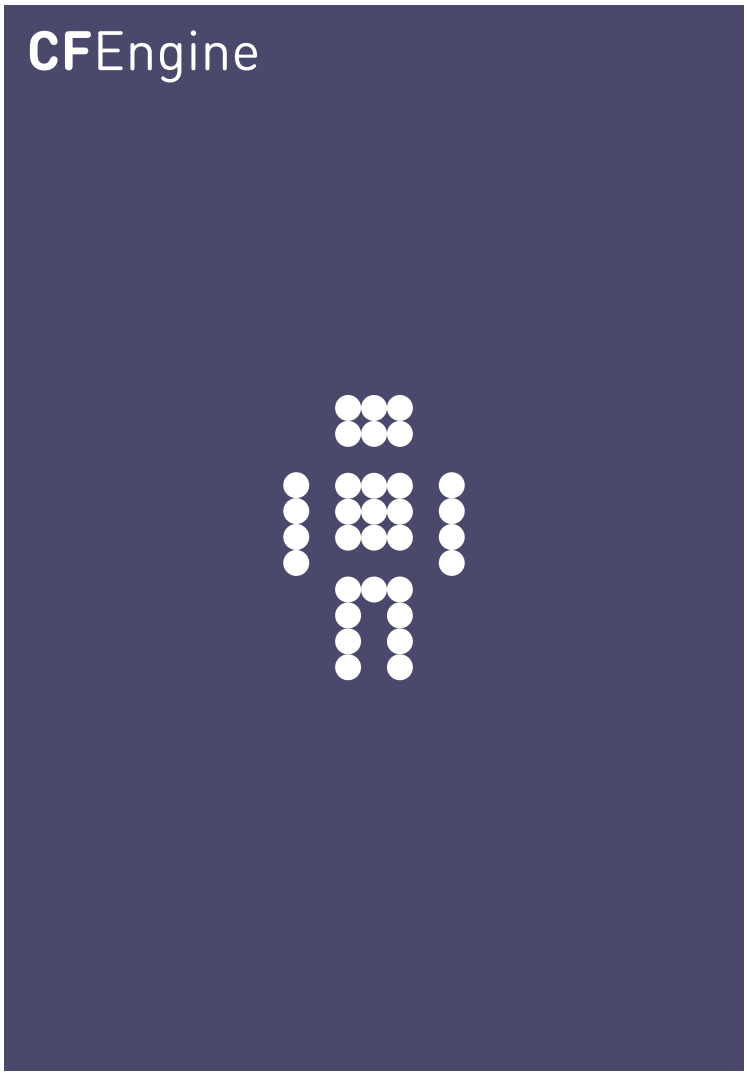


CFEngine Quick Start Guide

A CFEngine Handbook

Updated 26. February 2013

CFEngine AS

This short guide explains how to install the software and get it running.

Table of Contents

Welcome to CFEngine!

CFEngine is a systems management tool designed to help you configure and automate your IT infrastructure. It can expertly handle any number of tasks, ranging from the standardization of server builds to the deployment and management of software and services. In terms of real time administration, CFEngine has the ability to maintain system security from both a file system and process perspective and report on environmental health and status.

CFEngine Automation Overview

CFEngine is a distributed solution that is completely independent of host operating systems, network topology or system processes. You describe the ideal state of a given system by creating promises and the CFEngine agent ensures that the necessary steps are taken to achieve this state. Automation in CFEngine is executed through a series of components that run locally on all managed nodes.

CFEngine Components – What Are they and How Do They Work?

The power behind CFEngine lies in its components and the scripting language used to interact with them. Once you know the scripting language, you can clearly and specifically express your desired end state, allowing CFEngine to achieve your goals with precision.

There are a number of components in CFEngine, with each component performing a unique function. The following components form the basis of automation:

`cf-execd` The cf-execd process can be likened to the cron process in UNIX/Linux or the Windows Scheduler. It runs as a daemon and its job is to start the cf-agent process at a specified time interval. This user configurable interval defaults to 5 minutes.

`cf-agent` Is an agent that is called by the cf-execd process. The cf-agent component does the heavy work of system automation and maintenance.

`cf-serverd`
The cf-serverd process is used to distribute policy and/or data files.

In conjunction with the process/agents listed, CFEngine needs to know what you would like to automate, maintain or configure. For this purpose, CFEngine has two additional components: promises and policies. A promise is a statement, written in plain text using the CFEngine language, that describes the desired state of a system. A policy is a collection of one or more related promises, which is executed every time the cf-agent runs.

Next are the actual working files which are processed by CFEngine whenever the cf-agent process runs. Perhaps the most important file is the default policy file, which is called promises.cf. It is stored locally on every host, enabling execution with or without network dependencies. If you would like to update a policy, you simply make a change to the desired policy file on the hub/policy server, then update promises.cf accordingly. Each client Agent will automatically pull the new version in, then execute the new instructions on the next run.

Hubs, Servers and Clients

Once CFEngine is installed on a system, it can function as either a standalone client or as a part of a multi-node infrastructure. Clients are designed to pull policies from either a CFEngine policy server/hub, or, from themselves in the absence of a policy server. If a system is dedicated

as a CFEngine policy server or hub, it is still a client that will maintain it's own state regardless of it's function to the rest of CFEngine infrastructure. In addition, a client can assume various roles in any given infrastructure; be it a version control repository, distribution server or end host.

Checkpoint:

- CFEngine uses agents and language to perform automation and configuration tasks
- Instructions written in CFEngine syntax are known as promises
- One or more related promises can be written into a text file known as a policy
- The promise.cf file references policy files that each system will run in order to perform local automation, configuration and security tasks
- CFEngine maintains a desired system state on networked systems by utilizing client initiated pull technology; changes are never pushed or forced
- Networked CFEngine clients will check its policy server/hub in order pull new policy changes when they are updated
- The cf-agent process verifies the promises.cf file, then applies the policies to ensure that all promises are being kept
- The cf-execd daemon starts cf-agent process on a regular intervals
- The cf-serverd runs on a hub or server and allows client systems to retrieve policy changes and files.

CFEngine by Example

For the purpose of these exercises, it would be ideal to have 2 test systems available for CFEngine installation and configuration. You will name the first test system cfhub and the second system, which you will use in Step 3, will be named cfhost. Although installation and testing on a single host will sufficiently demonstrate the power of promises and policies, using 2 or more systems will allow you to experience what CFEngine brings to the table in terms of multi-system management.

Installation

The CFEngine package comes in many different flavors. For Linux systems, CFEngine is available as a pre-compiled binary for many distributions, including: Debian, Fedora, RHEL, SuSE and Ubuntu. These packages are available from `http://cfengine.com/inside/myspace`. To unpack the binaries (you must install them as root):

CFEngine 3 Enterprise Installation:

For CFEngine 3 Enterprise, you will designate one at least one server as the hub, which should be installed first.

CFEngine 3 Enterprise is provided in three packages (two hub and one client package). These are the three packages (inside the respective hub and client listings found under each platform in the software download page, example for 64 bit rpm packages):

- hub/cfengine-nova-3.0.xxx.x86_64.rpm
- hub/cfengine-nova-expansion-3.0.xxx.x86_64.rpm
- client/cfengine-nova-3.0.xxx.x86_64.rpm

The difference between the two cfengine-nova-3.0.xxx packages (for hub and client) is that the agent binaries for the hub are linked to the MongoDB library while the binary for clients are not. Installing a client package on the hub will result in a database connection error, the Mission Portal will not be available and reports will not be collected. Installing the hub package on clients will result in error reports stating that the agent failed to connect to MongoDB (because it is not existent in client packages; danger of filling up error logs), but otherwise functionality will be assured. Please take care to install the correct packages on the corresponding nodes. The expansion package is only installed on the policy hub. You should install and set up the hub first.

RedHat/SuSE

```
rpm -ihv cfengine-nova-<version_number>.rpm
rpm -ihv cfengine-nova-<version_number>.rpm
```

Debian/Ubuntu

```
dpkg --install  cfengine-nova-<version_number>.deb
dpkg --install  cfengine-nova-expansion.<version_number>.deb
```

To install CFEngine 3 Enterprise on non-Hub systems:

RedHat/SuSE

```
rpm -ihv cfengine-nova-<version_number>.rpm
```

Debian/Ubuntu

```
dpkg --install  cfengine-nova-<version_number>.deb
```

On the hub, a public key has now been created in

'`/var/cfengine/ppkeys/localhost.pub`'

as part of the package installation. You should send this public key to CFEngine Support as an attachment in the ticket system, to obtain a license file license.dat. Save the returned license file to '`/var/cfengine/masterfiles/license.dat`' on the hub before continuing.

For more details on software licensing see:

https://cfengine.com/software/Licensing.pdf

Community Installation:

Prior to installing CFEngine, you should first ensure that the following packages are installed:

OpenSSL Open source Secure Sockets Layer for encryption.
URL: http://www.openssl.org

Tokyo Cabinet (version 1.4.42 or later)
Lightweight flat-file database system.
URL: http://fallabs.com/tokyocabinet/

PCRE Perl Compatible Regular Expression library.
URL: http://www.pcre.org/

On Windows machines, you need to install the basic Cygwin DLL from http://www.cygwin.com in order to run CFEngine.

If they are not installed, then you can easily install them using the appropriate package manager:

Debian/Ubuntu:

```
apt-get install libtokyocabinet-dev libpcre3-dev
libssl-dev
```

CentOS/RedHat:

```
yum install tokyocabinet-devel pcre openssl
```

SuSE: `zypper install libtokyocabinet-devel pcre libopenssl`

Once the prerequisites are installed or verified, you may proceed to the CFEngine installation.

RedHat/SuSE:

```
rpm -ihv cfengine-community-3.x.y.rpm
```

Debian/Ubuntu:

```
dpkg --install cfengine-community-3.x.y.deb
```

For other UNIX/Linux systems, you may download and compile the source code, which is available from http://cfengine.com/source_code.

CFEngine File Structure

The CFEngine application is fully contained within the '`/var/cfengine`' directory tree. For CFEngine 3 Entprise files will also be placed in the operating system specific web document root (for instance in /var/www/ on Debian/Ubuntu). Although you will get to know this file system intimately as you progress, here is a quick breakdown of the directory structure and some of the files and functions associated with each subdirectory:

'`/var/cfengine/bin - Consists of the agents and daemons that run CFEngine, including:`'

- '`cf-agent`' - Agent: Executes the promises.cf file; ensures that all promises are being kept
- '`cf-execd`' - Daemon: Starts the cf-agent process at a specified time interval.
- '`cf-serverd`' - Daemon: Provides network services; used to distribute policy and data files
- '`cf-monitord`' - Daemon: Collects system statistics
- '`cf-promises`' - Agent: Verifies CFEngine's configuration syntax
- '`cf-runagent`' - Agent: Contacts a remote system to run cf-agent
- '`cf-report`' - Agent: Extracts and presents report data in HTML,XML or graph formats
- '`cf-know`' - Agent (CFEngine Enterprise only): Builds knowledge maps based on promises and data

'`/var/cfengine/masterfiles`'

Policy repository which grants access to local or bootstrapped CFEngine clients when they need to update their policies. Policies obtained from

'/var/cfengine/masterfiles' are then cached in '/var/cfengine/inputs' for local policy execution. The 'cf-agent' executable does not execute policies directly from this repository.

'/var/cfengine/inputs'
: Cached policy repository located on a CFEngine client. The 'cf-agent' executable executes policies from this repository.

'/var/cfengine/outputs'
: Directory where 'cf-agent' creates its output files.

'/var/cfengine/ppkeys'
: Directory used to store encrypted public/private keys for CFEngine client/server network communications.

'/var/cfengine/reports'
: Directory used to store reports generated by cf-report .

'/var/cfengine/lib'
: Directory to store shared objects and dependencies that are in the bundled packages.

Create and Manually Execute a Policy

If you have ever taken a course in programming, then you are probably familiar with the Hello World program. For CFEngine 3, the Hello World policy would look like this:

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

bundle agent test
{
reports:
  cfengine_3::
      "Hello world!";
}
```

This policy can be manually run by performing the following steps:

1. Log on to the new host; copy and paste the above Hello World script into a text file named '/var/cfengine/inputs/test.cf'
2. Verify the syntax by running cf-promises – if verification is successful, you will see no output:

   ```
   root@cfhub #  cf-promises -f /var/cfengine/inputs/test.cf
   ```

3. Execute the policy – if everything was successful, the string 'R: Hello World!' will be outputted immediately below the command:

   ```
   root@cfhub #   cf-agent -f /var/cfengine/inputs/test.cf
   R: Hello world!
   ```

Like many languages, CFEngine has a structure which allows declarations, variables and classes. The body structure is a place to list constraints and items that control the flow of a

policy. It can be seen as a template macro (similar to the main() function in c) that organizes promise attributes. The bundle structure can be viewed as a collection of promises under a single name, much like a subroutine in other languages.

Looking at the 'Hello World' promise syntax, you will notice that it starts with the following body structure:

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

In this example, the body common control is used to control promise behavior by calling the bundle agent test, which is a collection of promise attributes:

```
bundle agent test
{
reports:
  cfengine_3::
   "Hello world!";
}
```

This bundle structure contains a special type of promise known as reports: , which will automatically output 'Hello World' if all conditions of the promise are kept. Finally, there is the `cfengine_3::` class. This class is defined by CFEngine out of the box. When it's used in a promise, it checks to see that the CFEngine client is running version 3.

So, to summarize; the promise that you just wrote and executed checked that the version of CFEngine you are running is at version 3; and if it is at version 3, it sent the Hello World string to the command line.

Now that you have a better understanding of how the syntax works, we will go over the method we used to execute the policy. The policy that you wrote, test.cf was manually executed by the cf-agent binary. Prior to running the '`cf-agent`' command, you checked the promise syntax by running the '`cf-promises`' binary:

```
root@cfhub #  cf-promises -f /var/cfengine/inputs/test.cf
```

Normally, cf-promises is automatically executed by cf-agent as part of its run time operations, where, by default, it verifies the configuration syntax in the '`promises.cf`' file. By adding the -f flag and specifying a file name and path, you directed cf-promises to check and verify the specified file only: test.cf. In addition, we chose to run the policy manually, which is why we invoked cf-agent from the command line:

```
root@cfhub # cf-agent -f /var/cfengine/inputs/test.cf
```

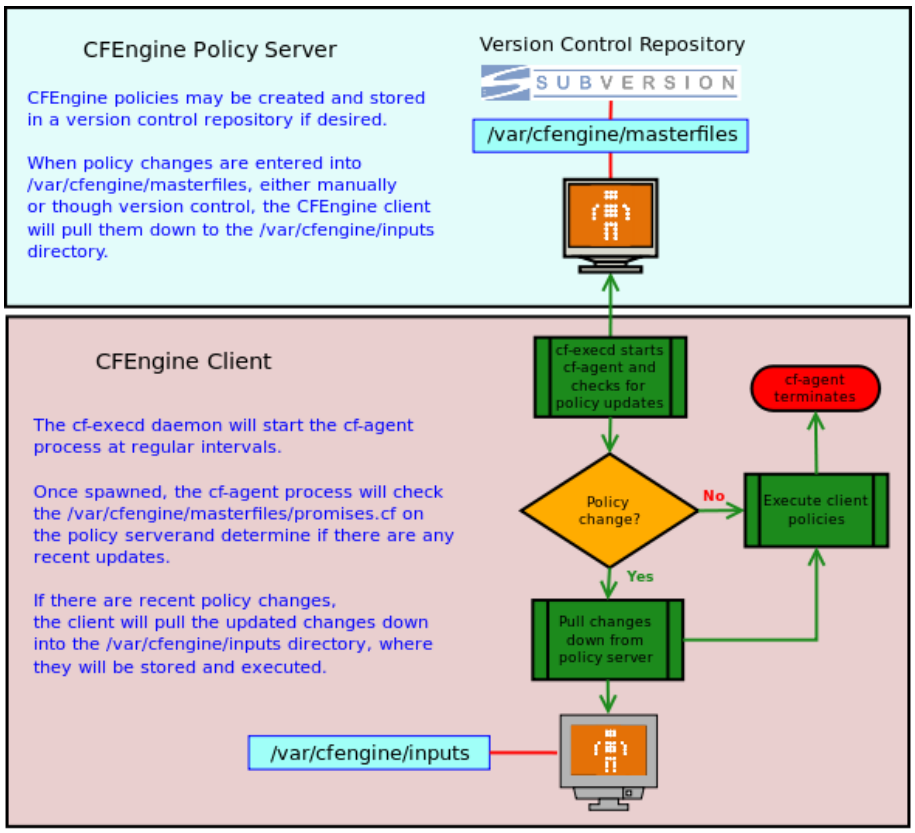
As a rule, CFEngine is fully automated and the cf-agent binary is started by the cf-execd daemon at a user specified interval (defaults to 5 minutes). Again, in this instance, the agent was manually started using the -f flag to execute the specified file.

If you would like to experiment with promise construction and manual execution, the following link will take you to a web based site designed for that purpose: Quickstart Promise Editor http://cfengine.com/policy_wizard/

CFEngine Client, Server Operations

In Step 2, we demonstrated how CFEngine operates as a standalone system. While there may be a few instances where running CFEngine on a standalone system is ideal; the real power

of CFEngine lies in its ability to perform multiple tasks in a multi-tiered network environment. To demonstrate the working relationship between a CFEngine policy server and client, we will set up a client named cfhost and configure it to pull and execute policies from the cfhub system, which will be set up as the policy server. Before we start, it might be helpful to get a 'look' at how CFEngine maintains a configured state in a networked environment. Note that the revision control system is optional. You only need to place your policy file into the '`/var/cfengine/masterfiles`' directory, then update the promises.cf file accordingly.



The objective of this exercise is to check that a file named '`cf_test_file`' exists on both the CFEngine policy server and client. If it does not exists, we are instructing CFEngine to create it. Additionally, we are directing CFEngine to ensure that the permissions are set to 644; that it is owned by root; and the group ownership is the '`sys`' group.

1. Create a second test system and name it cfhost.
2. Log on to cfhost; install and configure CFEngine according to the instructions given in Step 1.
3. Open a new terminal window and log on to cfhub.
4. Find the hostname or IP address of cfhub, here we will assume the address is 172.16.100.134
5. Designate cfhub as the policy server by bootstrapping it using its own IP address:

   ```
   root@cfhub# /var/cfengine/bin/cf-agent --bootstrap --policy-server 172.16.100.134
   ```

6. CFEngine will output diagnostic information upon bootstrap. If all is well you should see the following in the output:

```
-> Bootstrap to 172.16.100.134 completed successfully
```

7. Return to cfhost and bootstrap it to the IP address of the policy server cfhub: 172.16.100.134

   ```
   root@cfhost# /var/cfengine/bin/cf-agent --bootstrap --policy-server 172.16.100.134
   ```

   Ensure that the bootstrap was successful. If you have problems with this step, the most common issues are related to:

   Firewall Make sure port 5308 is open for both incoming and outgoing traffic.

   Policy access control list (acl)
   By default, the hub is available to clients that reside in the same class B network. If some clients are not part of that network, make sure the IPs are added to the acl in '/var/cfengine/masterfiles/def.cf', bundle common def:

   ```
    "acl" slist => {

                  # Assume /16 LAN clients to start with
                   "$(sys.policy_hub)/16",
                  # Add clients' IP addresses (or IP range) here (for
                  # example: an acl of 192.168.0.0/16 would allow all
                  # clients that have an IP address that starts with
                  # 192.168 to connect to the hub)
   ```

   cf-serverd is not running on the hub
   The CFEngine component `cf-serverd` takes care of all CFEngine communication. It can take a few minutes after you bootstrapped the hub to itself before this component is started, just wait a bit and retry.

8. On cfhub, go to the /var/cfengine/masterfiles directory, create a new file and name it cftest1.cf, then copy the following lines into it:

   ```
   bundle agent test
   {
   files:
     "/tmp/cf_test_file"
         comment => "Promise that a plain file exists with stated permissions",
           perms => mog("644", "root", "sys"),
          create => "true";
   }
   ```

9. Open the /var/cfengine/masterfiles/promises.cf file and locate the body common control section:

   ```
   body common control
   {
   bundlesequence => { "main" };
   inputs => {
             "cfengine_stdlib.cf",
             };
   version => "Community Promises.cf 1.0.0";
   }
   ```

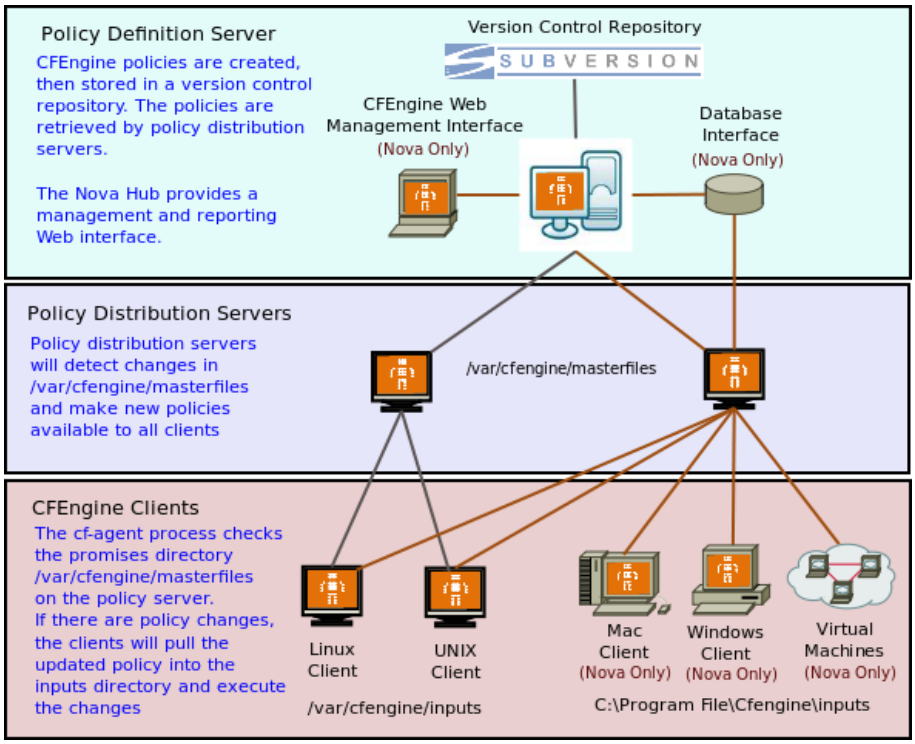
Modify this section by adding the new bundle name and input file:

```
body common control
{
bundlesequence => { "main","test" };
        inputs => {
                  "cfengine_stdlib.cf",
                  "cftest1.cf",
                  };
       version => "Community Promises.cf 1.0.0";
}
```

10. Save the /var/cfengine/masterfiles/promises.cf file.
11. Return to cfhost and manually start the agent by running: '`/var/cfengine/bin/cf-agent -Kv`'. The '`-K`' switch will bypass any locks and the '`-v`' switch will allow you to see the actual output.
12. After the agent runs successfully, check to see if the promise was kept by looking for the '`/tmp/cf_test_file`' file on both cfhub and cfhost.

Additionally, take a look at the '`/var/cfengine/inputs`' on cfhost; you will now see the cftest1.cf file, which was moved from cfhub. You will also notice that the original promises.cf file has been saved and the '`promises.cf`' file from cfhub is now in its place.

The exercise above was a simple example utilizing a very straightforward client/ server setup. CFEngine can scale to fit any existing environment as illustrated in the image below:



In this example, there are multiple policy distribution servers. Each policy server acts as a discrete distribution point for policies that are unique to a specific subset of hosts. Additionally,

a CFEngine 3 Enterprise Hub can act as a policy distribution point for Windows and Macs. The Enterprise Hub also provides a management and reporting Web interface for all of the systems that it manages.

We hope that this document has been a helpful first step in your quest to unleash the power of CFEngine in your environment! For more information, please visit the CFEngine Web Site, where you will find a wealth of documentation; including tips, tricks, tutorial's, manual's, references, How-To's and cookbooks.

Next steps

We recommend the following reading:

- CFEngine Concepts: http://cfengine.com/manuals/cf3-conceptguide.html
- Get started, first promises: http://cfengine.com/manuals/cf3-tutorial.html#First-promises

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 /var/cfengine/share/doc/examples/ which has over 200 examples.
- "CFEngine 3 Tutorial" by Neil Watson: http://watson-wilson.ca/2011/05/cfengine-3-cookbook-begins.html