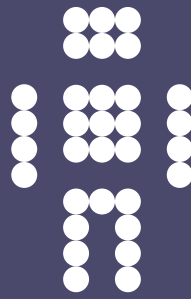


CFEngine



Windows Management with CFEngine Nova

A CFEngine Special Topics Handbook

CFEngine AS

CFEngine Nova is a cross-platform and versatile tool that unifies the desired state management on all major operating systems.

However, some operating systems are designed fundamentally different than others, requiring special CFEngine language features when being described. In this document, we highlight the Nova extensions provided to the Windows platform.

Table of Contents

System requirements	1
Installation	1
Testing policies locally	1
Windows registry management	2
Creating values	2
Removing values	3
Windows service management	3
File and folder permissions	4
Windows-aware features in CFE Nova	5
Windows special variables	7
Windows hard classes	7
Notes on windows policies	7

System requirements

CFEngine Nova, being so lightweight, works equally well on Windows clients as on Windows servers. Both native 32-bit/x86 (package name i686) and 64-bit/x64 (package name x86_64) packages are available to customers. It is important that you select the x86_64 package if you are running 64-bit Windows.

Of Windows client operating systems, anything from Windows XP SP2 and newer is supported. On the server side, Windows Server 2003 and newer is supported.

CFEngine Nova communicates bi-directionally on port 5308, so make sure that this port is open for outgoing and incoming TCP connections.

All software dependencies are bundled with the CFEngine Nova package. The total disk consumption is about 70 MB, and the memory usage is less than 30 MB.

Installation

The installation and set-up procedure on Windows is not different than that for other operating systems Nova runs on, so the Nova getting started document available at <http://software.cfengine.com> applies to the Windows version as well.

The Windows msi-packages will get silently installed (no prompts) to Cfengine under your program files directory (e.g. C:\Program Files\Cfengine on English Windows versions). It is important that the installer is run with Administrative privileges. To ensure this, open a Command Prompt in Administrative mode and run 'msiexec -i cfengine-nova-VERSION-ARCH.msi' (replace VERSION and ARCH appropriately).

Note that you need a policy hub to bootstrap to in order for the license to get downloaded and verified. This is required even if you just want to run some tests on the local machine manually. The policy hub needs to run a flavour of Linux, as noted in the Nova getting started document. Find the hostname or IP address of the hub, here we assume the address is '123.456.789.123' (do not bootstrap with a localhost address):

```
C:\Program Files\Cfengine\bin\cf-agent.exe --bootstrap --policy-server 123.456.789.123
```

Testing policies locally

After correctly bootstrapping, however, you can stop the CFEngine Nova service (the cf-execd process), and kill cf-serverd and cf-monitord to create an off-line test environment. You can also delete everything under Cfengine\inputs except license.dat and cfengine_stdlib.cf to test local policies manually.

Create a new text file Cfengine\inputs\promises.cf and input the following text using your favourite text editor.

```
body common control
{
bundlesequence => { "test" };
inputs => { "cfengine_stdlib.cf" };
host_licenses_paid => "1";
}
```

```
bundle agent test
{
reports:
windows::
    "Hello, Windows!";
}
```

Now, go to your terminal (e.g. Command Prompt) and navigate to `Cfengine\bin` under program files. Run `cf-promises.exe`. It should generate no output, which indicates correct syntax and license.

To execute the policy, run `cf-agent.exe -K`. You should see the following output.

```
C:\Program Files\Cfengine\bin>cf-promises.exe
C:\Program Files\Cfengine\bin>cf-agent.exe -KI
R: Hello, Windows!
C:\Program Files\Cfengine\bin>
```

We now have a basic skeleton policy that we can test our Windows promises with. These can later be integrated at the policy hub to ensure that they are run on all Windows systems. We will assume this general skeleton for the rest of this document, modifying the contents of the test bundle only.

Windows registry management

CFEngine Nova supports fine-grained management of the Windows registry. These promises are encapsulated under the `databases: promise` type.

Creating values

Let us modify our skeleton bundle to contain the following.

```
...
bundle agent test
{
databases:

    "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security"

    database_operation => "create",
    database_rows => { "MaxSize,REG_DWORD,84017152", "Retention,REG_DWORD,0"},
    database_type => "ms_registry",
    comment      => "Ensure eventlog size is in accordance with standards";
}
```

Now, we again run `cf-promises.exe` to ensure the syntax is correct, followed by `cf-agent.exe -KI`. Note that we added the `-I` option which tells `cf-agent.exe` to notify us on the existing state of the system and any actions done to ensure the desired state. The output should look like the following.

```
C:\Program Files\Cfengine\bin>cf-promises.exe

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
-> Repairing registry REG_DWORD value as <MaxSize,84017152>
-> Verified value <Retention,0> correct for HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
-> Verified value <MaxSize,84017152> correct for HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security
-> Verified value <Retention,0> correct for HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security

C:\Program Files\Cfengine\bin>
```

When we run `cf-agent.exe` twice, the second run will do nothing because the first run has already corrected the value. This is convergence — CFEngine is ensuring the desired state.

Removing values

In order to remove values instead, we just need to adjust the policy slightly, resulting in the following bundle.

```
...
bundle agent test
{
  databases:

    "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security"

    database_operation => "delete",
    database_columns   => { "value1", "value2"},
    database_type      => "ms_registry",
    comment            => "Remove stray values generated by an application";
}
```

Now, if you create 'value1' and 'value2' in the key above, `cf-agent.exe` should show the following output.

```
C:\Program Files\Cfengine\bin>cf-promises.exe

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
-> Registry value "value1" in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security deleted...
-> Registry value "value2" in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security deleted...

C:\Program Files\Cfengine\bin>cf-agent.exe -KI
-> Registry value "value1" in "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security" already gone
-> Registry value "value2" in "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Security" already gone
```




At the time of writing, Nova supports the `REG_DWORD` (double word), `REG_SZ` (string) and `REG_EXPAND_SZ` (expandable string) data types, as given in the middle field of the `database_rows` list elements. See the [CFEngine reference manual](#) for an updated list of supported data types.

Also note the `registryvalue()` function which can be used to read out value data from the registry and act upon it. Examples of its use are also available in the [CFEngine reference manual](#).

Windows service management

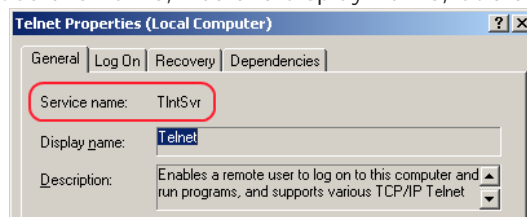
CFEngine Nova can maintain complete control of the state of all Windows services. For example, services prone to security issues or errors can easily be given a disabled state.

```
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).

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.



A complete example of a service management bundle is show below.

```
...
bundle agent test
{
services:

windows::
  "W32Time"
    service_policy => "start",
    service_method => bootstart,
    comment => "Ensure important services are running and starting at boot";

Windows_Server_2008::
  "RemoteRegistry"
    service_policy => "disable",
    service_method => force_deps,
    comment => "Disable services that create security issues";
}
```

This example ensures that the Windows Time service is running on all Windows hosts, and that Remote registry is disabled on all Windows 2008 servers.

File and folder permissions

CFEngine Nova can ensure the permissions or Access Control Lists (ACLs) of your Windows systems are correctly set. Windows ACLs are a complex topic by itself, with support for more than ten different permission bits and inheritance. Nova supports all of this, but we will just cover the basics in this document.

The following policy will ensure strict permissions on a directory 'C:\Secret' and a file 'C:\Secret\file.txt'.

...

```
bundle agent test
{
  vars:
    "acl_secret_dir" slist => { "user:Administrator:rx:allow",
                                "group:Administrators:rx:allow" };
    "acl_secret_file" slist => { "user:Administrator:rw:allow" };

  files:

  windows::
    "C:\Secret",
      acl => ntfs( "@(acl_secret_dir)" ),
      depth_search => include_base,
      perms => owner( "Administrator" );

    "C:\Secret\file.txt",
      acl => ntfs( "@(acl_secret_file)" ),
      perms => owner( "Administrator" );
}
```

The [CFEngine reference manual](#) contains a description of all the available ACL options. Also refer to the the CFEngine Nova Owner's manual for a more in-depth discussion of the ACL options available.

Windows-aware features in CFE Nova

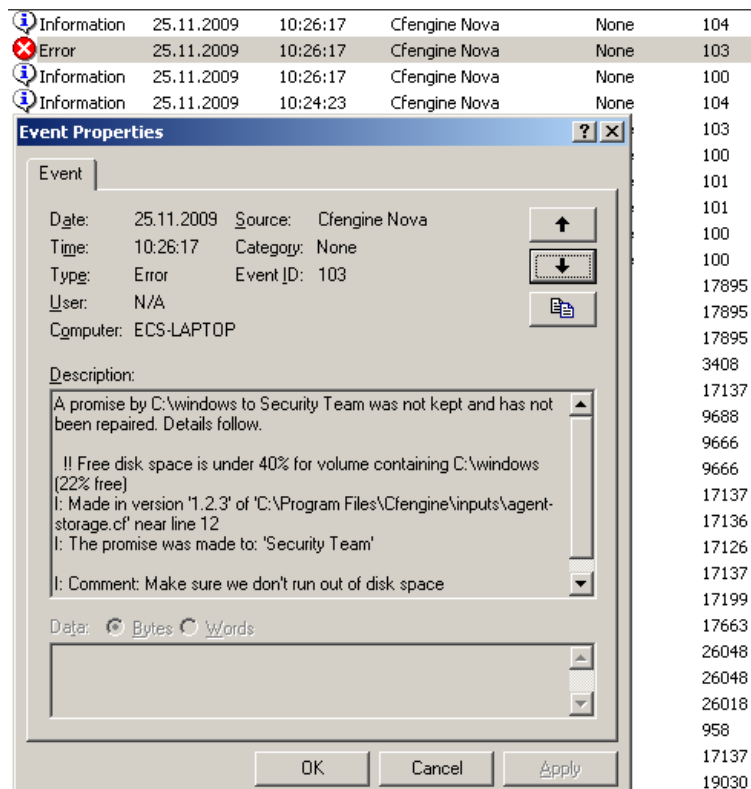
CFEngine Nova integrates with the Windows operating system in multiple ways.

The CFEngine scheduler in Nova (`cf-execd`) runs as a Windows service. This means it runs in the background and starts with Windows, before any user logs in. It can be configured, started and stopped from the "Services" listing in Windows, just like any other Windows service.

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. The following event ids are defined in CFEngine Nova, allowing categorisation of the log message based on its type. The Nova event logs can be found under the "System" logs.

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



By default, only promise not repaired and generic error events are logged to avoid flooding the Event Log. You can turn on verbose logging to log all messages, like the following example.

```
body common control
{
inputs => { "cfengine_stdlib.cf" };
bundlesequence => { "main" };
}

bundle agent main
{
files:
"C:\test.txt"
  create => "true",
  action => log_verbose;
}
```

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.

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`.

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 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
{
inputs => { "cfengine_stdlib.cf" };
bundlesequence => { "main" };
}

bundle agent main
{
commands:

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

Finally, one should note that Windows lacks support for certain features that are utilised 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 [CFEngine reference manual](#) documents exactly which promises are ignored and not. Also, `cf-agent.exe` from CFEngine Nova prints warning messages on ignored attributes when run in verbose mode (`cf-agent.exe-Kv`).