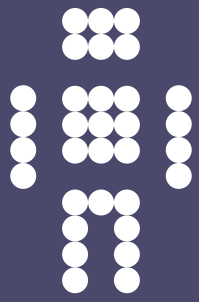
CFEngine



Get started with CFEngine

A CFEngine Handbook

CFEngine AS

This guide provides the first steps in understanding and writing your own CFEngine policy files. It is recommended that you have a general understanding of CFEngine concepts before proceeding (see CFEngine 3 Concept Guide: http://cfengine.com/manuals/cf3-ConceptGuide.html).

Table of Contents

1 Getting started with CFEngine

1.1 Prerequisites

This guide is assumes that CFEngine has been correctly installed on your system (see installation guide: http://cfengine.com/manuals/cf3-installation.html. We recommend you that have a basic understanding of CFEngine concepts before proceeding (http://cfengine.com/manuals/cf3-conceptguide.html), but thanks to CFEngine's clear syntax you can also get a fair understanding by starting directly with these examples.

1.2 The work directory

CFEngine operates with the notion of a work-directory. The default work directory for the `root` user is `/var/cfengine`. For any other user, the work directory lies in the user's home directory, named `~/.cfagent`. CFEngine prefers you to keep certain files here, you should not resist this too strongly or you will make unnecessary trouble for yourself. Most experiments can be safely tested as an ordinary user. You should spend some time experimenting with small examples before setting out to configure a system as root. To do that you should log onto your system as a regular unprivileged user and copy the binaries into the work directory:

```
host$ mkdir -p ~/.cfagent/bin
host$ cp /usr/local/sbin/cf-* ~/.cfagent/bin
```

You can now test the software and play with self-written experiments.

1.3 Example template

To begin with we will provide entire example policy files for clarity, but as we go along we will drop the parts that are purely repetitive. In such cases we will abide by the following template, you may test by entering the example as indicated and run CFEngine:

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


bundle agent test
{

   # example goes here

}
```

2 How to execute and test a CFEngine policy

2.1 Hello world

Here is the simplest 'Hello world' program in CFEngine 3:

```
# Every policy must have a bundlesequence

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

#

bundle agent test
{
reports:             # This is a promise type

 cfengine_3::        # This is a class context (the promise will only
                     # be kept on a CFEngine_3 system)

  "Hello world";   # This is a simple promise (it generates a report
                   # that says "Hello world")
}
```

Type this in to a file, e.g. '`emacs ~/test.cf`'. Then check the syntax like this

```
/usr/local/sbin/cf-promises -f ~/test.cf
```

If all is well there should be no output. Now execute as follows:

```
/usr/local/sbin/cf-agent -f ~/test.cf
```

You should see this:

```
R: Hello world
```

The '`R:`' tells you this is the output from a report (as opposed to a log '`L:`', or the quoted output of some embedded program '`Q:`').

This is not a typical CFEngine program, primarily because CFEngine is not normally meant to print messages except in exceptional circumstances. As a starter however, it is reassuring to see some output.

If you repeat the command immediately nothing will happen. But if you wait a minute, it will work again. Run the command in verbose mode (use the `-v` or the `--verbose` switch) to see why:

```
/usr/local/sbin/cf-agent --verbose -f ~/test.cf
```

Now you will see:

```
cf3>    ------------------------------------------------------------
cf3>    reports in bundle hello (1)
cf3>    ------------------------------------------------------------
```

```
cf3>
cf3>  XX Nothing promised here [lock.hello.reports..Hello_worl] (0/1
minutes elapsed)
cf3>
```

This tells you that CFEngine believes it is too soon to try to keep this promise again. The time it sets on this is determined by the `ifelapsed` parameter, which can be set individually for every promise. You can also ask CFEngine to ignore these locks using the '`-K`' option.

Before the 'Hello world' string, you see the class expression '`cfengine_3::`'. This is how CFEngine makes decisions. The promise to print the message will only apply if this condition is true. To see that this class is true for the execution, look at the verbose output from the command you just typed. You will see something like this:

```
Defined Classes = ( any verbose_mode Tuesday Hr08 Morning Min48
Min45_50 Q4 Hr08_Q4 Day7 July Yr2009 Lcycle_2 GMT_Hr6 linux atlas
undefined_domain 64_bit linux_2_6_27_23_0_1_default x86_64
linux_x86_64 linux_x86_64_2_6_27_23_0_1_default
linux_x86_64_2_6_27_23_0_1_default__1_SMP_2009_05_26_17_02_05__0400
compiled_on_linux_gnu localhost_localdomain localhost net_iface_lo
net_iface_wlan0 ipv4_192_168_1_100 ipv4_192_168_1 ipv4_192_168
ipv4_192 fe80__21c_bfff_fe6e_70ef CFEngine_3_0_2b4 CFEngine_3_0
CFEngine_3 SuSE lsb_compliant suse suse_n/a suse_11_1 suse_11
agent )
```

i.e. a list of all the currently defined classes. Any one of these classes (or a combination) could have been used to label the promise. That is the way CFEngine points to which promises will be kept in which scenarios.

A final thing to note: if you try to process this using the '`cf-promises -r`' command, you will see something like this:

```
atlas$ ~/LapTop/CFEngine3/trunk/src/cf-promises -r -f ~/test.cf
Summarizing promises as text to ~/test.cf.txt
Summarizing promises as html to ~/test.cf.html
```

The '`-r`' option produces a report. Examine the files produced:

```
cat ~/test.cf.txt
firefox ~/test.cf.html
```

You will see a summary of how CFEngine interprets the files, either in HTML or text. All the CFEngine components will produce debugging file with an expanded view when using this option (e.g. for the configuration file named '`promise_output_agent.h`', they will create the files '`promise_output_agent.html`' and '`promise_output_agent.txt`').

Until now we have tested examples from a file in your home directory (`~/`). It is of course possible to place and execute files anywhere, but it is good practice to keep a separate directory for your CFEngine policy files. The default directory is `~/.cfagent/inputs/` for regular users, and `/var/cfengine/inputs/` for root. In the following we assume that you work from the default input directory.

2.2 Create a file

We will be using the CFEngine Community Open Promise-Body Library (cfengine_stdlib.cf) in the following example. This is a library of definitions that can be obtained from the CFEngine website (most recent version) or found in '`/usr/local/share/cfengine/masterfiles`'. It should be included in the '`inputs`' directory and used as shown in the example below. To copy the file to '`inputs`':

```
cp /usr/local/share/cfengine/masterfiles/cfengine_stdlib.cf ~/.cfagent/inputs
```

Go to the default input directory (e.g. `cd ~/.cfagent/inputs`). Type in the following example in '`~/.cfagent/inputs/test.cf`':

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

bundle agent test
{
files:

# This is a throw-away comment, below is a full-bodied promise

"/tmp/testfile"                      # promiser

    comment => "This is for keeps",   # Live comment
    create => "true",                 # Constraint 1
    perms => m("612");               # Constraint 2, rw---x-w-

}
```

The template '`m`' is defined within '`cfengine_stdlib.cf`':

```
# This is a trivial body template, which makes parameterizing
# the promise body tidier and re-usable

body perms m(x)
{
mode => "$(x)";
}
```

This example shows how additional attributes are added to the body of the promise. The right hand side of the `perms` declaration is a template which we have called '`m()`', which uses a parameter. The template is defined below the bundle of promises that uses it, showing how we can create re-usable sets of parameters. In this case, the example is trivial, but we have barely begun. When things get more sophisticated, we shall hide a huge amount of detail in these parameters, thus keeping the main promise uncluttered and its intention clear.

> In every 'promise constraint' of the form '`left-hand-side => right-hand-side`', the left hand side is a CFEngine reserved word, and the right hand side is a decision you make, possibly expressed in terms of standard templates.

Now execute `cf-agent` with this promise:

```
host$ /usr/local/sbin/cf-agent -f test.cf -I
Couldn't find a private key (/home/geir/.cfagent/ppkeys/localhost.priv) - use cf-key to get one
 !!! System error for fopen: "No such file or directory"
 -> Created file /tmp/testfile, mode = 612
```

The '-I' flag tells CFEngine to 'inform' us about changes only. This provides a digestible amount of output that is more than the default (which is to only report un-fixable problems or explicit reports). Despite the error message,[1] we see that CFEngine creates the file as ordered, and sets the permissions appropriately. Look for the file that was that was created and see what happens when permissions are changed:

```
host$ ls -l /tmp/testfile
-rw---x-w- 1 mark users 33 2009-06-30 06:06 /tmp/testfile

host$ chmod 400 /tmp/testfile

host$ ls -l /tmp/testfile
-r-------- 1 mark users 33 2009-06-30 06:06 /tmp/testfile

host$ /usr/local/sbin/cf-agent -f test.cf -I
-> Object /tmp/testfile had permission 400, changed it to 612

host$ ls -l /tmp/testfile
-rw---x-w- 1 mark users 33 2009-06-30 06:06 /tmp/testfile
```

Once again, remember the comment about locking and `ifelapsed` from the previous example.

Notice that this promise does not have a class expression like `cfengine_3::`. The default class `any::` applies if nothing is stated, which means 'anytime, anyplace, anywhere'.

2.3 Create a directory

Creating a directory is not much different from creating a file, suffice to replace the file name with a directory name in the test bundle (inside `bundle agent test`; keep the rest of '`test.cf`' as is):

```
bundle agent test
{

files:

"/tmp/test_dir/."                          <--- This is the only change!
    comment => "Creating dir...",
    create => "true",
    perms => m("612");

}
```

[1] Encryption keys are user-specific and CFEngine complains since none have been created for users that are not root. This is not important in the setting of these examples, but you will have to set up a key if you wish to communicate with other machines as a non-root user under CFEngine management (see http://cfengine.com/manuals/cf3-reference.html#Remote-access-troubleshooting). Type cf-key to create keys and avoid seeing this error message in the future.

2.4 Copy a file

To copy the contents of our CFEnginge test policy file 'test.cf' to 'testfile':

```
files:

   "/tmp/testfile"

      copy_from => local_cp("$(sys.workdir)/inputs/test.cf");
```

2.5 Copy directory trees

```
files:

  "/tmp/test_dir"

    copy_from => local_cp("$(sys.workdir)/bin/."),
    depth_search => recurse("inf");
```

2.6 Edit password files

To change root password of a system, we need to edit a file. A file is a complex object – once open there is a new world of possible promises to make about its contents. CFEngine has bundles of promises that are specially for editing. Make a copy of a shadow file and copy it to '/tmp' so that you can play with it. We will use the sudo command to temporarily act as root in order to do this (you will be prompted for the root password), <OWNER> is your username, <GROUP> is your user group (often the same as your username on ubuntu, or users on other systems)):

```
host$ sudo cp /etc/shadow /tmp
host$ sudo chown <OWNER>:<GROUP> /tmp/shadow
```

Type the following into your test bundle:

```
files:

"/tmp/shadow"
       comment => "Set the root password",
     edit_line => set_user_field("root",2,"xyajd673j.ajhfu");
```

Thanks to CFEngine's easy-to-understand syntax this is all we need to see on first inspection to understand the promise that is being made. Now execute cf-agent with this promise:

```
host$ ~/.cfagent/inputs$ /usr/local/sbin/cf-agent -f test.cf -I
 -> Setting field sub-value xyajd673j.ajhfu in /tmp/shadow
 -> Edited field inside file object /tmp/shadow
 -> Edited file /tmp/shadow
```

Have a look at the file to see that the changes have been made:

```
host$ cat /tmp/shadow
root:xyajd673j.ajhfu:15112:0:99999:7:::
daemon:*:15112:0:99999:7:::
...
```

2.7 Disabling and rotating files

Examples will become more generic from now, you may need to modifiy to adapt to local variables.

```
files:

  "/tmp/test_create"
      rename => disable;

  "/tmp/rotateme"
      rename => rotate("4");
```

2.8 Hashing for change detection (tripwire)

```
files:

  "/home/mark/tmp"  -> "me"
       changes      => detect_all_change,
       depth_search => recurse("inf"),
       action       => background;

  "/home/mark/LapTop/words" -> "you"
       changes      => detect_all_change,
       depth_search => recurse("inf");
```

2.9 Command or script execution

```
commands:

 Sunday.Hr04.Min05_10.myhost::

  "/usr/bin/update_db";

 any::
```

```
  "/etc/mysql/start"

      contain => setuid("mysql");
```

2.10 Kill process

```
processes:

  "snmpd"

     signals => { "term", "kill" };
```

2.11 Restart process

```
processes:

  "httpd"

     restart_class => "lift_off";

commands:

 lift_off::

    "/etc/init.d/apache2 restart";
```

Why? Separating this into two parts gives a high level of control and conistency to CFEngine. There are many options for command execution, like the ability to run commands in a sandbox or as 'setuid'. These should not be reproduced in `processes`.

2.12 Check filesystem space

```
storage:

  "/usr" volume  => mycheck("10%");
```

2.13 Mount a filesystem

```
storage:

 "/home/mark/server_home"
```

```
  mount => nfs("myserver","/home/mark");
```

2.14 Software and patch installation

```
packages:

  "apache2"

     package_policy => "add",
     package_method => generic;
```

3 An example bundle - update

The default CFEngine configuration contains a bundle of promises that copies the CFEngine binaries into the cache directory and copies the policy files from the server into the default location. This example is for local copying from file to file on the filesystem. Later, when we set up a server component, you will be able to copy from a remote host. This is a simple example of system provisioning, with automated update.

```
bundle agent update
{
vars:

# A standard location for the source point
"master_location" string => "/var/cfengine/masterfiles";

files:

"/var/cfengine/inputs"

  comment => "Update the policy files from the master",
  perms => u_m("600"),
  copy_from => u_cp("$(master_location)","localhost"),
  depth_search => u_recurse("inf");

"/var/cfengine/bin"

  comment => "Update the cached binaries from installation",
  perms => u_m("700"),
  copy_from => u_cp("/usr/local/sbin","localhost"),
  depth_search => u_recurse("2");
}
```

These promises contain several attributes in their bodies that we have not seen yet. The `copy_from` attribute tells CFEngine how to source (copy) a file from a master location. The `depth_search` tells it to search recursively through the sub-directories and their files.

Try changing the source files and executing the agent.

Again there are library reusable templates:

```
body perms u_m(p)
{
mode  => "$(p)";
}

#

body copy_from u_cp(from,server)
```

```
{
servers     => { "$(server)", "failover.example.org" };
source      => "$(from)";
compare     => "digest";
}

#

body depth_search u_recurse(d)
{
depth => "$(d)";
exclude_dirs => { "\.X11", ".*kde.*", "logs", "log" };
}
```

Here is an exercise: try using the reference manual to look up the elements in this example. See if you can understand all the parts.

4 Beyond Getting Started

You should now have a basic understanding of CFEngine and how it works. A good place to continue is the CFEngine Solutions guide (http://cfengine.com/manuals/cf3-solutions.html), which shows solutions to generic low- and high-level issues. You may skip the Introduction as it has been covered in this document.

We recommend the following reading:

- CFEngine Concepts: http://cfengine.com/manuals/cf3-conceptguide.html
- Installation: http://cfengine.com/manuals/cf3-installation.html

For a complete overview:

- Tutorial: http://cfengine.com/manuals/cf3-tutorial.html
- Reference manual: http://cfengine.com/manuals/cf3-reference.html

Links to external resources:

- Getting Started with CFEngine 3 by Vertical Sysadmin: http://www.verticalsysadmin.com/cfengine/Getting_Started_with_CFEngine_3.pdf
- CFEngine 3 Beginning Examples: http://www.verticalsysadmin.com/cfengine/beginning_examples/ This is, basically, a selection from /usr/local/share/doc/cfengine/ which has over 200 examples.
- "CFEngine 3 Tutorial" by Neil Watson: http://watson-wilson.ca/2011/05/cfengine-3-cookbook-begins.html