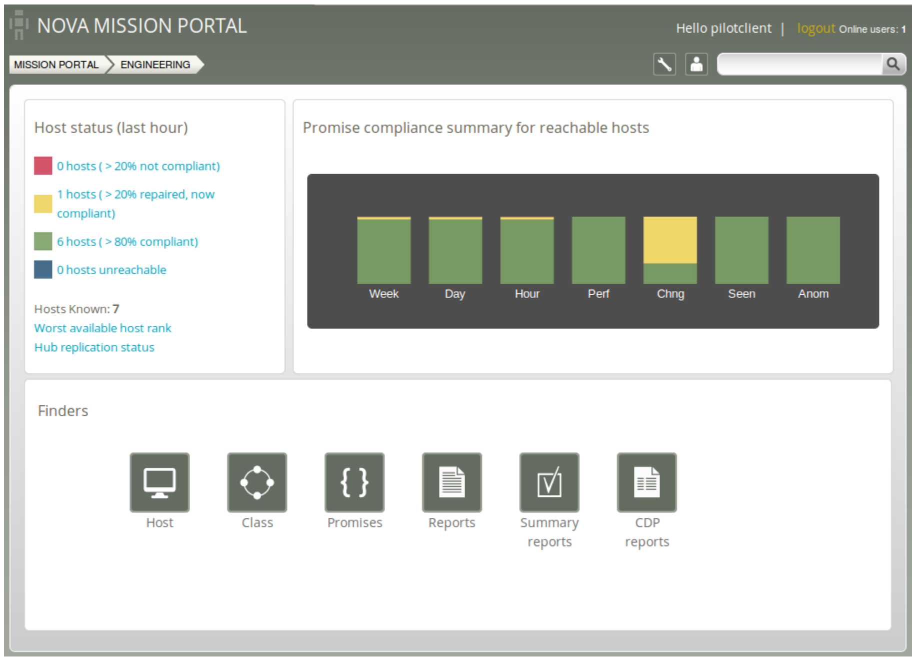
Action item: Click **NOVA MISSION PORTAL** on the top left (or **MISSION PORTAL** in the breadcrumb) to go back to the main Mission Portal page. Click the **Engineering** icon in the main page of the Mission Portal to enter the '`Engineering`' room.



The engineering room shows the technical status of the managed hosts.

Figure: Go to Engineering room

Mission engineering illustrates the state of the system in relation to the desired state at all scales. Zoom in to specific areas and examine the impact of promises, query data, and extract reports. Each section in the room is described below.



🟥 0 hosts (> 20% not compliant)
🟨 1 hosts (> 20% repaired, now compliant)
🟩 6 hosts (> 80% compliant)
🟦 0 hosts unreachable

Hosts Known: 7
Worst available host rank
Hub replication status

Figure: Engineering room

2.4.1 Host Status (last hour)

- The hosts are classified into red, yellow, green and blue according to the status of their compliance (as defined in Section 2.3.1 [Business Value and Host Status], page 4). Clicking a link produces a list of the hosts in that category.
- Hosts known: Shows the total number of hosts (red, green, yellow or blue) that are in the network
- Worst available host rank: Display the hosts (that have been in contact with the hub) with the most promises not kept over the last hour.
- Hub replication status: Display status of redundant monitoring hubs (not activated in evaluation environment).

2.4.2 Promise compliance summary for reachable hosts

The row of bar meters shows the compliance (average percentage of promises kept, repaired or not kept) of registered hosts. It summarizes performance and anomalous behavior in a simple red (promises not kept), yellow (promises repaired), and green (promises kept) scale. The "Chng" bar relates to the amount of changes made to files monitored by a CFEngine policy in the last hour (change watch). It is green if no changes have been made. The level of yellow increases as changes occur (but it will never be red). For the "Seen" bar, CFEngine monitors the average time between connections to the clients and reports deviations as green, yellow or red according to the size of the deviation. The "Anom" bar relates to anomalies and is generated from monitoring data (vital signs) for the last week. CFEngine uses the average value of each vital sign to report deviations as green, yellow or red according to the size of the deviation.

2.4.3 Finders

The Mission '`Engineering`' room comes with finder functions (modules that make it simple and intuitive to browse and search for objects of a particular type): Host, Class, Promises, Reports, Summary reports, and CDP (Content Driven Policies) reports. We will take a closer look at Reports and CDP reports in this document, but feel free to explore the different finder functions on your own.

2.5 Planning room

Action item: Click **NOVA MISSION PORTAL** on the top left (or **MISSION PORTAL** in the breadcrumb) to go back to the main Mission Portal page. Click the **Planning** icon in the main page of the Mission Portal to enter the '`Planning`' room.

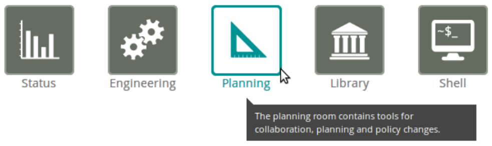
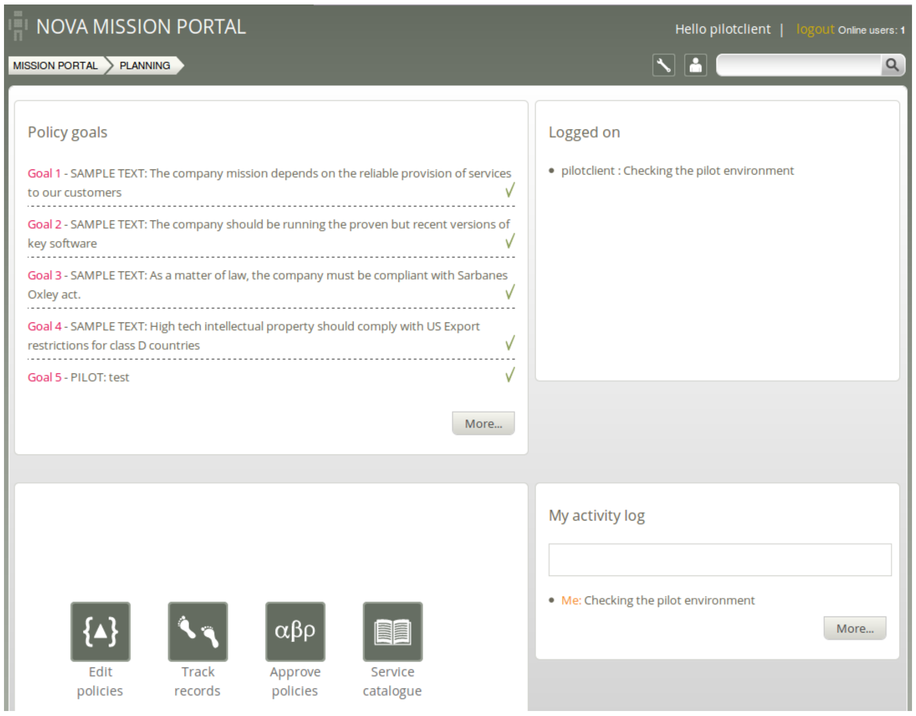

Status
Engineering
Planning
Library
Shell
The planning room contains tools for collaboration, planning and policy changes.

Figure: Go to Planning room

The '`Planning`' room allows you to make changes to policies, goals and implement specific tactics to achieve the desired state. Interact with data, approve changes and anomalies. Get an overview of users logged on to the Mission Portal, as well as their current activity. Each section in the room is described below.



Policy goals

Goal 1 - SAMPLE TEXT: The company mission depends on the reliable provision of services to our customers √

Goal 2 - SAMPLE TEXT: The company should be running the proven but recent versions of key software √

Goal 3 - SAMPLE TEXT: As a matter of law, the company must be compliant with Sarbanes Oxley act. √

Goal 4 - SAMPLE TEXT: High tech intellectual property should comply with US Export restrictions for class D countries √

Goal 5 - PILOT: test √

Logged on

- pilotclient : Checking the pilot environment

My activity log

- Me: Checking the pilot environment

Figure: Planning room

Policy Goals: List of policy goals as defined in policy files; these examples are from '`company_knowledge.cf`' (same as in the '`Status`' room).

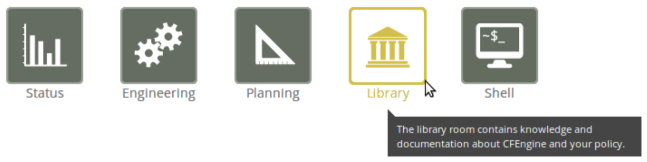
Icons:

- Edit policies: Edit policy files in the integrated policy editor
- Track records: Overview of promises repaired or not kept
- Approve policies: Add your approval to adopt a policy revision (attaches your comments to a revision number)
- Service catalogue: See which bundles contribute to policy goals

Logged on: Shows users currently logged on to the Mission Portal and their activity as stated in their activity log (see below).
Activity log: Shows the latest activity entries. Type in a new activity to keep colleagues posted on current work.

2.6 Library room

Action item: Click **NOVA MISSION PORTAL** on the top left (or **MISSION PORTAL** in the breadcrumb) to go back to the main Mission Portal page. Click the **Library** icon in the main page of the Mission Portal to enter the '`Library`' room.



The library room contains knowledge and documentation about CFEngine and your policy.

Figure: Go to Library room

The Library room contains finders for documents, topics, a notes archive, and a link to the CFEngine community web pages.

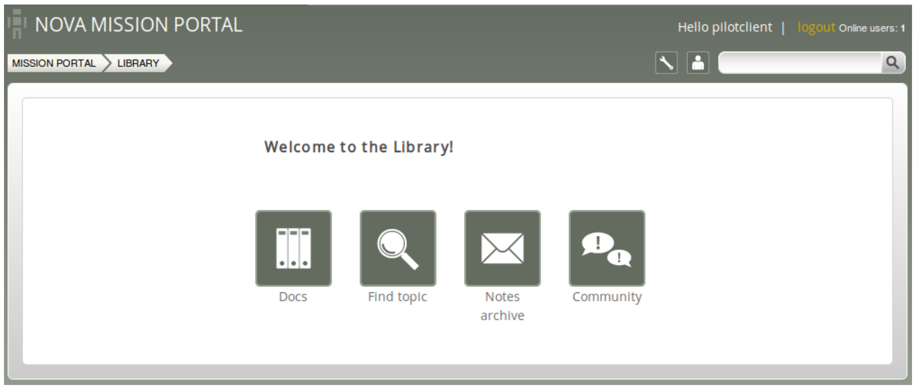


Figure: Library room

- Docs: Overview of documentation that was packaged with CFEngine 3 Nova.
- Find Topic: Opens a finder where you can search for topics either by scrolling through the alphabetical list or by typing in a search box (same as the search box on top right of page).
- Notes Archive: Get an overview of all notes made by Mission Portal users.
- Community: External link to the CFEngine community

2.7 Shell access (evaluation feature only)

Although the Mission Portal is the centerpiece of user interaction with CFEngine 3 Nova, system administrators often also wish to have shell access to their machines. We have therefore added a web based application specific to the Mission Portal in the evaluation environment (i.e. not found in the actual product) to provide a shell interface. You may explore the environment and verify the behavior of the Mission Portal by using this application.

Action item: Click **NOVA MISSION PORTAL** on the top left (or **MISSION PORTAL** in the breadcrumb) to go back to the main Mission Portal page. Click the **Shell** icon to enter the web based shell application.

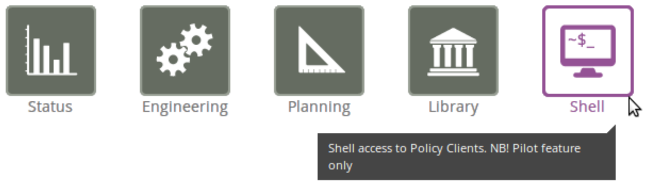
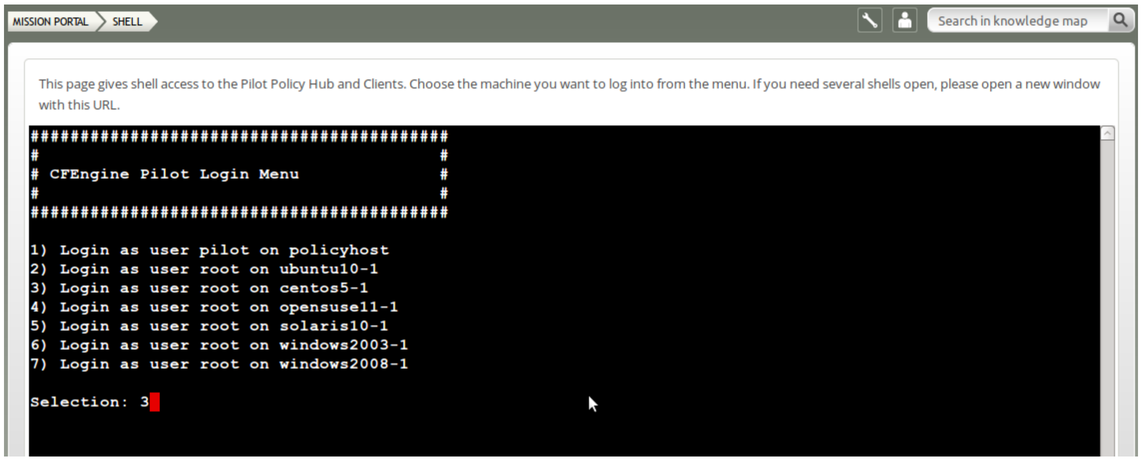


Figure: Go to shell application

The following screen will appear once the application is loaded:



This page gives shell access to the Pilot Policy Hub and Clients. Choose the machine you want to log into from the menu. If you need several shells open, please open a new window with this URL.

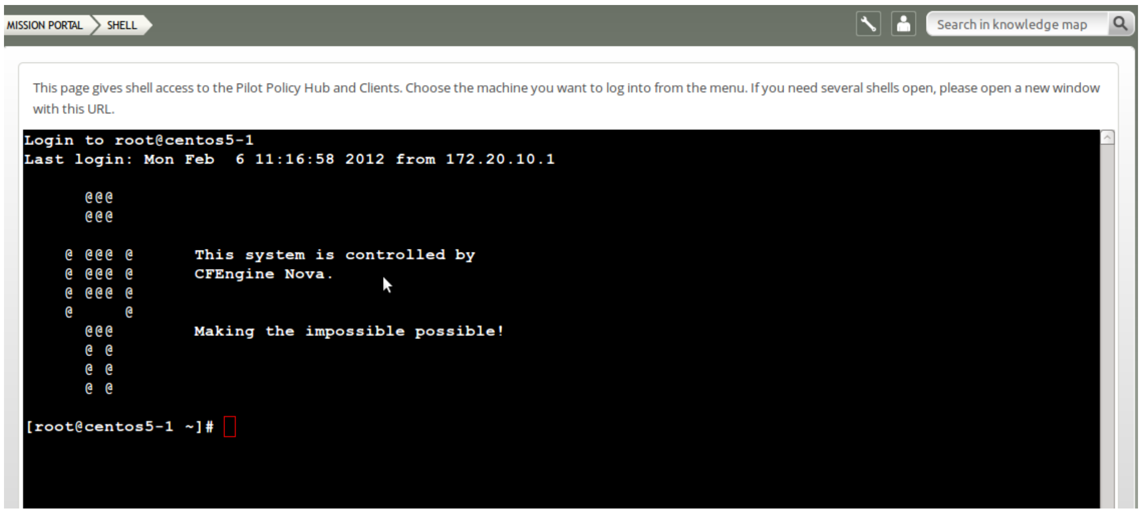
```
##########################################
#                                        #
# CFEngine Pilot Login Menu              #
#                                        #
##########################################

1) Login as user pilot on policyhost
2) Login as user root on ubuntu10-1
3) Login as user root on centos5-1
4) Login as user root on opensuse11-1
5) Login as user root on solaris10-1
6) Login as user root on windows2003-1
7) Login as user root on windows2008-1

Selection: 3
```

Figure: Shell menu

The shell application has restricted access on the hub (policy host) for security reasons. Root access has been granted on the clients, so you should be able to perform most tasks from this interface (although some restrictions have been put in place to contain access to the evaluation environment only).

Action item: Type the number of the node you wish to log on to and press **Enter**.

The following screen will appear after a successful login:

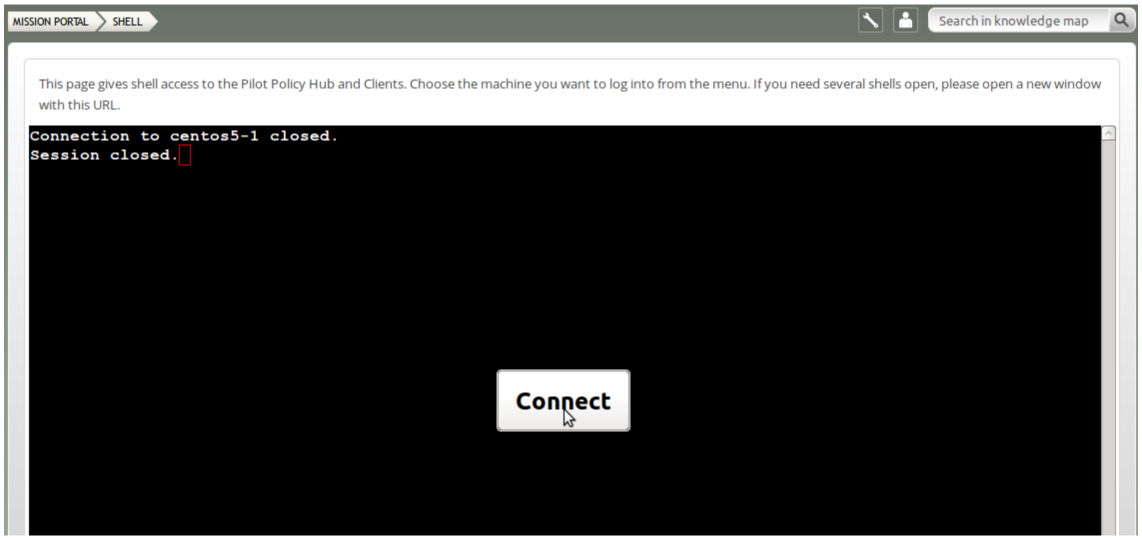


This page gives shell access to the Pilot Policy Hub and Clients. Choose the machine you want to log into from the menu. If you need several shells open, please open a new window with this URL.

```
Login to root@centos5-1
Last login: Mon Feb  6 11:16:58 2012 from 172.20.10.1

      @@@
      @@@

    @ @@@ @       This system is controlled by
    @ @@@ @       CFEngine Nova.
    @ @@@ @
    @     @
      @@@         Making the impossible possible!
      @ @
      @ @
      @ @

[root@centos5-1 ~]#
```

Figure: CFEngine Pilot Login Menu

Action item: Explore the environment through the command line interface. Type **exit** to finish the session on the current node. The following screenshot will appear:



This page gives shell access to the Pilot Policy Hub and Clients. Choose the machine you want to log into from the menu. If you need several shells open, please open a new window with this URL.

```
Connection to centos5-1 closed.
Session closed.
```

Figure: Shell logout/connect

Click **Connect** to access the '`CFEngine Pilot Login Menu`' again and choose another node to log on to. Alternatively you can navigate away from the shell access by clicking **NOVA MISSION PORTAL** on the top left (or **MISSION PORTAL** in the breadcrumb).

3 Standard reports in CFEngine 3 Nova

A significant capability of CFEngine 3 Nova is automated system reporting: it collects history, state and change data about computers and ties them together. A report is a tabular summary of CFEngine's internal information, tailored to a particular purpose, searchable, and describes attributes and qualities of managed hosts.

3.1 Reports finder

Standard reports in CFEngine 3 Nova can be accessed through the '`Reports finder`':

Action item: Enter the '`Engineering`' room as shown in Section 2.4 [Engineering room], page 4, locate and click the **Reports** icon to open the finder.

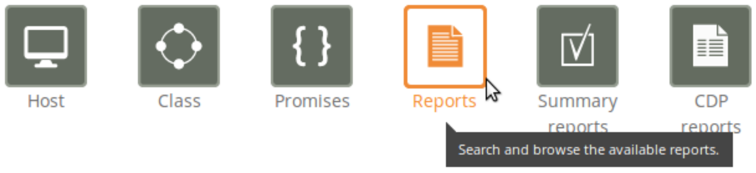
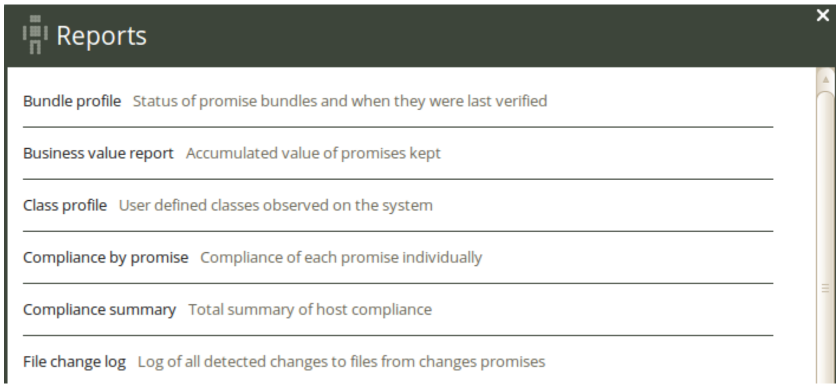


Figure: Click to open the Reports finder.



Reports

Bundle profile Status of promise bundles and when they were last verified

Business value report Accumulated value of promises kept

Class profile User defined classes observed on the system

Compliance by promise Compliance of each promise individually

Compliance summary Total summary of host compliance

File change log Log of all detected changes to files from changes promises

The finder lists all the standard report categories, each category contains information about different aspects of the Mission. When you click one of them, the '`Report finder`' will present a query form that is adapted to the chosen report category.

3.2 Example - Software report

Action item: Scroll down and click **Software installed** towards the bottom of the '`Report finder`' to open a query window.

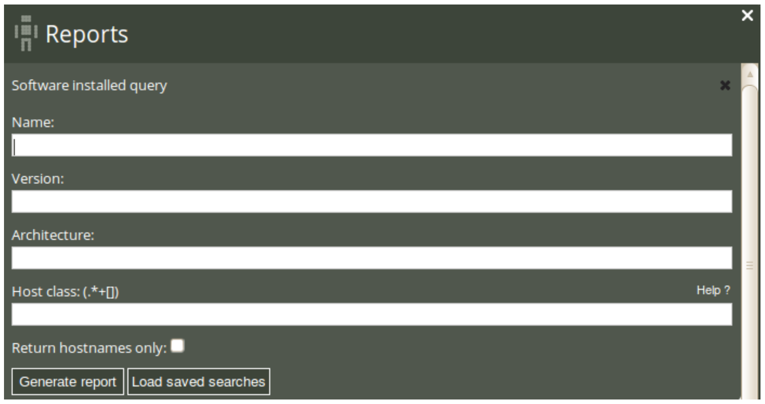
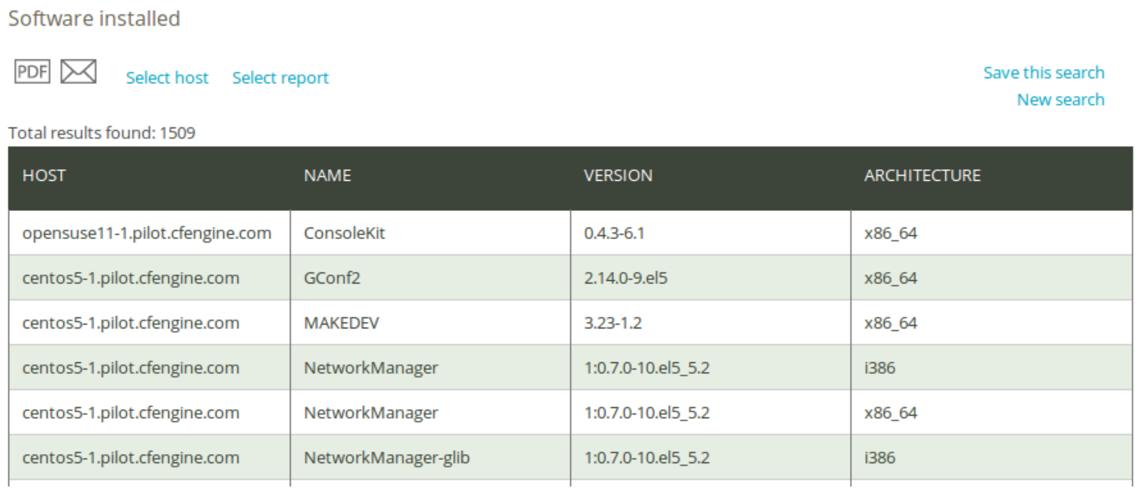


Figure: Software installed query

Action item: Click **Generate report** in the query window without filling in any of the search fields.

The above action will make a report listing all hosts and software packages claimed to be installed according to the local package manager. The results are presented in table form, with the columns '`Host`' (host name), '`Name`' (of software package), '`Version`' (of software package), and '`Architecture`' (of machine on which software runs). Sort the entries on the page by clicking the column headers.



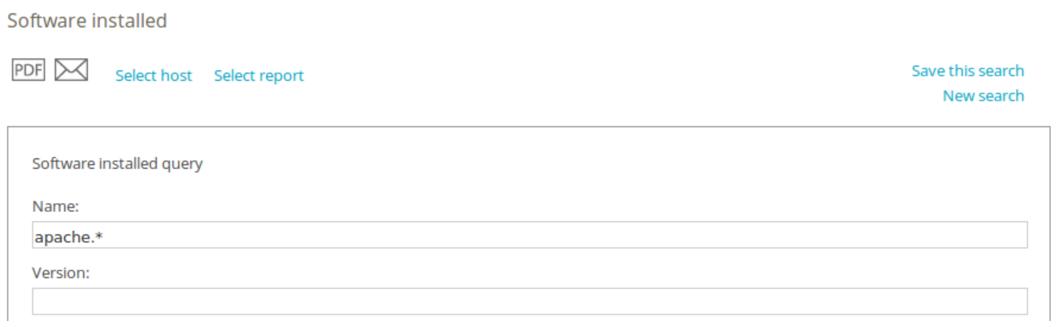
Total results found: 1509

HOST	NAME	VERSION	ARCHITECTURE
opensuse11-1.pilot.cfengine.com	ConsoleKit	0.4.3-6.1	x86_64
centos5-1.pilot.cfengine.com	GConf2	2.14.0-9.el5	x86_64
centos5-1.pilot.cfengine.com	MAKEDEV	3.23-1.2	x86_64
centos5-1.pilot.cfengine.com	NetworkManager	1:0.7.0-10.el5_5.2	i386
centos5-1.pilot.cfengine.com	NetworkManager	1:0.7.0-10.el5_5.2	x86_64
centos5-1.pilot.cfengine.com	NetworkManager-glib	1:0.7.0-10.el5_5.2	i386

Figure: Software installed report

There are two ways of narrowing down the listing in these reports: one is to enter filtering criteria directly in the report query window, the other is to click on '`New Search`' in the top right corner of the report itself and enter the filtering criteria there. We will do the latter:

Action item: Click **New Search** in the '`Software installed`' report and enter your search criteria as a regular expression (for instance, enter '`apache.*`' in the '`Name`' field to see what version of apache is running on the different hosts). Click **Generate report**.


Software installed

Select host Select report

Save this search
New search

Software installed query

Name:
apache.*

Version:

Figure: Narrow the search by entering search criteria

Once you have clicked ‘`Generate report`’, CFEngine 3 Nova will list an overview of all machines running some version of apache.

The combination of the extensive reports and detailed filtering is a flexible and powerful tool, made to give as general or granular an overview as the user needs it to be. A summary and explanation to all the standard reports can be found in the Nova Evaluation Guide Supplement.

4 Content Driven Policy (CDP) reports

Content-Driven Policies (CDPs) were introduced to make policy management easier. In contrast to policies written in the CFEngine language, they are composed of semi-colon separated fields in a text file that the user fills with content, like a spreadsheet or tabular file. Each line in the file is parsed and translated to be part of a regular CFEngine policy. We will take a closer look at CDP input files in a later section.

CDP reports are similar to standard reports, except that they reflect the effects of CDP input files instead of regular CFEngine policy files.

Action item: Enter the '`Engineering`' room as shown in Section 2.4 [Engineering room], page 4, then click the **CDP reports** icon to access the '`CDP reports finder`'.

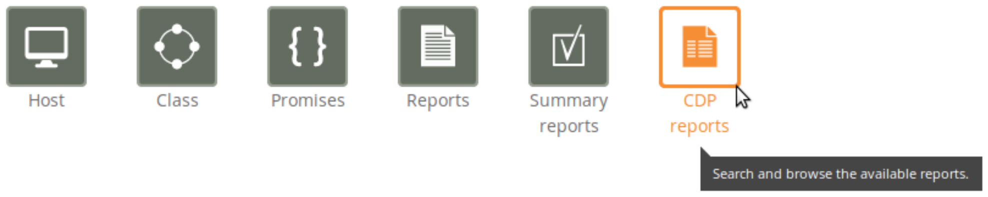
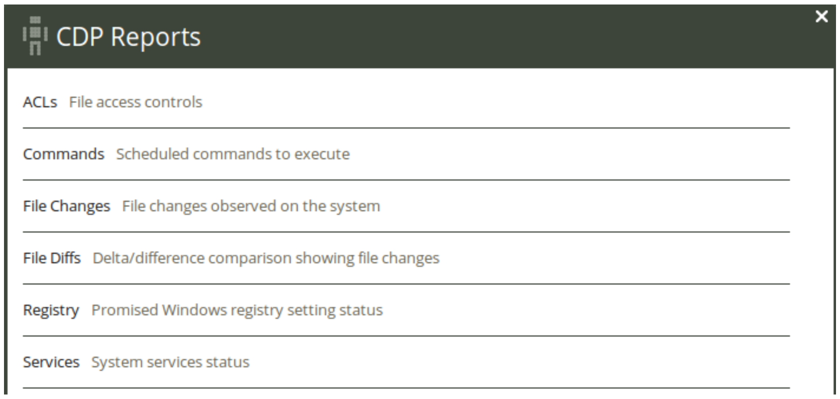


Figure: Go to CDP reports finder



CDP Reports

ACLs File access controls

Commands Scheduled commands to execute

File Changes File changes observed on the system

File Diffs Delta/difference comparison showing file changes

Registry Promised Windows registry setting status

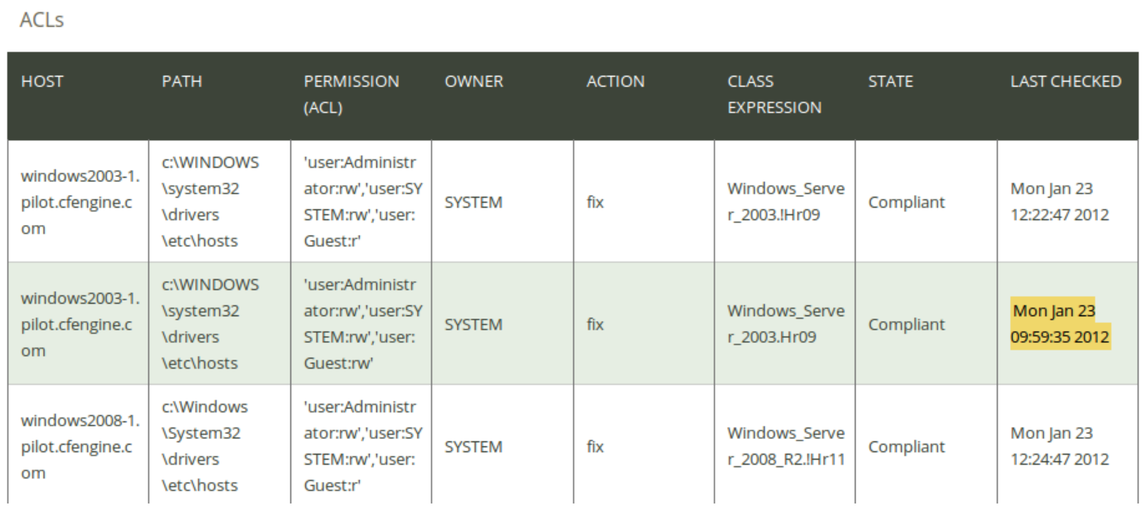
Services System services status

Figure: CDP reports finder

4.1 Example - ACLs (file access controls)

CFEngine 3 Nova comes with several default CDPs, we will use the '`ACLs report`' as an example. An access control list (ACL) is a list which specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects. Each entry in a typical ACL specifies a subject and an operation. For instance, if a file has an ACL that contains (Alice, delete), this would give Alice permission to delete the file.

Action item: Click on **ACLs** in the '`CDP Reports finder`' to access the ACLs report (there is no query window for CDP reports).



ACLs

HOST	PATH	PERMISSION (ACL)	OWNER	ACTION	CLASS EXPRESSION	STATE	LAST CHECKED
windows2003-1.pilot.cfengine.com	c:\WINDOWS\system32\drivers\etc\hosts	'user:Administrator:rw','user:SYSTEM:rw','user:Guest:r'	SYSTEM	fix	Windows_Server_2003.!Hr09	Compliant	Mon Jan 23 12:22:47 2012
windows2003-1.pilot.cfengine.com	c:\WINDOWS\system32\drivers\etc\hosts	'user:Administrator:rw','user:SYSTEM:rw','user:Guest:rw'	SYSTEM	fix	Windows_Server_2003.Hr09	Compliant	Mon Jan 23 09:59:35 2012
windows2008-1.pilot.cfengine.com	c:\Windows\System32\drivers\etc\hosts	'user:Administrator:rw','user:SYSTEM:rw','user:Guest:r'	SYSTEM	fix	Windows_Server_2008_R2.!Hr11	Compliant	Mon Jan 23 12:24:47 2012

Figure: ACLs report

The report lists an overview of host name (‘`Host`’), path of the affected object (‘`Path`’), the permission setting (‘`Permission (ACL)`’), owner of the affected object (‘`Owner`’), action that CFEngine should execute on the object (‘`Action`’), the context in which the promise was made (‘`Class expression`’), state of compliance (‘`State`’), and the time the promise was last checked (‘`Last checked`’).

5 Introduction to CFEngine policies

There are two types of policies that tell CFEngine what to do:

- Content Driven Policy (CDP): A text file containing semi-colon separated fields (‘.txt’ file extension) that specify actions
- Standard CFEngine policy file: Uses Standard CFEngine policy (‘.cf’ file extension) to specify actions

In the following, you will find a short introduction to both CDP and standard policies. You may choose to view and edit policy files in a beta version of the integrated Policy Editor (Section A.1 [The Policy Editor (beta)], page 23) or in an editor of your choice. The evaluation environment provides ssh access through a web interface so you may explore the environment or access policy files there (Section 2.7 [Shell access (evaluation feature only)], page 8). In the following, we will see a short introduction to both CDP and standard policies.

5.1 Content Driven Policy

As explained previously, content driven policies consist of semi-colon separated fields in a text file. The files also contain a header that explains the format and meaning of the fields and typically looks like this (lines have been split and indented for presentability):

```
#  ACLs On Files
#
#  FORMAT:   path;entity_type1:entity_name1:perms1,
      entity_type2:entity_name2:perms2,...;owner;action;class_expression
#
#  EXAMPLE:  C:\tmp;user:Administrator:rwx,user:SYSTEM:r;
      Administrator;fix;windows
#

# Windows 2003
c:\WINDOWS\system32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,
    user:Guest:r;SYSTEM;fix;Windows_Server_2003.!Hr09
c:\WINDOWS\system32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,
    user:Guest:rw;SYSTEM;fix;Windows_Server_2003.Hr09
# Windows 2008
c:\Windows\System32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,
    user:Guest:r;SYSTEM;fix;Windows_Server_2008_R2.!Hr11
c:\Windows\System32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,
    user:Guest:rw;SYSTEM;fix;Windows_Server_2008_R2.Hr11
```

Anything with a ‘;’ before or after it is a field entry. We need to look at the header of the file to understand the structure and meaning of the fields (under the FORMAT and EXAMPLE sections). In this case we have (line has been split and indented for presentability):

```
  #  FORMAT:   path;entity_type1:entity_name1:perms1,
         entity_type2:entity_name2:perms2,...;owner;action;class_expression
```

Splitting this up into separate fields:

path	Path of file to set permissions on.
entity_type1:entity_name1:perms1	This field defines the permissions ('`perms1`') that a user ('`entity_type1`'), and member of the group ('`entity_name1`'), has on the file defined in '`path`'.
entity_type2:entity_name2:perms2,...	Same as entity_type1:entity_name1:perms1, but for different user, group, and permission settings.
owner	Defines the owner of the file defined in '`path`'
action	Tells CFEngine what to do if the file permissions differ from what was defined in the ACLs policy. Can take the values '`fix`' (set permissions as defined in ACLs policy), '`warn`' (log and display a warning that the file permissions differ from what was defined in ACLs policy), and '`nop`' (no operation; no log entry, but print a warning in command-line interface).
class_expression	Context in which the permissions are set, i.e. a class expression (boolean) that needs to be fulfilled for the permissions to be set.

The advantage of CDPs is that it requires no knowledge of CFEngine syntax to quickly set up a basic policy. They function as an abstraction layer and allow system administrators to define specific functions with parameters that can be manipulated by others. The list format also allows for quick editing of multiple parameters, without having to go through tens or hundreds of lines of regular CFEngine code.

5.2 Standard CFEngine policy

Standard CFEngine policies consist of a declarative language that describes the desired state of a system. Individual statements are called *promises*, they can be grouped in bundles and have parametrized body templates ('`bundle`' and '`body`' are keywords in CFEngine that correspond to these). Below is an example of a simple CFEngine policy:

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

# Comments are defined by the hash tag (#), they will not be parsed by CFEngine.

bundle agent test      # This is a bundle of type agent, named 'test'
{
files:                 # This is the promise type, i.e. we make a promise about files
  "/tmp/testfile"      # This is the promiser (i.e. the concerned object)
```

```
    create => "true";  # This tells CFEngine to create the file
}
```

This policy contains the bare minimum of a standalone CFEngine policy and will create the file '`/tmp/testfile`'. A closer look at the different parts of the policy:

- The compulsory '`body common control`', containing (at least) a bundlesequence.
- Then follows a compulsory '`bundle`', of type '`agent`' and with the arbitrary name '`test`' (the bundle type reflects the affected CFEngine component and can take several values, here we use '`agent`').
- The bundle contains a promise type (here '`files:`'), a promiser (i.e. the affected object, here '`/tmp/testfile`') and an attribute about desired state (here '`create => "true"`'; will create the file if it does not already exist).

5.3 Example - Install packages from remote repository

In this example we will use the web based shell access to create and include a policy so that it installs a software package from the operating system standard repository.

Action item: Go to '`Shell access`' as described in Section 2.7 [Shell access (evaluation feature only)], page 8. Enter the number **1** and press **Enter** to log on to the host called `policyhub`. You will need to check out the files in the subversion repository connected to policyhub, type in the following to do so:

```
pilot@policyhub-1:~$ svn co http://localhost/svn/CFEnginePilot
```

Username and password for the subversion repository is 'pilot' and 'pilot'.

You now have read and write access to the policy files in the '`~/CFEnginePilot`' directory. Enter the following to change directories and create a new policy file, '`pilot_packages.cf`' (we use the vim editor in this example):

Action item: Type in the following in the command line:

```
pilot@policyhub-1:~$ cd CFEnginePilot
pilot@policyhub-1:~/CFEnginePilot$ vim pilot_packages.cf
```

Let us make a policy to install apache (httpd) on centos machines:

Action item: Press **i** to enter edit mode in vim, then enter the following content in the file (comments (text behind the hash tag (#)) are optional):

```
bundle agent pilot_packages        # bundle type agent, named 'pilot_packages'
{
vars:                              # promise type vars (promise about variables)
   "match_package" slist => {      # variable name is 'match_package' (this is the
                                   # promiser) and has type 'list of strings'
         "httpd"           # list entry value is httpd (for apache on centos)
         };

packages:                          # promise type packages
   centos::                        # class, or context, in which the promise
                                   # applies (here on all centos machines)
         "$(match_package)"        # expand the variable 'match_package' (this is
                                   # the promiser)
         handle                    => "pilot_remote_package_install",
         package_policy            => "add",         # add all packages in variable
         package_method            => yum;           # specify what package method
                                                     # to use (here yum)
}
```

Press **Esc** to exit the vim edit mode, then type `:wq` to write to file and quit.

The next step is to include this policy in 'promises.cf':

Action item: Open 'promises.cf':

```
   pilot@policyhub-1:~/CFEnginePilot$ vim promises.cf
```

Press **i** to enter edit mode in vim, then add the following lines to the first bundle (`bundle common pilot`) as indicated:

```
   bundle common pilot {
   vars:
      "bundlesequence" slist => {
                                   "pilot_simple_edit_policy",
                                   "pilot_packages",         # add this line
   #                               "pilot simple custom install",
   #                               "pilot_simple_custom_report_policy",
                                   };
      "inputs"   slist =>          {
                                   "pilot_simple_edit_policy.cf",
                                   "pilot_packages.cf",      # add this line
   #                               "pilot simple custom install.cf",
   #                               "pilot_simple_custom_report_policy.cf",
                                   };
}
```

Press **Esc** to exit the vim edit mode, then type `:wq` to write to file and quit.

We now need to check the syntax of the file by running the syntax checker (`cf-promises`) on '`promises.cf`':

Action item: Type the following in the command line interface (line has been split and indented for presentability):

```
pilot@policyhub-1:~/CFEnginePilot$
          /var/cfengine/cf-promises -f ~/CFEnginePilot/promises.cf
```

cf-promises will only output an error message if there is a syntax error in the policy, i.e. the command prompt will be blinking and waiting for input on a new line if everything is fine. If applicable, please correct any errors by opening and editing '`pilot_packages.cf`' as shown previously and proceed to the next step when all problems have been solved.
The next step to add '`pilot_packages.cf`' to the repository and commit the changes:

Action item: Type in the following in the command line:

```
pilot@policyhub-1:~/CFEnginePilot$ svn add pilot_packages.cf
pilot@policyhub-1:~/CFEnginePilot$ svn ci -m "added remote package install"
```

The policy will be executed at the next run of cf-agent (here in the evaluation environment we have set the interval to every two minutes). You can check that the policy has been implemented either through the mission portal or the web ssh interface:

Action item:

- Mission Portal: Software reports (Section 3.2 [Example - Software report], page 10) are only updated every six hours so this is not a convenient place to look for immediate changes. We can use the '`Promises repaired (log)`' report instead: Open the reports finder as shown in Section 3.1 [Reports finder], page 10, scroll down and click **Promises repaired (log)**. Enter the bundle name (pilot_packages) or promise handle (pilot_remote_package_install) as search criteria and click **Generate report**. The report should list the promise (not be empty).
- Web ssh: Choose the centos machine from the web ssh menu (as shown in Section 2.7 [Shell access (evaluation feature only)], page 8) and look for '`/var/cfengine/inputs/pilot_packages.cf`'.

5.4 Example - Install packages from local repository

In this example we will look at installing a custom package from a local repository. The package will install a text file in a given location, showing that the package has been successfully deployed. The example is transferable to other custom packages such as weblogic and includes installation on different operating systems (OS; CentOS, SuSE and Ubuntu). The following CFEngine policy has been written for this purpose (some lines have been split and indented for presentability):

```
bundle agent pilot_simple_custom_install {

        vars:
                any::
                        "pkg_version" string => "1.0.0-1";

                centos|ubuntu|SuSE::
                        "pkg_name" string => "cfengine-testpackage";

                ubuntu::
                        "pkg_file" string => "cfengine-testpackage_$(pkg_version)_
                                amd64.deb";

                centos|SuSE::
                        "pkg_file" string => "cfengine-testpackage-$(pkg_version).
                                x86_64.rpm";

        files:
                centos|SuSE|ubuntu::
                        "$(sys.workdir)/software_repository/$(pkg_file)"
                                comment => "Copy test package to Linux hosts",
                                copy_from => remote_cp("$(def.dir_custompkgs)/
                                        $(pkg_file)", "$(sys.policy_hub)"),
                                classes => if_ok("package_downloaded");

        packages:
                package_downloaded.(centos|SuSE)::
                        "$(pkg_name)"
                                comment => "Add package to rpm based hosts",
                                package_policy => "add",
                                package_version => "$(pkg_version)",
                                package_select => ">=",
                                package_architectures => { "x86_64" },
                                package_method => rpm_version("$(sys.workdir)/
                                        software_repository/");

                package_downloaded.ubuntu::
                        "$(pkg_name)"
                                comment => "Add package to dpkg based hosts",
                                package_policy => "add",
                                package_select => ">=",
                                package_version => "$(pkg_version)",
                                package_architectures => { ".*" },
                                package_method => dpkg_version("$(sys.workdir)/
                                        software_repository/");
}
```

This policy is divided into three main parts, each involving different promise types (i.e. promises about):

- vars: Promises about variables
  - In the context `any::` (valid for all OSs): `pkg_version` - the version of the package
  - In the context `centos|ubuntu|suse` (valid for CentOS, Ubunto and SuSE): `pkg_name` - the name of the package
  - In the context `ubuntu` (valid for Ubuntu): `pkg_file` - the OS-specific package file name
  - In the context `centos|suse` (valid for CentOS and SuSE): `pkg_file` - the OS-specific package file name
- files: Promises about files
  Promise to copy the OS-specific package files from the OS-specific work directories and set the class `package_downloaded` to true if the operation was successful
- packages: Promises about packages Promise to install the OS-specific package using the OS-specific package manager

The policy has already been prepared and is ready for use in the evaluation environment. To execute this policy:

Action item: If you have not already checked out the subversion repository as shown in the previous example Section 5.3 [Example - Install packages from remote repository], page 17, please do so. Open '`promises.cf`' as shown in the same example, then uncomment the following two lines (remove the hash tag (#)) in `bundle common pilot`:

- # `pilot_simple_custom_install`
- # `pilot_simple_custom_install.cf`.

Save and commit to subversion as shown in the same example.

The policy will be executed at the next run of cf-agent (here in the evaluation environment we have set the interval to every two minutes). You can check that the policy has been implemented either through the mission portal or the web ssh interface:

Action item:

- Mission Portal: Software reports (Section 3.2 [Example - Software report], page 10) are only updated every six hours so this is not a convenient place to look for immediate changes. We can use the '`Promises repaired (log)`' report instead: Open the reports finder as shown in Section 3.1 [Reports finder], page 10, scroll down and click **Promises repaired (log)**. Enter the bundle name (pilot_packages) or promise handle (pilot_remote_package_install) as search criteria and click **Generate report**. The report should list the promise (not be empty).
- Web ssh: Choose the centos machine from the web ssh menu (as shown in Section 2.7 [Shell access (evaluation feature only)], page 8) and look for '`/etc/cfengine_installed.txt`'.

6 Next steps

We recommend that users familiarize themselves with the CFEngine 3 documentation and continue to learn about the CFEngine language and CFEngine 3 Nova. Documents of interest include:

- Nova Evaluation Guide Supplement
- CFEngine 3 Nova Owner's Manual
- CFEngine 3 Concept guide
- CFEngine 3 Reference Manual

We also recommend that you attend a CFEngine 3 Training Course and/or employ CFEngine Consulting to help them plan your Nova implementation.

Use our contact form, or send an email to contact@cfengine.com, to be contacted about purchasing CFEngine 3 Nova.

Appendix A Policy editor (beta)

A.1 The Policy Editor (beta)

CFEngine 3 Nova has a beta version of an integrated editor, made for working on CFEngine policies. It provides syntax high-lighting and look-up to make policy writing easier.

Action item: To access the policy editor, enter the '`Planning`' room as described in Section 2.5 [Planning room], page 6, then click the **Edit policies** icon.

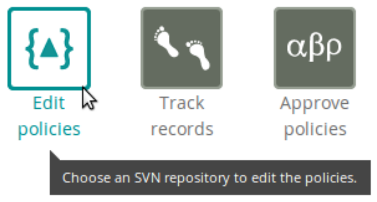
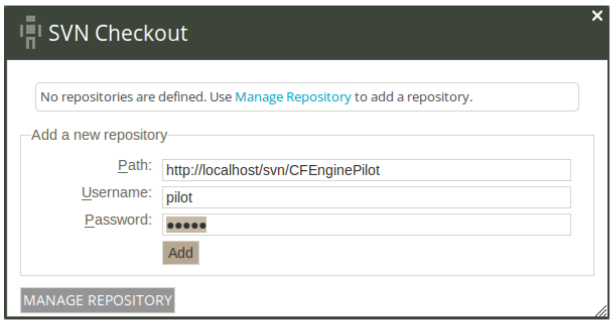


Figure: Go to Edit policies

The policy editor comes with a tie-in for Subversion version control repositories; the Nova Mission Portal will prompt you for the path and credentials to perform an SVN checkout.

Action item: Enter the path '`http://localhost/svn/CFEnginePilot`', username '`pilot`', and password '`pilot`'. Click **Add**.


SVN Checkout

No repositories are defined. Use Manage Repository to add a repository.

Add a new repository

Path: http://localhost/svn/CFEnginePilot
Username: pilot
Password: •••••

Figure: Add SVN repository

Action item: Click on **Checkout** to complete the checkout procedure.

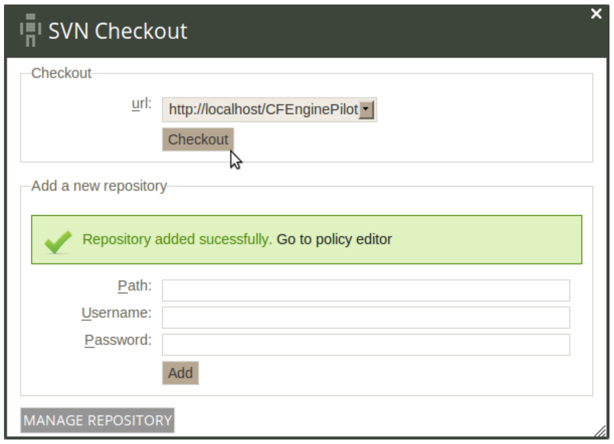
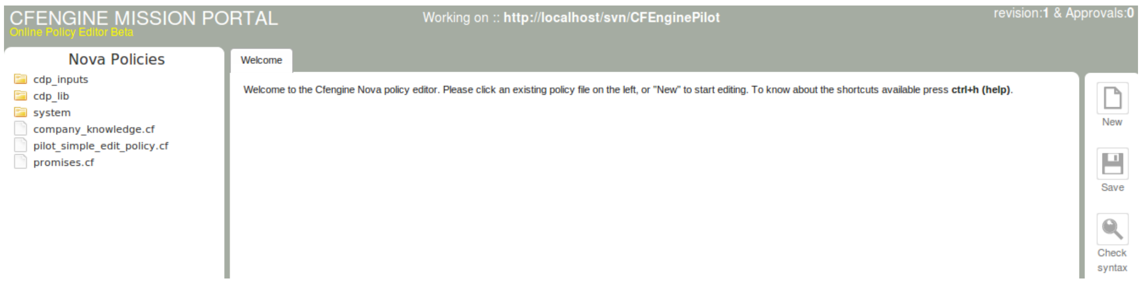


Figure: SVN checkout

The Policy Editor appears after a successful checkout. It consists of three main columns: on the left, a list of all the policies in the checked out repository (‘`Nova Policies`’); in the center, the editor space (where policy files will appear as sheets/tabs); on the right, basic file and Subversion commands. We take a closer look at these functions in the following sections.



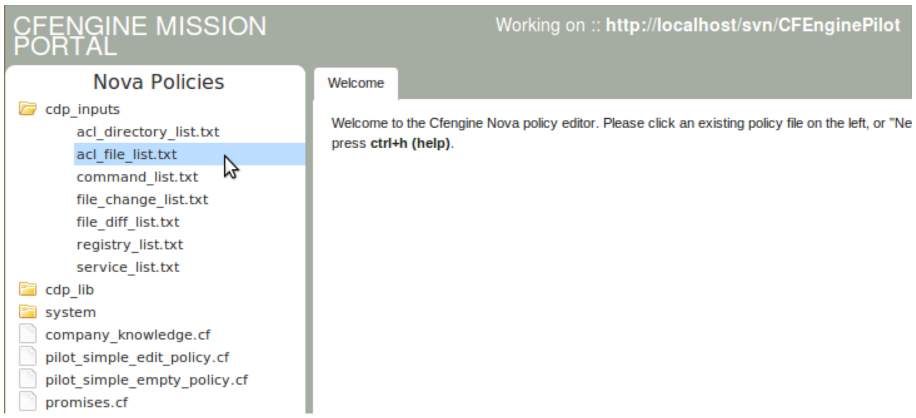
Welcome to the Cfengine Nova policy editor. Please click an existing policy file on the left, or "New" to start editing. To know about the shortcuts available press **ctrl+h (help)**.

Figure: The Policy Editor

A.2 Example - Edit a CDP input file

If you have not already opened the Policy Editor, follow the instruction in Section A.1 [The Policy Editor (beta)], page 23 to do so. The file '`acl_file_list.txt`' can be found under the '`cdp_inputs`' catalog in the policy editor.

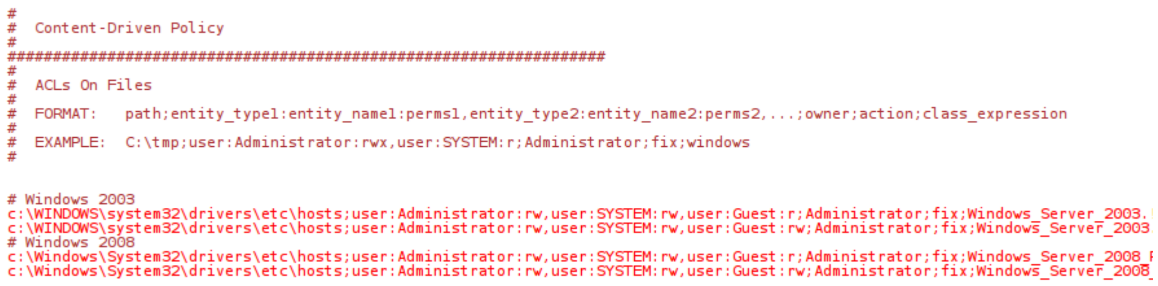
Action item: Click **cdp_inputs** in the file menu, then click **acl_file_list.txt** to open the file in the editor window.



Welcome to the Cfengine Nova policy editor. Please click an existing policy file on the left, or "Ne
press **ctrl+h (help)**.

Figure: Open the ACLs input file

The content of the file looks like this:



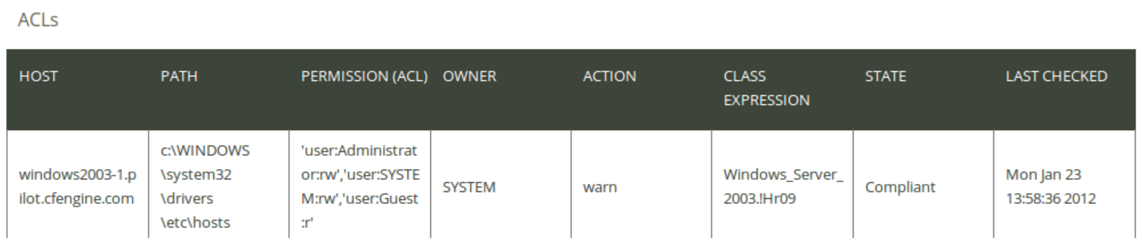
```
#
#  Content-Driven Policy
#
##################################################################
#
#  ACLs On Files
#
#  FORMAT:   path;entity_type1:entity_name1:perms1,entity_type2:entity_name2:perms2,...;owner;action;class_expression
#
#  EXAMPLE:  C:\tmp;user:Administrator:rwx,user:SYSTEM:r;Administrator;fix;windows
#


# Windows 2003
c:\WINDOWS\system32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,user:Guest:r;Administrator;fix;Windows_Server_2003.
c:\WINDOWS\system32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,user:Guest:rw;Administrator;fix;Windows_Server_2003
# Windows 2008
c:\Windows\System32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,user:Guest:r;Administrator;fix;Windows_Server_2008_
c:\Windows\System32\drivers\etc\hosts;user:Administrator:rw,user:SYSTEM:rw,user:Guest:rw;Administrator;fix;Windows_Server_2008_
```

Figure: ACLs input file

We will now modify a field in this policy and check the result in the Mission Portal. The first two lines below `# Windows 2003` concern the file '`c:\WINDOWS\system32\drivers\etc\hosts`', lets change the action taken by CFEngine if the promise is not compliant.

Action item: Change the next to last field from '`fix`' to '`warn`'. Click the **Save** icon on the right, then **Commit** (add a comment in the pop up, for example 'Changed to warn'), click **Run now** and wait for the execution to have finished (can take up to a minute). To check the result, go back to the main page, enter the '`Engineering`' room (Section 2.4 [Engineering room], page 4), click the **CDP reports** icon and finally click **ACLs File access controls** in the '`CDP reports`' finder.

The result should look like the following figure (note that the value in the 'Action' column says 'warn' instead of 'fix').


ACLs

HOST	PATH	PERMISSION (ACL)	OWNER	ACTION	CLASS EXPRESSION	STATE	LAST CHECKED
windows2003-1.pilot.cfengine.com	c:\WINDOWS\system32\drivers\etc\hosts	'user:Administrator:rw','user:SYSTEM:rw','user:Guest:r'	SYSTEM	warn	Windows_Server_2003.!Hr09	Compliant	Mon Jan 23 13:58:36 2012

Figure: ACLs report

A.3 Example - Create a standard policy

If you have not already opened the Policy Editor, follow the instruction in Section A.1 [The Policy Editor (beta)], page 23 to do so. We will create a custom report as an example of writing a standard CFEngine policy. Custom reports are useful when your reporting needs differ from the CFEngine 3 Nova standard reports. Data processing and extraction from CFEngine's embedded databases must be scripted by the user if the procedure is not covered in any of the reports found in the Mission Portal. Output to files or logs may be customized on a per-promise basis and users can design their own log and report formats.

Action item: Click **New** in file menu in the right column of the policy editor, a tab ('Untitled-1') will appear in the center, and enter the following in the text space (you can always hit **Ctrl** + **h** to see a list of available keyboard shortcuts).

```
bundle agent pilot_simple_custom_report_policy {

   # promise type: make a promise about variables, see details below
   vars:
      "file_to_check" string => "/tmp/somefile.txt";

   # promise type: promise about classes (context), see details below
   classes:
      "file_exists" expression => fileexists("$(file_to_check)");

   # promise type: make a promise about reports, see details below
   reports:
      !file_exists::                # context in which the promise should be executed
                                    # in this case if the file does not exist
         "WARNING: File $(file_to_check) does not exist on host $(sys.uqhost)"
            handle  => "pilot_simple_report_policy",
            comment => "Simple reports policy to show up in Mission Portal";
}
```

The astute reader will remark that there is no '`body common control`' statement in the above policy. The reason is that we will use this bundle in the global policy '`promises.cf`', which already contains the compulsory `body common control`. We can therefore omit that statement here. This policy consists of a bundle with three main parts:

1. Variable definition: Define a variable called `file_to_check`, of type `string`, containing a value that represents a file name ('`/tmp/somefile.txt`').
2. Class definition: check whether the file exists through the CFEngine function `fileexists` and set a class (boolean) called `file_exists` based on the result.
3. Report definition: Generate a report if the file does not exist (consists of a warning; it uses the `file_to_check` and `sys.uqhost` variables to display the appropriate file- and host names, respectively.

Action item: Save this file as '`pilot_simple_custom_report_policy.cf`': click **Save** in the right menu and enter the file name.

We now need to add the policy to '`promises.cf`' to verify the results in the Mission Portal:

Action item:

1. Open '`promises.cf`' by clicking it in the policy listing on the left
2. Uncomment the corresponding line by removing the hash tag (#) in front of it (only if you have created and saved the file as instructed above)

```
    "bundlesequence" slist => {
                   "pilot_simple_edit_policy"
#                  "pilot_simple_custom_report_policy",    #Uncomment this line
                              };
    "inputs"         slist => {
                   "pilot_simple_edit_policy.cf"
#                  "pilot_simple_custom_report_policy.cf", #Uncomment this line
                              };
```

3. Save '`promises.cf`'
4. Run syntax check by clicking **Check syntax** on the right
5. If everything is fine, commit the changes by clicking **Commit** on the right (you will be prompted for a comment, enter for example 'Added custom report')

The policy will have been adopted and executed at the next CFEngine run (every five minutes by default). Again, you may execute the policy immediately by clicking the **Run now** button in the right column (this might take a while, please be patient and wait until the execution is finished before checking the reports for updates). You can check that the policy has been run by searching for its handle in a report.

Action item: Open the '`Report finder`' as shown previously, scroll to and click **Compliance by promise**, enter the handle ('`pilot_simple_custom_report_policy`') in the '`By handle`' query field and click **Generate**.