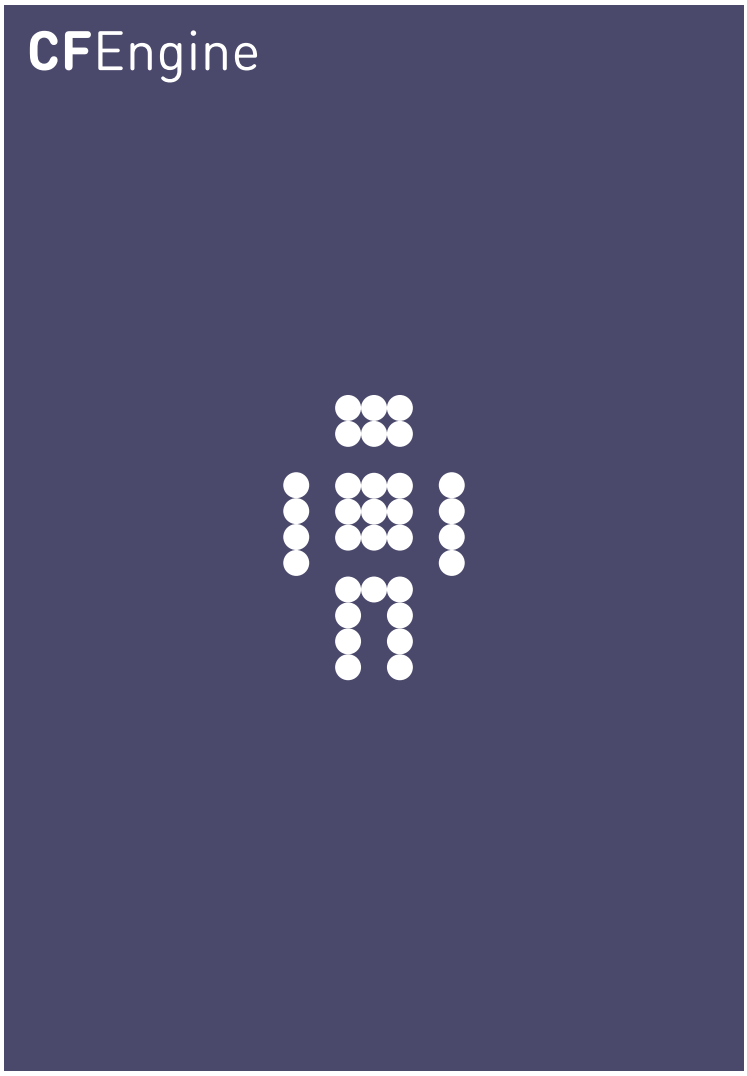


CFEngine 3 Nova Pilot Handbook

CFEngine Enterprise Documentation

Updated 5. January 2011

CFEngine

Table of Contents

1 Introduction

CFEngine comes in three software editions: CFEngine Community, CFEngine 3 Nova, and CFEngine Constellation[1], each designed for different types of users and budget levels. 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.

This pilot 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 interphase. 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. We will look at some common concepts in the following sections and allow the user to quickly:

- View standard reports
- Create (query) reports
- Modify a Content Dependent Policy (CDP) datalist, and verify the result
- Edit a generic policy and verify the result
- Create a generic policy and verify the result

For the purpose of this pilot 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 flavours and windows servers.

A brief introduction to terms used in the following pages: 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`'.

[1] Coming soon.

2 Get started

You should have received an email with account information after making a pilot agreement with CFEngine. In it, you will find information on how to access online resources, the CFEngine support ticketing system, and a dedicated CFEngine 3 Nova instance. 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 Nova Mission Portal is divided into four main sections called *rooms*. Each of these offer 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 bank that connects information together

You can always use the breadcrumb in the top left corner to navigate and see where you are in the Mission Portal. We will now enter each of the rooms to briefly see what is inside.

2.1 Status room

The '`Status`' room provides a high level summary of how well the entire system is behaving in a manner that is understandable for non-technical people. Click the **Status** icon in the main page of the Mission Portal to enter the '`Status`' room:

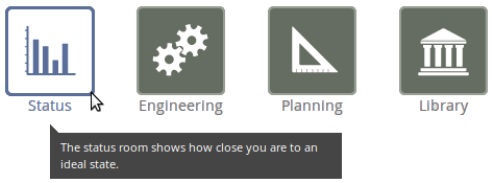


Figure: Got to Status room

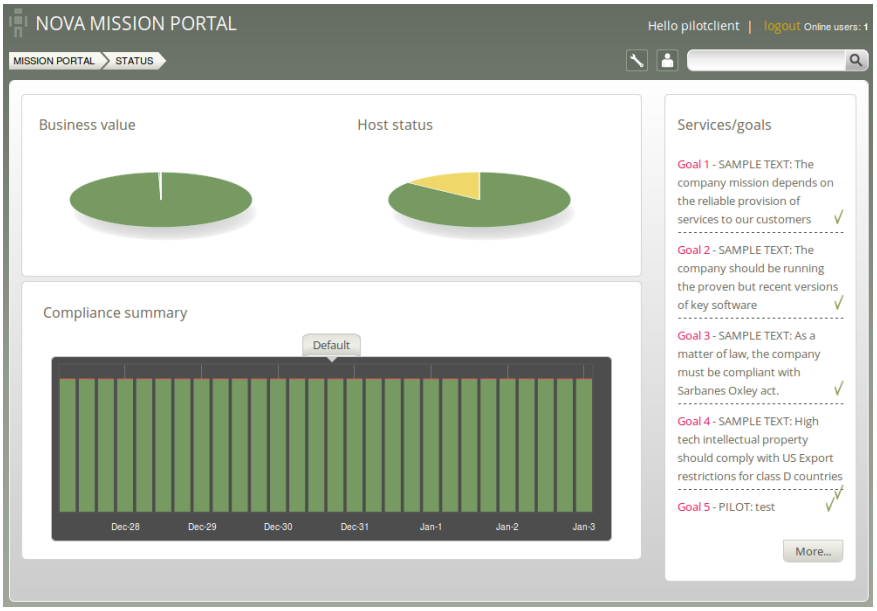


Figure: Status room

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. A host is red if less than 80% of its promises are kept, yellow if 20% or more of its promises were repaired and host is now compliant, green if more than 80% of its promises are kept, and blue if there is no contact between the hub and the client host (unreachable).

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.

Services/Goals: A summary of Mission goals as defined in user policy files (these examples are from '`company_knowledge.cf`'). We will look at editing policy files in later sections.

We will now have a look in the '`Engineering`' room: click the **Back** button in your browser, or **MISSION PORTAL** in the breadcrumb, to go back to the main Misson Portal page.

2.2 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. Click the **Engineering** icon in the main page of the Mission Portal to enter:

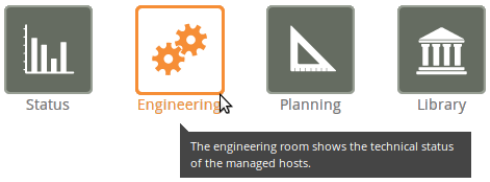


Figure: Got to Engineering room

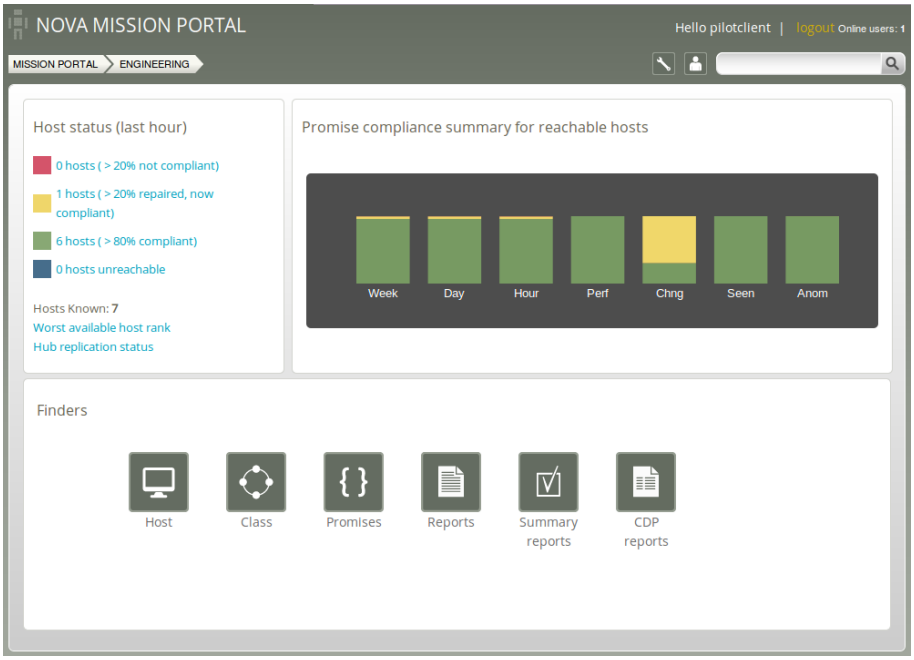


Figure: Engineering room

Host Status:

- The hosts are classified into red, yellow, green and blue according to the status of their compliance. A host is red if less than 80% of its promises are kept (> 20% not compliant), yellow if 20% or more of its promises were repaired and host is now compliant (> 20% repaired, now compliant), green if more than 80% of its promises are kept (> 80% compliant), and blue if there is no contact between the hub and the client (host unreachable). 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 weakest hosts (that have been in contact with the hub) over the last hour.
- Hub replication status: Display status of redundant monitoring hubs (not activated in pilot environment).

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.

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 pilot document, but feel free to explore the different finder functions on your own.

We will now have a look in the '`Planning`' room: click the **Back** button in your browser, or **MISSION PORTAL** in the breadcrumb, to go back to the main page.

2.3 Planning room

The '`Planning`' room allows you to make changes to policies, goals determined by promises 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. Click the **Planning** icon in the main page of the Mission Portal to enter.

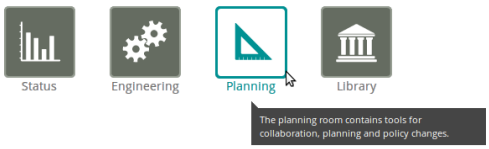


Figure: Got to Planning room

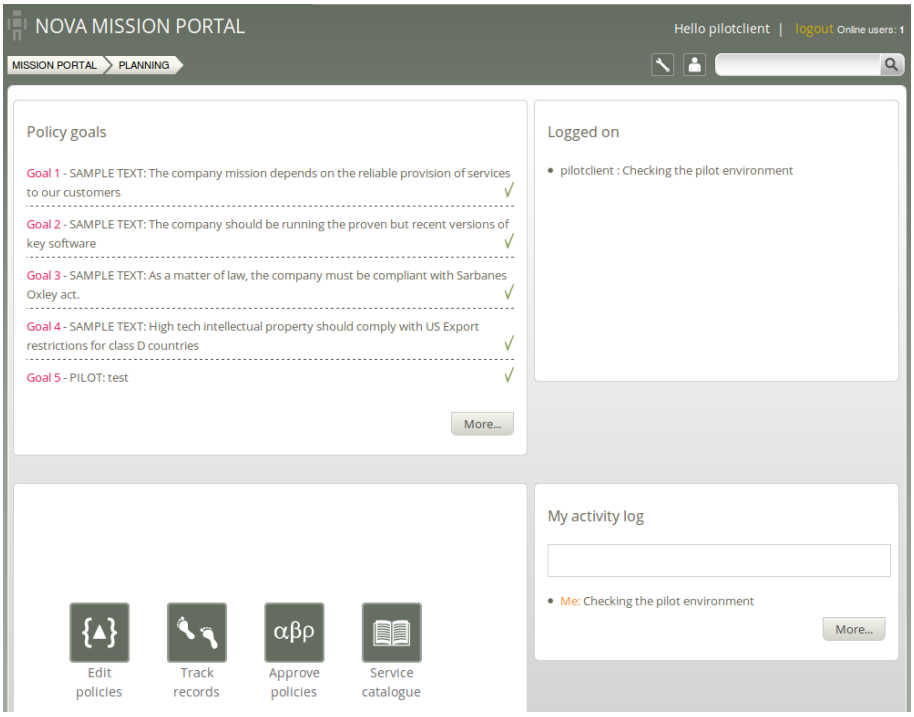


Figure: Planning room

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

Action icons:

- Edit policies: Edit policy files in the integrated policy editor
- Track records: Overview of promises repaired or not kept
- Approve policies: To be developed
- Service catalogue: See which bundles contribute to policy goals

Logged on: Shows users currently logged on to the Mission Portal and their activity.

Activity log: Shows the latest activity entries. Type in a new activity to keep colleagues posted on current work.

Finally, we will have a look at the '`Library`' room: click the **Back** button in your browser, or click **MISSION PORTAL** in the breadcrumb, to go back to the main page.

2.4 Library room

The Library contains finders for documents, topics, a notes archive, and (external) link to the CFEngine community. Click the **Library** icon in the main page of the Mission Portal to enter.

Figure: Got to Library room

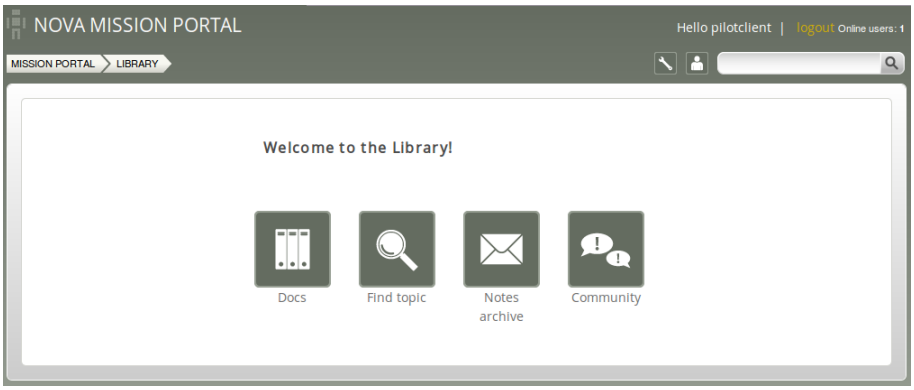


Figure: Library room

- 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 overview of all notes made by Mission Portal users in regard to hosts or reports.
- Community: External link to the CFEngine community

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.

Standard reports in CFEngine 3 Nova can be accessed through the '`Reports finder`': enter the '`Engineering`' room, locate and click the **Reports** icon to open the finder.

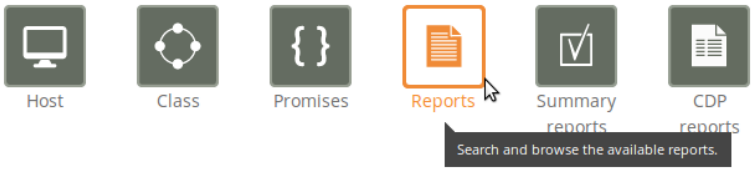
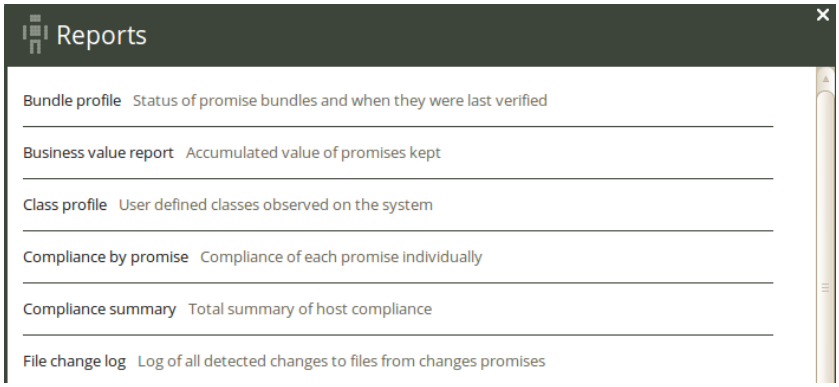


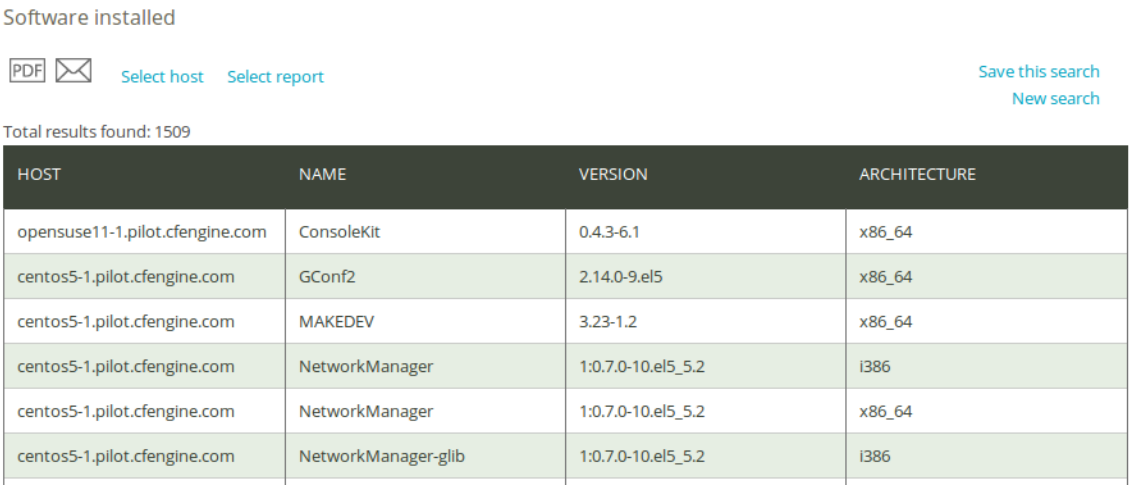
Figure: Click to open the Reports finder.



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. As an example, scroll down and click **Software installed** towards the bottom of the '`Report finder`' to open a query window:

Figure: Software installed query

Click **Generate report** in the query window without filling in any of the search fields. This 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.



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: 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):

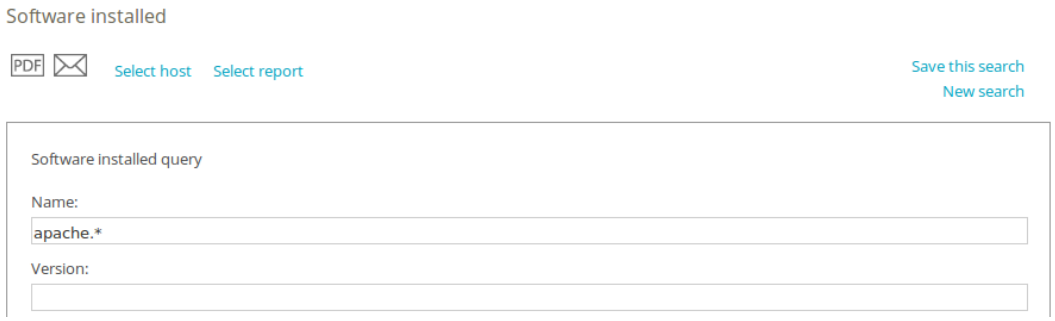


Figure: Narrow the search by entering search criteria

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 at `http://cfengine.com/<address TBD>`.

4 Content Driven Policy (CDP) reports

Content-Driven Policies (CDP) 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 results in a specific type of promise being made. 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. To access CDP reports, enter the '`Engineering`' room from the Mission Portal, then click the **CDP reports** icon:

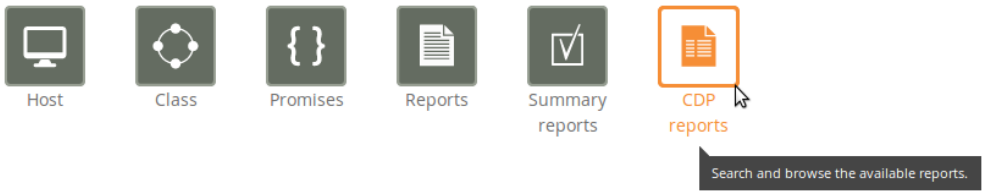


Figure: Go to CDP reports finder

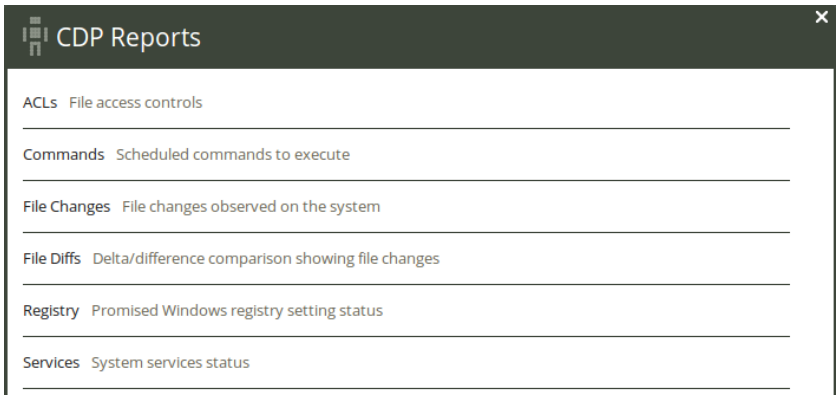


Figure: CDP reports finder

CFEngine 3 Nova comes with several default CDPs, we will use the access control list 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.

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

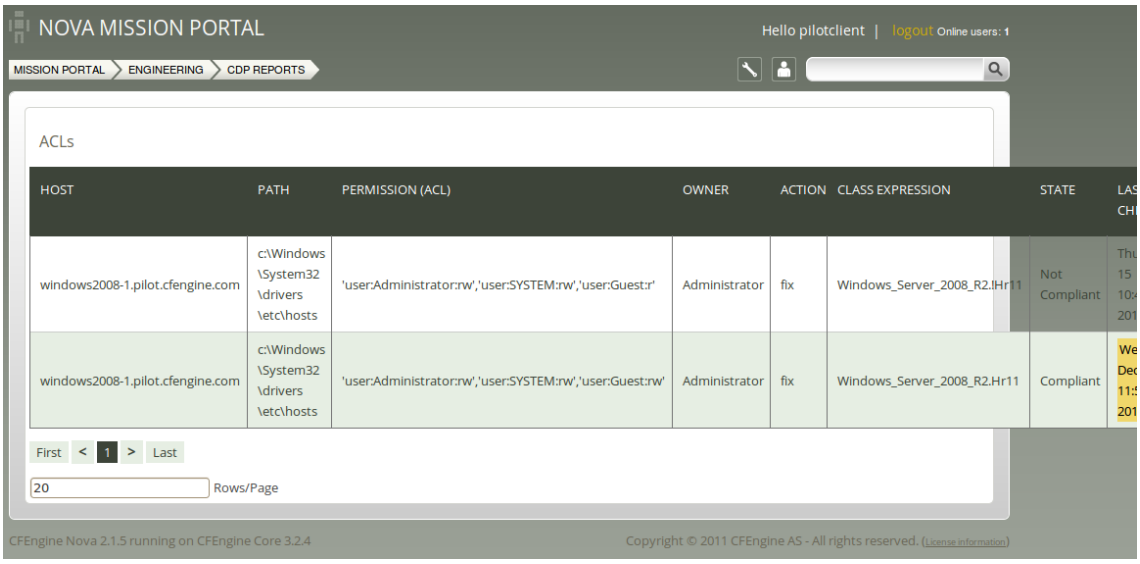


Figure: ACLs report (bug in pilot env)

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 Working with CFEngine policies

5.1 The policy editor

CFEngine 3 Nova has an integrated editor for working on CFEngine policies, providing syntax highlighting and look-up to make policy writing easier. To access the policy editor, enter the '`Planning`' room from the front page (as shown previously), then click the **Edit policies** icon:

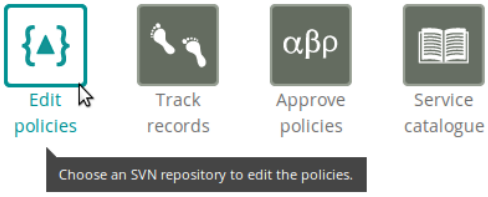


Figure: Go to Edit policies

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. Use the path '`http://localhost/svn/CFEnginePilot`', username '`pilot`', password '`pilot`' and click **Add**:

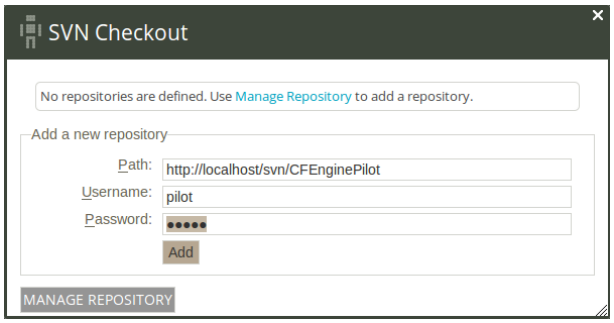


Figure: Add SVN repository

The following screenshot will appear after the Subversion repository has been added. Click on **Checkout** to complete the checkout procedure.

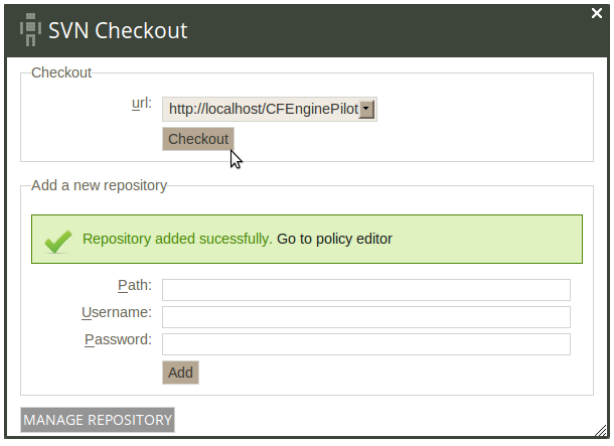


Figure: SVN checkout

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.

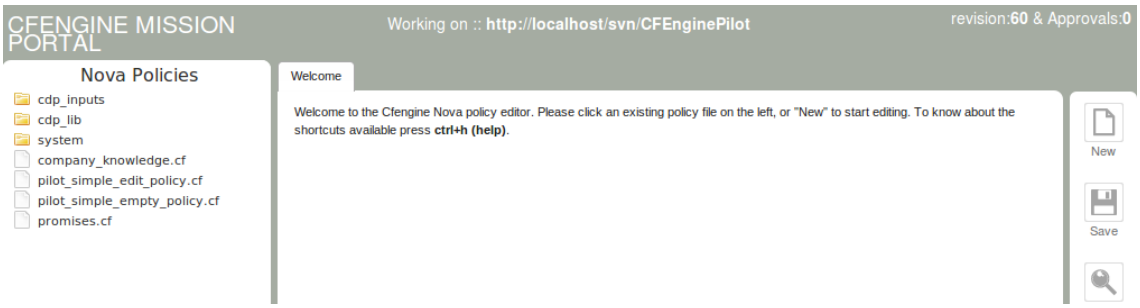


Figure: The Policy Editor

5.2 Edit a CDP input file

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 (this is necessary since their meaning depend on the policy type). We will look at the ACLs input file as an example. The default CFEngine 3 Nova ACLs policies allow you to set permissions to directories and files using two different input files ('`acl_directory_list.txt`' and '`acl_file_list.txt`', respectively). We will limit ourselves to manipulate one of these as they are conceptually identical.

The file '`acl_file_list.txt`' can be found under the '`cdp_inputs`' catalog in the policy editor, click it to open:

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 saw earlier that the input consisted of lines containing semi colon separated fields, so anything with a ';' before or after it is a field entry. We need to look at the header of the file to understand its structure, the input fields are explained in the `FORMAT` section. 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.

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: 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, 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`').

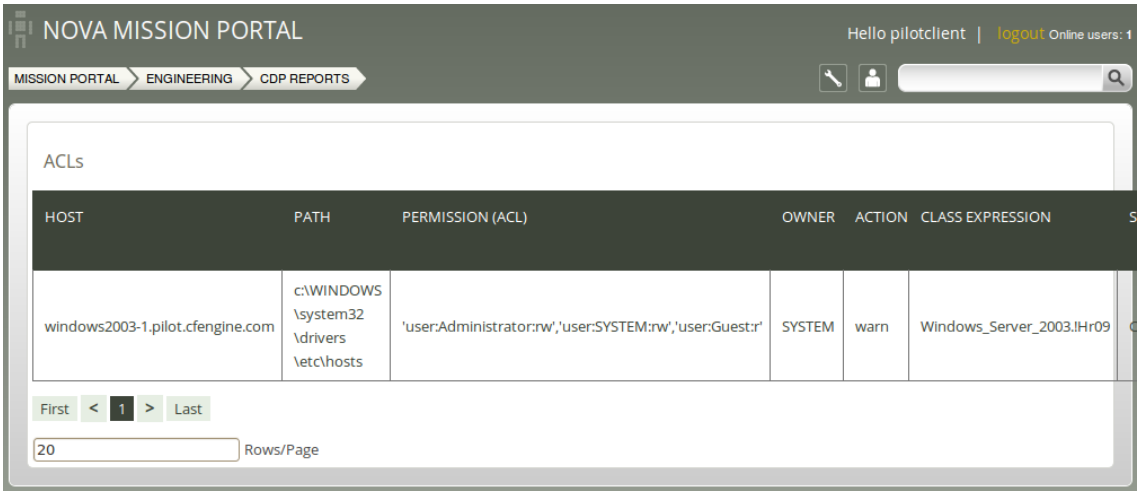


Figure: ACLs report (bug in presentation)

5.3 CFEngine syntax

As mentioned in the introduction, 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 will create the file '`/tmp/testfile`'. It contains the bare minimum of a CFEngine policy, consisting of the compulsary '`body common control`', containing (at least) a bundlesequence. Then follows a compulsary '`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 a promise about desired state (here '`create => "true"`').

5.4 Example policy: Create a custom report

Custom reports are a useful tool 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. We will go through a simple example of this by creating a new policy: 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_should_exist" expression => fileexists("$(file_to_check)");

   # promise type: make a promise about reports, see details below
   reports:
      !file_should_exist::         # 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 compulsary `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_should_exist` 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.

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:

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 them

```
    "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 promted 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 immediatly 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: 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**.