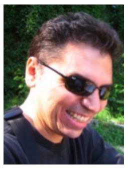
Configuring Your Linux System with the CFEngine Design Center

DIEGO ZAMBONI



Diego Zamboni is a computer scientist, consultant, author, programmer, sysadmin, and overall geek who works as a senior security advisor at CFEngine. He has more than 20 years of experience in system administration and security, and has worked in both the applied and theoretical sides of the computer science field. He holds a Ph.D. from Purdue University, and has worked as a sysadmin at a supercomputer center, as a researcher at the IBM Zurich Research Lab, and as a consultant at HP Enterprise Services. Zamboni is the author of the book "Learning CFEngine 3", published by O'Reilly Media. He lives in Queretaro, Mexico with his wife and two daughters. diego.zamboni@cfengine.com

CFEngine is an efficient, lightweight, and powerful configuration management tool for computer systems of all kinds. The most recent version, CFEngine 3.5.2, was released in August 2013. With CFEngine, you can express the desired state of your systems in two main ways: by writing policy in the CFEngine policy language directly, or by using the CFEngine Design Center [1], a repository of ready-to-use components called sketches, which allow you to perform entirely data-driven configuration. There are sketches for all sorts of tasks, from basic system configuration to complex cloud deployments. In this article, I will use simple examples to show you how to perform basic configuration tasks using the Design Center.

Getting Ready

First, you must install CFEngine and the Design Center on your system. Two versions of CFEngine are available: the open-source version (CFEngine Community) and the commercial version (CFEngine Enterprise). In my examples, I will use the Community version, which is available as packages for most Linux distributions and can also be downloaded and compiled from source code [2].

The easiest way to set up a test environment is to use Vagrant [3]. If you have Vagrant installed, fetch the sample Vagrantfile [4], put it in a directory, run *vagrant up*, and you will have a freshly installed Ubuntu 12.04 VM with both CFEngine and the Design Center ready to use. Then you can skip the rest of this section. If you prefer to do this on your own machine, or you don't want to use Vagrant, follow the instructions below.

I will use a fresh Ubuntu 12.04/64bit install, and follow the instructions from https://cfengine.com/cfengine-linux-distros to install CFEngine (command output edited for brevity):

```
# wget -q http://cfengine.com/pub/gpg.key
# apt-key add gpg.key
# rm gpg.key
# echo "deb http://cfengine.com/pub/apt $(lsb_release -cs) main" >  \
> /etc/apt/sources.list.d/cfengine-community.list
# apt-get -qq update
# apt-get -qq install cfengine-community
Selecting previously unselected package cfengine-community.
...
```

Now CFEngine is installed but not running. For this, we need to bootstrap CFEngine to a policy server. We will set up our machine as its own policy server, so we need to bootstrap to its own IP address:

```
# ifconfig eth0
eth0     Link encap:Ethernet  HWaddr 08:00:27:fe:aa:af
         inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
...
# /var/cfengine/bin/cf-agent --bootstrap 10.0.2.15
2013-08-19T22:25:53+0000   notice: Q: "...f-serverd"": 2013-08-
19T22:25:53+0000   notice: Server is starting...
2013-08-19T22:25:53+0000   notice: R: This host assumes the
role of policy server
2013-08-19T22:25:53+0000   notice: R: Updated local policy from
policy server
2013-08-19T22:25:53+0000   notice: R: Started the server
2013-08-19T22:25:53+0000   notice: R: Started the scheduler
2013-08-19T22:25:53+0000   notice: Bootstrap to '10.0.2.15'
completed successfully!
```

Now CFEngine is running, which you can verify by looking at the running processes:

```
# ps  ax | grep cf-
 1869 ?        Ss     0:00 /var/cfengine/bin/cf-execd
 1875 ?        Ss     0:00 /var/cfengine/bin/cf-serverd
 1889 ?        Ss     0:00 /var/cfengine/bin/cf-monitord
```

From now on, the CFEngine command *cf-agent* will run every five minutes to execute its policies. In this article, I will not go into more detail about how CFEngine works, but rather show you how you can use the CFEngine Design Center to configure your system without having to write CFEngine policies. The Design Center is hosted on GitHub [5], and its repository includes both the sketches and the tools used to manage them. We will clone the repository using *git*:

```
# apt-get -qq install git libterm-readline-gnu-perl
# cd /var/cfengine/
# git clone https://github.com/cfengine/design-center
Cloning into 'design-center'...
...
```

Using the CFEngine Design Center

Now we are ready to start using the Design Center. From the command line, the *cf-sketch* tool is the main way to manage Design Center sketches on your systems. CFEngine Enterprise includes a GUI for the Design Center, but for now we will stick to the command-line tools.

First, we need to run *cf-sketch*, which will put us in an interactive prompt:

```
# cd /var/cfengine/design-center/tools/cf-sketch/
# ./cf-sketch.pl
Welcome to cf-sketch version 3.5.0b1.
CFEngine AS, 2013.
Enter any command to cf-sketch, use 'help' for help, or 'quit' or
'^D' to quit.

cf-sketch> _
```

You can type *help* at this prompt to see all the commands available. In particular, you can type *search* to produce a listing of all the sketches available in the repository. For now, we will dive straight into the configuration of our system.

Let's look at some of the system configuration sketches available in the Design Center:

```
cf-sketch> search system

The following sketches match your query:

System::Logrotate Manage log rotation settings
System::Routes Manage system routes
System::Sudoers Sets defaults and user permissions in the
sudoers fileSystem::Syslog Configures syslog
System::access Manage access.conf values
System::config_resolver Configure DNS resolver
System::cron Manage crontab and /etc/cron.d contents
System::etc_hosts Manage /etc/hosts
System::motd Configure the Message of the Day
System::set_hostname Set system hostname. Domain name is also
set on Mac, Red Hat and and Gentoo derived distributions (but
not Debian).
System::sysctl Manage sysctl values
System::tzconfig Manage system timezone configuration
```

First we will configure the system timezone. For this, we will use the *System::tzconfig* sketch. We can use the *info* command to get detailed information about the sketch, including the parameters it uses:

```
cf-sketch> info -v System::tzconfig

The following sketches match your query:

Sketch System::tzconfig
Description: Manage system timezone configuration
Authors: Nick Anderson <nick@cmdln.org>, Ted Zlatanov <tzz@
lifelogs.com>
Version: 1.2
License: MIT
Tags: cfdc
Installed: No
Parameters:
  For bundle set
    timezone: string
    zoneinfo: string
Return values:
    Bundle set: [ timezone ]
```

The first step is to install it:

```
cf-sketch> install System::tzconfig

Sketch System::tzconfig installed under /var/cfengine/
masterfiles/sketches.
```

We can verify that the sketch has been installed using the *list* command. Note that a couple of library sketches were automatically installed as dependencies of *System::tzconfig*:

```
cf-sketch> list

The following sketches are installed:

CFEngine::dclib Design Center standard library
CFEngine::stdlib The portions of the CFEngine standard library
(also known as COPBL) that are compatible with 3.4.0 releases
System::tzconfig Manage system timezone configuration
```

Next, we need to define a parameter set for our sketch, which contains the values of the parameters needed by the sketch (enter your own timezone instead of the one shown here):

```
cf-sketch> define params System::tzconfig

Please enter a name for the new parameter set (default:
System::tzconfig-set-000): tzconfig1
Querying configuration for parameter set 'tzconfig1' for bundle
'set'.
Please enter parameter timezone.
  (enter STOP to cancel)
timezone : Mexico/General
Please enter parameter zoneinfo.
  (enter STOP to cancel)
zoneinfo : /usr/share/zoneinfo
Defining parameter set 'tzconfig1' with the entered data.
Parameter set tzconfig1 successfully defined.
```

Now we need to define an environment, which is short for "set of conditions under which a sketch will be executed with certain parameters". The conditions are expressed as CFEngine class expressions, so they can represent arbitrary conditions on the system, either automatically detected by CFEngine or set by your own CFEngine policies. For our example, we will activate our sketches in all Linux machines, so we will use the *linux* class, which is automatically set by CFEngine when it runs on a Linux host:

```
cf-sketch> define env

Please enter a name for the new environment: env_linux
I will now prompt you for the conditions for activation, test,
and verbose mode
that will be associated with environment 'env_linux'. Please
enter them as
CFEngine class expressions.
Please enter the activation condition: linux
Please enter the test condition: !any
Please enter the verbose condition: !any
Environment 'env_linux' successfully defined.
```

Now all we need to do is activate our sketch, telling it that we want to run it with the parameter set we defined, on all Linux machines:

```
cf-sketch> activate System::tzconfig tzconfig1 env_linux

Using generated activation ID 'System::tzconfig-1'.
Using existing parameter definition 'tzconfig1'.
Using existing environment 'env_linux'.
Activating sketch System::tzconfig with parameters tzconfig1.
```

Note that both parameter sets and environments have names, and all that the activate command does is "tie together" a sketch, a parameter set, and an environment.

After we have activated a sketch, we need to deploy and execute it, which can be done as a one-time operation using the run command (mostly for testing your parameters). Note the change in the system timezone before and after the sketch executes:

```
# date
Tue Aug 20 06:40:29 UTC 2013
#./cf-sketch.pl

cf-sketch> list activations

The following activations are defined:

Activation ID System::tzconfig-1
  Sketch: System::tzconfig
  Parameter sets: [ tzconfig1 ]
  Environment:  'env_linux'

cf-sketch> run

Runfile /var/cfengine/masterfiles/cf-sketch-runfile-standalone.
cf successfully generated.
Now executing the runfile with: /usr/local/sbin/cf-agent  -f /
var/cfengine/masterfiles/cf-sketch-runfile-standalone.cf

2013-08-20T06:40:47+0000   notice: R: System timezone updated
to Mexico/General
```

```
cf-sketch>
# date
Tue Aug 20 01:40:51 CDT 2013
```

Of course, you don't want to run the sketches manually, when the purpose of CFEngine is to keep your systems automatically configured. To automate the process, we must incorporate the execution of the sketches into the periodic execution of CFEngine by using the deploy command:

```
cf-sketch> deploy

Runfile /var/cfengine/masterfiles/cf-sketch-runfile.cf
successfully generated.
```

In the current release of CFEngine, we must make one change to the included CFEngine policy files in order for the sketches to be properly loaded. Open the */var/cfengine/masterfiles/promises.cf* file and find this section:

```
        # COPBL/Custom libraries.  Eventually this should use
wildcards.
             @(cfengine_stdlib.inputs),

         # Design Center
             # MARKER FOR CF-SKETCH INPUT INSERTION
             "cf-sketch-runfile.cf",
```

Because sketches load their own libraries, we must comment out the line that loads the CFEngine standard library and add a line that loads sketch-required files. The end result looks like this:

```
        # COPBL/Custom libraries.  Eventually this should use
wildcards.
         #   @(cfengine_stdlib.inputs),

         # Design Center
             # MARKER FOR CF-SKETCH INPUT INSERTION
             "cf-sketch-runfile.cf",
             @(cfsketch_g.inputs),
```

Now the sketch we activated will be executed every five minutes to check whether anything needs to be fixed. If you manually change the timezone of your system, you will notice that within five minutes it will change back to the one you configured in the sketch.

We will now configure two additional sketches for basic system configuration tasks, following the same install-parameters-activate sequence we already saw. First, we will use CFEngine to maintain our */etc/motd* file:

```
cf-sketch> install System::motd

Sketch System::motd installed under /var/cfengine/masterfiles/
sketches.

cf-sketch> define params System::motd

Please enter a name for the new parameter set (default:
System::motd-entry-000): motd1
Querying configuration for parameter set 'motd1' for bundle
'entry'.
Please enter parameter motd (Message of the Day (aka motd)).
  (enter STOP to cancel)
motd : This sytem is managed by CFEngine. Go away!
Please enter parameter motd_path (Location of the primary,
often only, MotD file).
  (enter STOP to cancel)
motd_path [/etc/motd]: /etc/motd
Please enter parameter prepend_command (Command output to
prepend to MotD).
  (enter STOP to cancel)
prepend_command [/bin/uname -snrvm]: /bin/uname -snrvm
Please enter parameter dynamic_path (Location of the dynamic
part of the MotD file).
  (enter STOP to cancel)
dynamic_path :
Please enter parameter symlink_path (Location of the symlink to
the motd file).
  (enter STOP to cancel)
symlink_path :
Defining parameter set 'motd1' with the entered data.
Parameter set motd1 successfully defined.

cf-sketch> activate System::motd motd1 env_linux

Using generated activation ID 'System::motd-1'.
Using existing parameter definition 'motd1'.
Using existing environment 'env_linux'.
Activating sketch System::motd with parameters motd1.
```

Note that we do not need to define additional environments for these sketches; they are being activated using the same *env_linux* environment we prepared previously.

We will also use CFEngine to maintain some security-related parameters in the system's sshd configuration:

```
cf-sketch> install Security::SSH

Sketch Security::SSH installed under /var/cfengine/masterfiles/
sketches.

cf-sketch> define params Security::SSH
```

```
Please enter a name for the new parameter set (default:
Security::SSH-sshd-000): ssh1
Querying configuration for parameter set 'ssh1' for bundle
'sshd'.
Please enter parameter params.
  (enter STOP to cancel)
Next key (Enter to finish): PermitRootLogin
params[PermitRootLogin]: no
Next key (Enter to finish): X11Forwarding
params[X11Forwarding]: no
Next key (Enter to finish):
Defining parameter set 'ssh1' with the entered data.
Parameter set ssh1 successfully defined.

cf-sketch> activate Security::SSH ssh1 env_linux

Using generated activation ID 'Security::SSH-1'.
Using existing parameter definition 'ssh1'.
Using existing environment 'env_linux'.
Activating sketch Security::SSH with parameters ssh1.
```

Before those sketches take any effect, they must be deployed:

```
cf-sketch> deploy

Runfile /var/cfengine/masterfiles/cf-sketch-runfile.cf
successfully generated.
```

Within a few minutes, you should see those changes reflected in your system:

```
# cat /etc/motd
This sytem is managed by CFEngine. Go away!
# egrep 'PermitRoot|X11For' /etc/ssh/sshd_config
PermitRootLogin no
X11Forwarding no
```

We are done! Your system automatically will be maintained according to the criteria you set. Try modifying any of these settings to see how CFEngine automatically brings them back into compliance. To get a better idea of all the things you can do with the Design Center, I encourage you to explore the available sketches.

Conclusion

In this article, I have only touched on the surface of what the Design Center can do. To learn more about the Design Center's capabilities and how to contribute new sketches, read the CFEngine documentation at http://cfengine.com/docs.

References

[1] CFEngine Design Center: http://cfengine.com/cfengine-design-center/

[2] Download CFEngine: https://cfengine.com/downloads.

[3] Vagrant: http://www.vagrantup.com/

[4] Vagrantfile: https://raw.github.com/cfengine/vagrant-cfengine-provisioner/master/sample/community_vagrant-1.2/Vagrantfile

[5] CFEngine Design Center on GitHub: https://github.com/cfengine/design-center