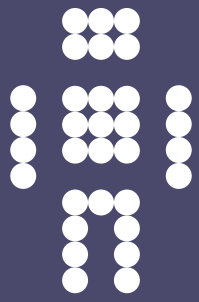
CFEngine



Upgrading from CFEngine 2 to 3

A CFEngine Handbook

CFEngine AS

Table of Contents

1 General remarks and expectations

This document concerns the translation of system configuration policies from the legacy CFEngine 2 language to the new CFEngine 3 promise language. CFEngine 3 is a new language that was designed with careful research to satisfy the needs of system configuration in a *convergent* fashion.

1.1 On the translation of policies

Translating one language directly into another rarely makes sense. Every language has its quirks and idioms that make some formulations more natural than others.

In migrating from CFEngine 2 to CFEngine 3, you will see that the novelty is more of a dialect than an unrelated language. The underlying parameterization of the promises is the same, and you will recognize the main features, even if they are inflected with an 'accent'.

This suggests that translation might be easy. However, we don't want you to trivialize this translation, as there are new mechanisms in CFEngine 3 that bring new benefits, and this means that simple and direct translation can be a poor choice that misses the opportunity for improvement. In this guide the principles for translation are simply as follows:

- We make as direct a translation as possible, sometimes offering alternatives that better illustrate CFEngine 3 paradigms.
- We use standardized templates from the *CFEngine Community Open Promise-Body Library* to simplify the translation. However, readers should understand that in every 'constraint' expression in CFEngine, of the form,

  ```
  LHS => RHS
  ```

  the left hand side is always a pre-defined CFEngine word, and the right hand side is always a user-defined term. In other words, if you don't like the choices we have made, you can make your own choices on the right hand side.

1.2 On 'best practices'

There are new features in CFEngine 3 that we strongly recommend you use. Perhaps the most useful practice is to make use of the standard library of template parts for promise bodies and bundles, called te *CFEngine Community Open Promise-Body Library*. This is available from the CFEngine website. By standardizing simple template names, you will be able to communicate more effectively with others, and facilitate knowledge efficiency in your organization.

Another feature is the ability to annotate or comment promises in a way that follows the promise through its lifecycle. This is part of the strategy of integrated knowledge management (see the Special Topics Guide on this subject).

```
files:

  "/etc/passwd" -> "stakeholder"

    comment => "Verify the integrity of the password file to change",
    content => detect_all_changes;
```

When a promise has a comment, this comment will be used to mark logs entries and error messages, providing context to these events for effective knowledge management.

You can give each promise a name, if you like, to make it easy to refer to or search for. We call this the promise 'handle'.

```
files:

  "/etc/passwd" -> "stakeholder"

    handle  => "passwd_change",
    comment => "Verify the integrity of the password file to change",
    content => detect_all_changes;
```

You can use this handle to document relationships between promises. For example, consider this hypothetical promise:

```
files:

  "/etc/group"

    handle  => "group_change",
    comment => "Add new users to the user groups",
 depends_on => { "passwd_change", "other_promise" },
  edit_line => fix_groups;
```

In CFEngine Nova and other commercial editions of the software, this documentation is automatically turned into browsable system documentation.

1.3 Completely new features

The CFEngine 3 Community Edition has many new features over CFEngine 2, and CFEngine Nova and the other commercial editions have many features over the Community Edition. You can write promises in the Community Edition for any of the commercial features without error – these will just not be functional in the Community Edition. This makes it easy to upgrade (or downgrade) freely.

New features in CFEngine 3 Community Edition include:

- All the components of CFEngine are now configurable and they read the same configuration file or files. In other words, you no longer have to maintain a separate input file for the server and the agent – self-contained configurations can be made for all parts of the system in one.
- Promises now have containers called *promise bundles*. There is no analogue of promise bundles in CFEngine 2, so you will need to sort through your promises and divide them into suitable bundles yourself, making sure to give each a sensible name.
- Powerful pattern matching and expression features that simplify the promises by allowing a consistent promises to be made from a whole set of objects according to a programmed pattern.
- Consistent use of Perl Compatible Regular Expressions for text matching.
- Array and list handling functions allow powerful associative patterns.
- Basic tools for Knowledge Management integrated with the configuration technology.
- Generic package management.
- Role based access control for remote activation of special promises.

New features in CFEngine Nova include:

- Automated Knowledge Management and analysis
- Database management promises.
- LDAP integration.
- Extended and integrated lightweight monitoring capabilities.
- Service and virtualization abstractions.
- Full native Windows support and promise types.

2 Conversion Strategy

2.1 Converting by module

To optimize the translation, you should think about the modularity of your code. First, look at how your CFEngine 2 configuration is modularized and consider how you want your final CFEngine 3 configuration to be modularized. CFEngine 3 has 'promise bundles' as modular entities (like subroutines or methods in other languages). The typical procedure is to take each file and convert it into a separate bundle. This makes each module into a separate entity in the integrated knowledge management.

There are potentially many ways to cut the cake, however. You can organize your configuration, by operating system, by service, by geography, etc. We recommend that you make separate bundles for each 'issue', 'service' or slice of the system that you are managing. Bundles should 'bundle together' related promises.

2.2 Assembling a compilable file set

The next step is to move as quickly as possible to a compilable CFEngine 3 configuration. You will be tweaking and perfecting this basic file set for ever more, but the sooner you have a syntactically valid file, the sooner you will benefit from the CFEngine tools.

1. Start by copying the Community Open Promise Body Library from the www.CFEngine.org website into the '`CFEngine3`' directory. We will base the converted configuration on these industry standard templates.

2. Now create the new master configuration file '`CFEngine3/promises.cf`'. This will replace the '`CFEngine2/cfagent.conf`' file.

   Suppose you start with a top level `cfagent.conf` that is organized as in the example below,

```
control:

  # ....

import:

  any::

    cfagent.global.conf

  freebsd::

    cfagent.freebsd.conf

  123.456.789::

    cfagent.usa.conf

  MailServers::

     cfagent.email.conf
```

This file is typically converted to something of the following form.

```
body agent control
{
# this is where control settings will go
}

body executor control
{
# this is where control settings will go
}

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

body common control
{
# Keep the bundlesequence simple

bundlesequence => { "g", "main" };

# The equivalent of imports
```

```
inputs => {
          "cfagent.freebsd.cf",
          "cfagent.usa.conf",
          "cfagent.email.conf",
          # ...
          "cfengine_stdlib.cf"
          };
}

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

bundle common g
{
vars:

  "localroot" string => "/a/b/c";
  "cfsrvhost" string => "198.129.252.125";
  "masterBuild" string => "/usr/local/CFEngine_export/RELEASE/build";

classes:

 # ...

}

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

bundle agent main
{
methods:

  any::
   "any" usebundle => global_stuff;

  freebsd::
   "any" usebundle => freebsd_stuff;

  MailServers::
   "any" usebundle => mail_stuff;


}
```

3. Compare these two control files above. Notice that there is no `actionsequence` in the CFEngine 3 configuration. CFEngine 3 can determine an order more automatically using

a best-effort heuristic algorithm.[1] All we need to do is include the different bundles in the basic order that we want and CFEngine will do the rest.

2.3 Validating the conversion

Validating the conversion is potentially an arduous process and the work is not over yet. Because CFEngine 3 uses body templates to simplify the appearence of promises, and promote the reusability of code, the conversion requires us to create these.

To complete the conversion, you will need to:

1. Check that the intention of the converted promise matches the orginal.
2. Check that variable references are ok - global ones can be referenced with def.varname (if they are defined in the bundle common def)
3. Insert comments
4. Simplifying repeated patterns using lists.
5. Run through cf-promises

2.4 Optimizing the configuration

You could optimize by not importing files that you don't need on all systems. This reduces memory and processing time. To do this, you can adapt the '`inputs`' and '`bundlesequence`' of the converted file appropriately.

[1] Some scheduling tools talk about 'sorting of dependencies' to determine order, but this is not possible in a dynamic environment, since you don't know which dependencies will be in play until after the sort.

3 Translation Codebook

3.1 upgrading from CFEngine 2 'acl'

File Access Control Lists have been completely re-implemented in CFEngine 3. They are only available now in the commercial version of CFEngine, but with the benefit a unifying, platform-independent (least common denominator) model (in addition to system specific models) for Posix, Solaris, Linux, Windows and other ACL models.

```
files:

   /some/path

       acl=myacl
    action=fixall

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

acl:

  { myacl

  fstype:posix
  method:overwrite
  mask:*:rwx
  user:*:rwx
  group:*:r-x
  other:*:r
  user:www:=rwx
  user:mark:=rwx
  default_mask:=rwx
  default_user:=rwx
  default_group:=r
  default_other:=r
  }
```

In CFEngine Nova, this would become approximately:

```
bundle agent acls

{
files:

  "/some/path"

      acl => myacl;
}

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

body acl myacl

{
acl_method => "overwrite";
acl_type => "posix";
acl_directory_inherit => "specify";

aces => {
        "mask:rwx",
        "user:*:rwx",
        "group:*:r,-x",
        "all:r",
        "user:www:=rwx",
        "user:mark:=rwx"
        };

specify_inherit_aces =>
       {
       "mask:=rwx",
       "user:*:=rwx",
       "group:*:=r",
       "all:=r"
       };
}
```

3.2 upgrading from CFEngine 2 'admit'

The admit declarations belong to the server configuration.

```
admit:   # or grant:

   /export/nexus/local/gnu/bin/CFEngine *.example.com
   /export/waldo/local/gnu/bin/CFEngine *.example.com

   /export/nexus/local   *.example.com
   /export/nexus/ud      dax.example.com
   /export/nexus/u4      dax.example.com dump-truck.example.com
   /etc                  *.example.com
```

These are translated as access promises:

```
bundle server rules
{
access:

   "/export/nexus/local/gnu/bin/CFEngine" admit => { ".*.example.com" };
   "/export/waldo/local/gnu/bin/CFEngine" admit => { ".*.example.com" }

   "/export/nexus/local" admit => { ".*.example.com" };
   "/export/nexus/ud"    admit => { "dax.example.com"};
   "/export/nexus/u4"    admit => { "dax.example.com", "dump-truck.example.com" };
   "/etc"                admit => { ".*.example.com" };
}
```

3.3 upgrading from CFEngine 2 'alerts'

In CFEngine 2, the term for reporting was 'alert'. This seemed too reactionary.

```
alerts:

   myclass::

       "Reminder: say hello every hour"

            ifelapsed=60

   nfsd_in_high_dev2::

       "High NFS server access rate 2dev at $(host)"

            ShowState(incoming.nfs)
```

In CFEngine 3, you would write:

```
reports:

   myclass::

     "Reminder: say hello every hour"

         action => ifelapsed("60");

   nfsd_in_high_dev2::

     "High NFS server access rate 2dev at $(host)"

         showstate => { "incoming.nfs" };
```

CFEngine 3 extends the possiblities for messaging considerably. CFEngine Nova generates many standard reports automatically.

3.4 upgrading from CFEngine 2 'binservers'

The CFEngine Mount Model has been deprecated in version 3. The introduction of the automounter largely superceded the use of this model, and while it is still possible to use CFEngine as a static automounter, there is no longer any need for an explicit definition of its parts, as simple pattern matching combined with mount promises suffices to solve this problem, See Section 3.25 [upgrading from CFEngine 2 miscmounts], page 41.

```
control:

 site = ( mysite )

 MountPattern = ( /$(site)/$(host) )
 HomePattern  = ( home? )

      actionsequence =
         (
         mountall
         mountinfo
         addmounts
         mountall
         )

mountables:

  any::

    serv1:/mysite/serv1/home1
    serv1:/mysite/serv1/home2
    serv1:/mysite/serv1/local
    serv3:/mysite/serv3/local1
    serv3:/mysite/serv3/local2
    serv4:/mysite/serv4/homeA
    serv4:/mysite/serv4/homeB

binservers:

    group1::

        serv1 serv2

    group2::

        serv3
```

In CFEngine 3, you might write this:

```
storage:

 group1::

   "/mysite/serv1/local" mount => nfs("serv1","/mysite/serv1/local");

 group2::

   "/mysite/serv3/local1" mount => nfs("serv3","/mysite/serv3/local1");
   "/mysite/serv3/local2" mount => nfs("serv3","/mysite/serv3/local2");

   # Or use lists to iterate this
```

3.5 upgrading from CFEngine 2 'broadcast'

This is deprecated in CFEngine 3.

3.6 upgrading from CFEngine 2 'control'

In CFEngine 2, the `control` part has two muddled functions:

- Setting parameters that control the internal behaviour of CFEngine.

  These are details that adjust the behaviour of promises that are hard-coded into CFEngine. Thus, they belong formally in `body` declarations according to the CFEngine 3 promise model.
- Defining user variables (macros).

  These are actual user-defined promises (the promise that a certain name will represent a certain value). They are thus represented as `vars` promises in CFEngine 3.

Note that there is no `actionsequence` in CFEngine 3. It is no longer needed and can be ignored.

The following CFEngine 2 code

```
control:

   Access     = ( root )        # Only root should run this

   site       = ( iu )
   domain     = ( iu.hio.no )
   sysadm     = ( CFEngine@example.com )
   smtpserver = ( smtp@example.com )

   # Welcome to Norway...!

   timezone  = ( MET CET )

   #
   # Where backup files (for copy/tidy) are kept
   #

   Repository = ( /var/spool/CFEngine )

   SplayTime = ( 4 )

   OutputPrefix = ( "cf:$(host)" )

   IfElapsed   = ( 15 )
   ExpireAfter = ( 240 )


   SensibleSize  = ( 1000 )
   SensibleCount = ( 2 )
   EditfileSize  = ( 40000 )

   cfbin       = ( /var/cfengine/bin )
   gnu         = ( "/local/gnu" )
```

```
   ftp         = ( /local/iu/ftp )
```

translates in CFEngine 3 into several pieces:

```
# Hard-coded promise parameters, common to all parts

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

# Hard-coded promise parameters for cf-agent

body agent control
{
default_repository => "/var/spool/CFEngine";
ifelapsed => "15";
expireafter => "240";
sensiblesize  => "1000";
sensiblecount => "2";
editfilesize  => "40000";
}

# Hard-coded promise parameters for cf-execd

body executor control
{
splaytime => "4";
mailto => "cfengine@example.com";
smtpserver => "smtp.example.com";
}

# User defined promises common to all parts

bundle common global_promises
{
vars:

  "cfbin"       string => "/var/cfengine/bin";
  "gnu"         string => "/local/gnu";
  "ftp"         string => "/local/iu/ftp";

}
```

Note that `vars` promises that are declared '`common`' are seen by all bundles and all agents. It is also possible to have variables in '`agent`' or '`server`' bundles that are seen only by those parts of CFEngine.

Control information from the '`cfservd.conf`' file goes naturally into a control body:

```
control:

  cfrunCommand  = ( "/var/cfengine/bin/cfagent" )
  AllowConnectionsFrom = ( 127.0.0.1 ::1 )
  AllowMultipleConnectionsFrom = ( 127.0.0.1 ::1 )
  TrustKeysFrom  = ( 127.0.0.1 ::1 )
  AllowUsers = ( root  mark )
```

becomes:

```
body server control

{
allowconnects         => { "127.0.0.1" , "::1" };
allowallconnects      => { "127.0.0.1" , "::1" };
trustkeysfrom         => { "127.0.0.1" , "::1" };
cfruncommand          =>
  "$(sys.workdir)/bin/cf-agent -f failsafe.cf && $(sys.workdir)/bin/cf-agent";
allowusers            => { "mark", "root" };
}
```

3.7 upgrading from CFEngine 2 'classes'

```
classes:  # same as groups

  Setup_SSH_OK   = ( '/usr/bin/test -f /etc/ssh2/ssh2_config' )
  science        = ( saga tor odin )
  notthis        = ( !this )
  ip_in_range_1  = ( IPRange(129.0.0.1-15) )
  ip_in_range_2  = ( IPRange(129.0.0.1/24) )
  compute_nodes  = ( HostRange(cpu-,1-32) )
  science        = ( +science-allhosts )
  physics_theory = ( +@physics-theory-sun4 dirac feynman schwinger )
  group1         = ( +mynetgroup -specialhost -otherhost )
  group2         = ( +bignetgroup -smallnetgroup )
  SpecialTimes   = ( Hr00 Monday Day1 )
```

These promises translate into CFEngine 3 as:

```
classes:

  "Setup_SSH_OK"   expression => fileexists("/etc/ssh2/ssh2_config");
  "science"               or => { "saga", "tor", "odin" };
  "notthis"        expression => "!this";
  "ip_in_range"    expression => iprange("129.0.0.1-15");
  "ip_in_range"    expression => iprange("129.0.0.1/24");
  "compute_nodes"  expression => hostrange("cpu-","1-32");
  "science"        expression => hostinnetgroup("science-allhosts");

  "physics_theory"         or => {
                                 hostinnetgroup("physics-theory-sun4",
                                 "dirac",
                                 "feynman",
                                 "schwinger"
                                 };

  "group1"                and => {
                                 hostinnetgroup("mynetgroup"),
                                 "!specialhost",
                                 "!otherhost
                                 };

  "helper"         expression => hostinnetgroup(smallnetgroup");
  "group2"                and => { hostinnetgroup("bignetgroup"), "!helper" };
  "SpecialTimes"           or => { "Hr00", "Monday", "Day1" };
```

3.8 upgrading from CFEngine 2 'copy'

Copying of files from one location to another had the following form in CFEngine 2:

```
copy:

    /masterfiles/hosts.deny dest=/etc/hosts.deny mode=644 server=nexus

   !(dax|cube|sigmund)::

    /masterfiles/hosts.allow dest=/etc/hosts.allow mode=644 server=nexus

  128_39_89.!securehosts::

     /masterfiles/ssh_banner_message_89 dest=/etc/ssh2/ssh_banner_message
                                        mode=644 owner=root group=root
                                        encrypt=true
```

In CFEngine 3, beware that the order of the source and destination have been reversed to follow the general principle in CFEngine 3 that the affected object (in this case the destination) is always the first object in the promise (the promiser).

```
files:

  "/etc/hosts.deny"
     copy_from => remote_cp("/masterfiles/hosts.deny","nexus"),
         perms => m("644");

  #

  !(dax|cube|sigmund)::

  "/etc/hosts.allow"
     copy_from => remote_cp("/masterfiles/hosts.allow","nexus"),
         perms => m("644");

  #

  128_39_89.!securehosts::

   "/etc/ssh2/ssh_banner_message"
     copy_from => secure_cp("/masterfiles/ssh_banner_message_89")
         perms => mog("644","root","root");
```

3.9 upgrading from CFEngine 2 'defaultroute'

This function is deprecated in CFEngine 3. Today it can normally be implemented by editing a file.

3.10 upgrading from CFEngine 2 'deny'

```
deny:

  $(public)/special *.moneyworld.com
```

becomes:

```
access:

  "$(public)/special"

      deny => { "*.moneyworld.com" };
```

3.11 upgrading from CFEngine 2 'disks'

```
disks:

  /usr

      freespace=10%
```

becomes

```
storage:

  "/usr"

      volume => min_free_space("10%");
```

3.12 upgrading from CFEngine 2 'directories'

```
directories:

   /usr/local/bin

        mode=755
       owner=root
       group=wheel
```

becomes

```
files:

  "/usr/local/bin/."

     create => "true",
     perms => mog("755","root","wheel");
```

3.13 upgrading from CFEngine 2 'disable'

Disabling files has many meanings in CFEngine 2. It covers log rotation as well as file disablement.

```
disable:

   /usr/bin/rsh
   /var/log/xferlog rotate=3
   /local/etc/fingerdir/userdata rotate=empty
```

```
files:

   "/usr/bin/rsh" rename => disable;

   "/var/log/xferlog"
        rename => rotate("3");

   "/local/etc/fingerdir/userdata"
        rename => rotate("0");
```

3.14 upgrading from CFEngine 2 'editfiles'

File editing is a complex subject. A few examples are provided.

```
editfiles:

  !rom21X::

     { /etc/ssh2/sshd2_config

     ReplaceAll "PrintMotd.*yes" With "PrintMotd no"
     ReplaceAll ".*Ssh1Compatibility.*yes.*" With "Ssh1Compatibility no"
     AppendIfNoSuchLine "Ssh1Compatibility no"
     HashCommentLinesMatching ".*Sshd1Path.*"
     DeleteLinesMatching ".*PasswordAuthentication.*"
     DeleteLinesMatching ".*PubkeyAuthentication.*"
     DeleteLinesMatching ".*AllowCshrcSourcingWithSubsystems.*"
     }
```

```
files:

  !rom21X::

   "/etc/ssh2/sshd2_config"

      edit_line => ssh_config;


# ..

bundle edit_line ssh_config
{
replace_patterns:

  "PrintMotd.*yes"
      replace_with => all("PrintMotd no");

  ".*Ssh1Compatibility.*yes.*"
      replace_with => value("Ssh1Compatibility no");

  ".*Sshd1Path.*"
      replace_with => comment("#");

delete_lines:

  ".*PasswordAuthentication.*";
  ".*PubkeyAuthentication.*";
  ".*AllowCshrcSourcingWithSubsystems.*";

insert_lines:

   "Ssh1Compatibility no";
}
```

```
editfiles:

   { /etc/shells

   AppendIfNoSuchLine "/bin/tcsh"
   AppendIfNoSuchLine "/bin/bash"
   AppendIfNoSuchLine "/local/gnu/bin/bash"
   }
```

```
vars:

  "lines" slist => { "/bin/tcsh", "/bin/bash", "/local/gnu/bin/bash" };

files:

   "/etc/shells"

      edit_line => append_if_no_lines(@(lines));
```

or an alternative solution

```
files:

   "/etc/shells"

      edit_line => shells;

# ..

bundle edit_line shells
{
insert_lines:

   "/bin/tcsh";
   "/bin/bash";
   "/local/gnu/bin/bash";
}
```

3.15 upgrading from CFEngine 2 'files'

The `files` action in CFEngine 2 was mostly about permissions. In CFEngine 3, all file related operations are collected under this banner.

```
files:

   PrimeServers::

    /local/dns/pz

           owner=dns
           mode=644
           action=fixall
           recurse=1
           exclude=Fixserial

    /local/dns/pz/Fixserial
           m=755
           action=fixplain

   NameServers::

    /local/logs/admin

           o=dns
           m=644
           act=fixplain

    /local/logs/security

           o=dns
           m=644
           act=fixplain

    /local/logs/updates

           o=dns
           m=644
           act=fixplain

    /local/logs/xfer                   o=dns m=644 act=fixplain

      #
      # Make sure anonymous ftp areas have the correct
      # protection, or logins won't be able to read files
      #

    $(ftp)/pub    mode=644  o=root g=other act=fixall
    $(ftp)/pub    mode=644 act=fixall  r=inf

    $(ftp)/etc        mode=111 o=root g=other      act=fixdirs
    $(ftp)/usr/bin/ls mode=111 o=root g=other      act=fixall
    $(ftp)/dev        mode=555 o=root g=other      act=fixall
    $(ftp)/usr        mode=555 o=root g=other      act=fixdirs
```

may be translated into:

```
vars:

  "ns_files" slist => {
                       "/local/logs/admin",
                       "/local/logs/security",
                       "/local/logs/updates",
                       "/local/logs/xfer"
                       };
files:

   PrimeServers::

    "/local/dns/pz"

            perms => mo("644","dns")
     depth_search => recurse("1"),
      file_select => exclude("FixSerial");

    "/local/dns/pz/FixSerial"

            perms => m("755"),
      file_select => plain;

   NameServers::

    "$(ns_files)"

            perms => mo("644","dns"),
      file_select => plain;

    #
    # Make sure anonymous ftp areas have the correct
    # protection, or logins won't be able to read files
    #

    "$(ftp)/pub"
             perms => mog("644","root","other");

    "$(ftp)/pub"
             perms => m("644"),
      depth_search => recurse("inf");

    "$(ftp)/etc"        perms => mog("111","root","other");
    "$(ftp)/usr/bin/ls" perms => mog("111","root","other");
    "$(ftp)/dev"        perms => mog("555","root","other");
    "$(ftp)/usr"        perms => mog("555","root","other");
```

3.16 upgrading from CFEngine 2 'filters'

Filters have been redefined as 'select' body templates. Filters exist for processes and files in CFEngine 2. These translate into keywords `process_select` and `file_select`.

```
filters:

   { testfilteralias

   Owner:     "mark"
   Group:     "cfengine"
   Type:      "dir|link"

   Result:    "Type|(Owner.Group)"  # Both owner AND group required correct
   }
```

becomes

```
body file_select testfilteralias

{
search_owners => { "mark" };
search_groups => { "cfengine" };
file_types    => { "dir","symlink" };

file_result => "file_types|(owners.groups)";
}
```

3.17 upgrading from CFEngine 2 'groups'

Groups are a synonym for classes, see See Section 3.7 [upgrading from CFEngine 2 classes], page 19.

3.18 upgrading from CFEngine 2 'homeservers'

The CFEngine Mount Model has been deprecated in version 3. The introduction of the automounter largely superceded the use of this model, and while it is still possible to use CFEngine as a static automounter, there is no longer any need for an explicit definition of its parts, as simple pattern matching combined with mount promises suffices to solve this problem, See Section 3.25 [upgrading from CFEngine 2 miscmounts], page 41.

```
control:

 site = ( mysite )

 MountPattern = ( /$(site)/$(host) )
 HomePattern  = ( home? )

      actionsequence =
         (
         mountall
         mountinfo
         addmounts
         mountall
         )

mountables:

  any::

    serv1:/mysite/serv1/home1
    serv1:/mysite/serv1/home2
    serv1:/mysite/serv1/local
    serv3:/mysite/serv3/local1
    serv3:/mysite/serv3/local2
    serv4:/mysite/serv4/homeA
    serv4:/mysite/serv4/homeB

homeservers:

    group1::

        serv1 serv2

    group2::

        serv4
```

In CFEngine 3, you might write this:

```
storage:

 group1::

   "/mysite/serv1/home1" mount => nfs("serv1","/mysite/serv1/home1");
   "/mysite/serv1/home2" mount => nfs("serv1","/mysite/serv1/home2");

 group2::

   "/mysite/serv4/homeA" mount => nfs("serv4","/mysite/serv3/homeA");
   "/mysite/serv4/homeB" mount => nfs("serv4","/mysite/serv3/homeB");
```

3.19 upgrading from CFEngine 2 'ignore'

Ignore is used in CFEngine 2 to skip directories or filenames during searches. CFEngine 3 does not have a global list for this, but uses local lists analogous to the 'ignore=' attributes.

To make a global list in CFEngine 3, you can simply define a list of names and attach it to any promise in the program. Instead of CFEngine 2:

```
ignore:

   one
   two
   three
```

we use:

```
bundle common defs
{
vars:

  "ignore_list" slist => {  "one", "two", "three" };
}

# ...

bundle agent filestuff
{
files:

   "/mypath"

     depth_search => recurse_ignore("inf",@(defs.ignore_list));
}
```

3.20 upgrading from CFEngine 2 'import'

```
import:

      one.cf
      two.cf
      three.cf
```

becomes

```
bundle common control
{
inputs => { "one.cf", "two.cf", "three.cf" };
}
```

In CFEngine 3, file imports are no longer order sensitive in the manner of CFEngine 2.

3.21 upgrading from CFEngine 2 'interfaces'

This promise type has been temporarily placed on hold, pending future developments. Interface management has become much simpler since the early days of CFEngine, but this will eventually include routing promises for network management.

3.22 upgrading from CFEngine 2 'links'

Linking files in CFEngine 2:

```
links:

    nexus::

      /etc/rsyncd.conf -> /local/etc/rsyncd.conf
```

In CFEngine 3 this becomes

```
files:

  nexus::

    "/etc/rsyncd.conf"

       link_from => ln_s("/local/etc/rsyncd.conf");
```

Linking directories of multiple children:

```
links:

   /usr/local/bin +>  /usr/local/lib/perl/bin
   /opt           +>! /local
```

In CFEngine 3 this becomes

```
files:

   "/usr/local/lib/perl/bin" => linkchildren("/usr/local/bin");
   "/local"                  => linkchildren("/opt");
```

Or alternatively, use recursive copy with `linkcopy_patterns => { ".*" }`

3.23 upgrading from CFEngine 2 'mailserver'

This section has been deprecated in CFEngine 3. It can be handled by `mount` promises.

3.24 upgrading from CFEngine 2 'methods'

In CFEngine 2, methods were experimental. Methods are ways of making subroutines of CFEngine code. They were executed as separate programs following a special protocol, and could be activated remotely.

There is no direct mapping between methods in CFEngine 2 and CFEngine 3. In CFEngine 3, methods are simply bundles of promises that are executed as a group. These bundles can be parameterized and re-used. They are what methods should have been in CFEngine 2. Remote methods, are not implemented in CFEngine 3. Instead CFEngine Nova provides the means for agents to share data remotely by 'voluntary cooperation'.

```
# cfagent.conf

control:

actionsequence = ( methods )

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

methods:

   SimpleMethod(null)

     action=cf.simple
     returnvars=null
     returnclasses=null
     server=localhost
```

and

```
# cf.simple

control:

 MethodName   = ( SimpleMethod )
 MethodParameters = ( null )
 actionsequence   = ( timezone )

classes:

  dummy = ( any )

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

alerts:

  dummy::

   "This simple method does nothing"

   ReturnVariables(void)
   ReturnClasses(void)
```

This can be achieved more simply in CFEngine 3 as:

```
bundle agent parent
{
methods:

  "some_id" usebundle => SimpleMethod;

#...
}

bundle agent SimpleMethod
{
classes:

  "dummy" expression => "any";

reports:

  dummy::

      "This simple method does nothing";
}
```

3.25 upgrading from CFEngine 2 'miscmounts'

```
miscmounts:

 host:/foo /mnt/foo

 myserver:/$(site)/libraryserver/data1
               /mnt/data1 ro

         # consistent syntax

 myserver:/$(site)/libraryserver/data2
               /mnt/data2 mode=ro
```

```
storage:

 "/foot" mount => nfs("host","/foo");

 "/$(site)/libraryserver/data1"

        mount => nfs_p("myserver","/$(site)/libraryserver/data1","ro");

 "/$(site)/libraryserver/data2"

        mount => nfs_p("myserver","/$(site)/libraryserver/data2","ro");
```

3.26 upgrading from CFEngine 2 ‘`mountables`’

This list has been deprecated in CFEngine 3, see See Section 3.25 [upgrading from CFEngine 2 miscmounts], page 41.

3.27 upgrading from CFEngine 2 'processes'

In CFEngine 2 process promises were muddled with commands that were used to restart processes that were not running. The led to inconsistency in the handling of commands. CFEngine 3 separates commands to restart processes so that the full range of promise attributes can be applied during process start control.

```
processes:

      "inetd"

         signal=hup

      "bootp"

          signal=kill
         exclude=rpc.bootparamd

      "cfservd"

          restart "/usr/local/sbin/cfservd"
          useshell=false

          #  matches=>6   warn number of matches is greater than or equal to 6
          #  matches=1    warn if not exactly 1 matching process
          #  matches=<2   warn if there are less than or equal to 2 matching processes
```

Translates to:

```
processes:

   "inetd"
             signals => { "hup" };

   "bootp"
             signals => { "kill" },
      process_select => exclude_procs(".*rpc.bootparamd.*");


   "cf-serverd"
            restart_class => "start_cfserverd";

          #  process_count => check_range(cfserv,6,inf); warn number of matches is greater
          #  process_count => check_range(cfserv,1,1);   warn if not exactly 1 matching pro
          #  process_count => check_range(cfserv,0,2);   warn if there are less than or equ

commands:

   start_cfserverd::

    "/usr/local/sbin/cf-serverd";

reports:

   cfserv_out_of_range::

     "cf-serverd is out of control!!";
```

We can make use of lists to simplify the checking of multiple processes:

```
processes:

   Syslogdhup::

      "Syslogd" signal=hup

   any::

      "snmp"                                signal=kill
      "powerd"                              signal=kill
      "mibiisa"                             signal=kill
```

becomes:

```
vars:

   "kill_list" slist => { "snmp", "powerd", "mibiisa" };

processes:

   Syslogdhup::

      "Syslogd" signals => { "hup" };

   any::

      "$(kill_list)" signals => { "kill" };
```

Lists can also be used to simplify process starting. The following script

```
processes:

  "named"   restart "/local/sbin/named -u dns"
            useshell=false
            inform=true

  "cfservd"  restart "/var/cfengine/bin/cfservd"
  "cfenvd"   restart "/var/cfengine/bin/cfenvd"
  "cfexecd"  restart "/var/cfengine/bin/cfexecd"
```

would translate more efficiently into:

```
vars:

   "daemons" slist => { "cf-monitord", "cf-serverd", "cf-execd" };

processes:

  "named"      restart_class => "restart_named";

  "$(daemons)" restart_class => canonify("start_$(component)");

commands:

   "/bin/echo /var/cfengine/bin/$(component)"
       ifvarclass => canonify("start_$(component)");

  restart_named::

    "/local/sbin/named -u dns"
            action => inform;
```

3.28 upgrading from CFEngine 2 'packages'

Package handling in CFEngine 3 is far superior and more flexible than in CFEngine 2. There are many ways to code packages promises. Here is a simple way to code specific lists of versioned packages. In CFEngine 2 one might write:

```
packages:

  autoconf-2.13.000227_6  version=2.13.000227_6 cmp=ge  action=install
  automake-1.9.6_3        version=1.9.6_3       cmp=ge  action=install
  gmake-3.81_3            version=3.81_3        cmp=ge  action=install
  help2man-1.36.4_2       version=1.36.4_2      cmp=ge  action=install
  mysql-server-5.0.67     version=5.0.67        cmp=ge  elsedefine=InstallMySQL

  # ...
```

This could be translated efficiently using an associative array:

```
vars:

  "v[autoconf-2.13.000227_6]" string => "2.13.000227_6"
  "v[automake-1.9.6_3]"       string => "1.9.6_3"
  "v[gmake-3.81_3]"           string => "3.81_3"
  "v[help2man-1.36.4_2]"      string => "1.36.4_2"

  # ...

  "packages" slist => getindices("v");

packages:

  "$(packages)"

      package_policy => "add",
          package_method => freebsd,
          package_select => ">=",
          package_version => "$(v[$(package)])";
```

3.29 upgrading from CFEngine 2 'rename'

This is an alias, see See Section 3.13 [upgrading from CFEngine 2 disable], page 25.

3.30 upgrading from CFEngine 2 'required'

This is an alias, see See Section 3.11 [upgrading from CFEngine 2 disks], page 23.

3.31 upgrading from CFEngine 2 'resolve'

The special resolver configuration in CFEngine 2 has been deprecated in favour of using straightforward editing commands to manage the resolver file. The special variable $(sys.resolv) points to the system's current resolver configuration file. Thus the CFEngine 2 configuration:

```
resolve:

     "search iu.hio.no CFEngine.com"
     128.39.89.10
     158.36.85.10
     129.241.1.99
```

may be translated as:

```
vars:

   "r" slist => { "128.39.89.10", "158.36.85.10", "129.241.1.99" };

files:

   "$(sys.resolv)"

         edit_line => resolvconf("iu.hio.no CFEngine.com",@(mybundle.r));
    # edit_default => empty;
```

3.32 upgrading from CFEngine 2 'scli'

SCLI (SNMP Command Line Interface) promises are deprecated in CFEngine 3. There are no plans to integrate CFEngine directly with SNMP.

Users of CFEngine Nova can use the generic `measurement` promises to encapsulate SNMP monitoring into the CFEngine framework if necessary.

3.33 upgrading from CFEngine 2 'shellcommands'

Shellcommands scheduled the execution of scripts and programs external to the CFEngine framework in CFEngine 2. The following examples

```
shellcommands:

 nexus::

  "/usr/sbin/shareall"

         ifelapsed=240

 cube.nfs_update::

  "/etc/init.d/nfs-server restart > /dev/null 2>&1"
```

may be translated as:

```
commands:

 nexus::

  "/usr/sbin/shareall"

      action => ifelapsed("240");

 cube.nfs_update::
  "/etc/init.d/nfs-server restart > /dev/null 2>&1"

     contain => in_shell;
```

3.34 upgrading from CFEngine 2 'strategies'

Strategies in CFEngine 2 define probabilistic classes. This has become part of a `classes` promise is CFEngine 3.

```
strategies:

  { spread_load

  percent_10: "1"
  percent_30: "3"
  precent_60: "6"
  }
```

translates as:

```
classes:

 "percent" dist => { "10", "30", "60" };
```

3.35 upgrading from CFEngine 2 'tidy'

```
tidy:

  /tmp/       pattern=*       recurse=inf   age=1
  /var/tmp    pattern=*       recurse=inf   age=2
  /           pattern=core    r=1           a=0
  /etc        pattern=core    r=1           a=0
```

This may be translated into the following:

```
files:

  "/tmp"

     depth_search => recurse("1"),
      file_select => name_age(".*","1");

  "/var/tmp"

     depth_search => recurse("inf"),
      file_select => name_age(".*","2");

  "/"
     depth_search => recurse("1"),
      file_select => name_age("core","0");

  "/etc"
     depth_search => recurse("1"),
      file_select => name_age("core","0");
```

3.36 upgrading from CFEngine 2 'unmount'

```
unmount:

  /mnt
```

Translates to:

```
storage:

 "/mnt" mount  => unmount;
```