Pro Office 2007 Development with VSTO

ISBN-13 (pbk): 978-1-4302-1072-6

ISBN-13 (electronic): 978-1-4302-1071-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Tony Campbell
Technical Reviewer: John Mueller
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Sofia Marchant
Copy Editor: Nicole Abramowitz
Associate Production Director: Kari Brooks-Copony
Production Editor: Janet Vail
Compositor: Pat Christenson
Proofreader: Nancy Bell
Indexer: Broccoli Information Management
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

The source code for this book is available to readers at http://www.apress.com.

CHAPTER 4

Configuring Systems with cfengine

So far we've been discussing automation in a general way. At this point we'll move beyond single ad hoc measures to a more systematic and robust approach. While you certainly have the option of writing your own collection of automation scripts, we recommend you use a proven automation framework: cfengine.

Getting an Overview of cfengine

Cfengine is software you can use to automate changes on UNIX (and UNIX-like) systems. It is a high-level language that describes system state, not a general-purpose programming language such as Perl or a shell. It's primarily declarative, meaning that the SA writes out a technical description instead of a list of low-level steps to accomplish the goal. It is high-level enough that someone familiar with UNIX concepts and usage can read a cfengine configuration and understand what is being done without any prior cfengine knowledge.

The language drives what you should consider your personal software robot. This robot (called `cfagent`) does your repetitive work for you while you move on to other tasks.

In this chapter we'll use the current version of cfengine at the time of this writing: version 2.2.7.

Defining cfengine Concepts

Cfengine was designed to save time and reduce errors though automation. Its second, related goal is to enable computer systems to self-correct errors. It might take you some time to set up and configure cfengine, but you will be happier when everything has been said and done.

At first, performing a new task with cfengine might take longer than performing the same task manually. But when you upgrade the operating system and lose a change made under the old OS installation, you'll be glad you used cfengine because it will simply perform the change again. Or, when you realize a few other systems need the same change,

you can use cfengine to make this happen in seconds (by adding the new systems to the appropriate class).

If you made the change manually, on the other hand, it might take some time before you even notice that the change was lost. Once you notice, you'll have to make the change manually all over again—that is, of course, if you remember how you did it the last time. If ten new systems need a specific change, you might spend an hour changing each system yourself, whereas cfengine could have just done it for you.

Cfengine allows you to use one set of configuration files. Each host can transfer the configuration files from one or more cfengine servers before each run. As long as you make all the changes in that set of configuration files, all systems will receive the configuration updates automatically. You will no longer need to remember to make manual system changes. You will no longer need to use special scripts for special systems and/or scripts that have so many conditionals (based on hostname, operating system, etc.) that they've become unreadable and difficult to maintain. Cfengine comes with a rich set of automatically detected UNIX characteristics that the SA can use to perform tasks on only the desired systems.

Perhaps most important, this set of configuration files documents every change you make to every system. If you put a few comments in the files along with the commands, you will document not only what you have done but also *why* you did it.

CENTRALIZED CFENGINE CONFIGURATION FILES

Cfengine doesn't force centralized configuration files onto its users. In our examples, we choose to maintain a single configuration-file tree and distribute it to all hosts, and cfengine allows us to update configuration files any way we choose. Some sites choose to retrieve some or all files directly from a revision-control system such as Concurrent Versions System (CVS) or Subversion on all client systems. Some sites have configuration files copied from multiple remote servers to create a single configuration tree, in what would be considered a decentralized model.

For some tasks, cfengine abstracts the desired action from the technical specifics of the underlying operating system. For other tasks (namely editing files), cfengine provides an editing-specific command that allows you to specify the modifications exactly. Using these commands is similar to using `sed`. The cfengine text-editing commands have low-level abilities in addition to higher-level ones. We will cover the `editfiles` feature of cfengine later in this chapter.

Cfengine doesn't provide native support for certain tasks, but it lets you execute external scripts based on a system's class membership. When possible, you should use the internal commands that cfengine provides. If you don't, you can use custom shell and Perl scripts, but you should still get cfengine to execute them on your behalf.

Once you decide to use cfengine, you'll want to use it for as many tasks as possible. So you'll probably need to change your habits because you might be tempted to just "fix it real quick" instead of going through the proper cfengine process. The quotes around the word "quick" are carefully positioned. If you do a manual "quick fix" to your existing set of systems, a newly deployed system might be missing the change. When you work to redeploy the change using cfengine, you'll have to figure out how the change was made, test it, and deploy it once again. The simple act of figuring out how to make the change again is time-consuming. Using cfengine to deploy the change in the first place ensures you don't have to go through that process again for your existing systems and configuration, at least until you upgrade your OS or major applications.

You don't have to use cfengine to control all aspects of system configuration, so you can easily introduce it into an existing management framework without eliminating any existing methods of system configuration. You are free to use cfengine to control only the aspects of your systems that you're initially comfortable with. Over time, you can migrate the old configuration methods into cfengine. This situation isn't ideal for the long term because having two administration frameworks incurs increased complexity, but it will help you get comfortable with cfengine before committing all your site's administration to it.

Once you switch to cfengine, you will enjoy many benefits:

- *A standardized configuration for all hosts on your network that you can use to enforce homogeneity or to support diversity, each in an automated manner*: Cfengine configuration rules are each essentially a promise about the nature of the system. The `cfagent` program ensures that promises are kept.
- *The ability to change specific systems*: You can classify systems using a variety of built-in methods and classes (even ones added by the SA staff), and make changes only on the appropriate systems.
- *The ability to record system changes and perform them again if necessary*: One of cfengine's primary goals is to bring systems into conformance with policy and *keep* them that way.
- *Systems that might have intermittent uptime or network connectivity but will eventually make any necessary changes*: You won't need to keep track of what systems were up when you made a particular change across all your systems.

Evaluating Push vs. Pull

Yet another advantage to using cfengine is that it pulls from a server instead of pushing from the master system. This doesn't make a big difference when you have a local set of reliable servers that are usually up and running. But the pull method is much

more reliable if your systems are spread out over an unreliable network or if they aren't always running.

For example, if some systems can boot to either Linux or Windows, they can pull from the server whenever they are running Linux. If you were to use a push technique instead, the system might get neglected if it's running Windows at the time of the attempted push because the master push system expects a Linux host to be running at the remote IP address. When such systems boot into Linux again, cfengine will copy the latest configuration files to the system and ensure that the current promises are kept.

Another problem situation arises when you have UNIX running on one or more laptops that are not always connected to the network. A system like this might never be updated using the push method, because it would have to be connected to the network at the exact time a push occurs. With the pull method, the laptop would automatically pull the configuration changes the next time it contacts the configuration server.

Pull also scales more manageably because each host can decide where to get its updates and can fail over to an alternative if the request times out or otherwise fails. In a push model where multiple central hosts attempt to push to the same client at once, conflicts are likely to occur. Cfengine's author Mark Burgess says: "Push methods are basically indistinguishable from an attack on the system. They attempt to remove each user's local right to decide about their own systems. In a world where we increasingly own the devices we use as personal objects, this all seems a bit like something from the cold war." Mark is speaking to the freedom that both the pull model and cfengine allow. In this book we set standardized schedules and policies on our systems, but this is strictly a local policy choice. The cfengine framework is remarkably free of assumptions and requirements, and you can use it to implement the appropriate policies for your site.

Although cfengine typically pulls from a server and executes at regular intervals (configurable, but defaults to once an hour), it also supports the ability to force updates to all or any subset of the systems on demand. Obviously, you will find this useful when you are performing mission-critical bug fixes (e.g., something else you did messed up a system or two and you need to fix it very quickly).

You can also run cfengine directly on each system by logging in and manually running the `cfagent` command. Cfengine follows a good theory in system-administration automation: the more ways you can initiate changes to a system, the better—as long as all changes are done in the same way. In other words, cfengine provides several methods for updating each system, but all of them use the same configuration files and operate exactly the same way (once initiated). We can thank `cfagent` for this standardization; it's always the program reading the configuration files and implementing the policy.

Delving into the Components of cfengine

The cfengine suite consists of several compiled programs. Modern systems have plenty of disk space to house the binaries locally. Some of the older cfengine documentation implied that it's wise to share out `/usr/local` or a similar directory through NFS—or Andrew File System (AFS) or Distributed File System (DFS)—and utilize the shared files from all systems. But most SAs today would see a real disadvantage to the single point of failure inherent in the remote mounting of critical software like cfengine, at least when disk space is as cheap as it is.

We'll list some of cfengine's programs here:

`cfagent`: The autonomous configuration agent (the heart of the framework). This command can be run manually (on demand), by `cfexecd` on a regular basis, and/or by `cfservd` when triggered by a remote `cfrun` invocation. The necessary and sufficient condition for using cfengine is to run `cfagent` somehow.

`cfservd`: The file-transfer and remote-activation daemon. You must run this on any cfengine file servers and on any system where you would like to execute `cfagent` remotely.

`cfexecd`: The execution and reporting daemon. You run this either as a daemon or as a regular cron job. In either case, it handles running `cfagent` and reporting its output.

`cfkey`: Generates public/private key pairs and needs to be run only once on every host.

`cfrun`: You can run this command from a remote system that will contact the clients (through `cfservd`) and tell them to execute `cfagent`.

For any given command, you can see a summary of its options by using the `-h` command-line option. When running a command, you can always specify the `-v` switch to see detail. For example, `cfagent` is nonverbose by default but will describe the actions it takes when the `-v` switch is used.

When debugging a program, you should use the `-d2` switch to view debugging information (and, for daemons, the `-d2` switch prevents them from detaching from the terminal).

Mapping the cfengine Directory Structure

You must install the binaries in a directory mounted on every host or install them independently on each host. Everything cfengine uses during its normal operation is located under the `/var/cfengine/` directory by default, although Debian and its derivative distributions use `/var/lib/cfengine2` by default. The directory's contents are as follows:

`bin/`: Important binaries (`cfagent`, `cfexecd`, and `cfservd`) are usually copied here to ensure they are available when needed. Normal operation doesn't require this, but the cfengine example documentation recommends it and many sites adhere to it.

`inputs/`: This is the standard location for all the configuration files cfengine needs. We will initially use three files from this directory—`cfagent.conf`, `cfservd.conf`, and `update.conf`—but we'll be expanding on those initial files quite a bit as the book progresses.

`outputs/`: This is where the output from each `cfexecd` run is logged.

`ppkeys/`: This is where this system's public and private keys, as well as other systems' public keys, are located.

Managing cfengine Configuration Files

Each system must have a minimal number of configuration files. You should place these in the `/var/cfengine/inputs/` directory on each system (`/var/lib/cfengine2/inputs` on Debian and derivatives), but you can maintain them in a central location and copy them (using a pull) to all client systems:

`update.conf`: This file *must* be kept simple. It is always parsed and executed by `cfagent` first. Its main job and intended usage is to copy the set of configuration files from the server. If any of the other configuration files contains an error, this file should still be able to update files so that the next run will succeed. If this file contains an error, you will have to fix any affected systems manually. At most sites this file should go unchanged for long periods of time once the cfengine infrastructure is initially set up.

`cfagent.conf`: This file contains the guts of your automation system—all the actions, group declarations, and so on. Throughout this book we utilize the `cfagent.conf` file as the starting point for all the included files (using cfengine's `import` feature) that make up the bulk of the configuration. This approach leads to more modular and easy-to-read configurations.

`cfservd.conf`: As the name suggests, this is the configuration file for the `cfservd` daemon. It defines which hosts can remotely execute `cfagent` and which remote hosts can transfer particular files.

You should manage the master copy of the configuration files with a source-control system such as Subversion. This way you have a record of any changes made and you can revert to an older configuration file if you introduce a problem into the system. If you feel that you don't have the time to set up Subversion or even CVS, then something extremely simple such as Revision Control System (RCS) will still be better than nothing. We'll

give examples on how to set up and use Subversion with cfengine in Chapter 11 (using cfengine itself to fully automate the process).

WHERE IS MY /VAR/CFENGINE/INPUTS DIRECTORY?

The `/var/cfengine/inputs` and `/var/cfengine` directories are not included with UNIX or Linux default installations. They might not even exist on systems with cfengine installed. Package-based cfengine installations often create some of the required cfengine directories, but usually not all.

It is up to you to create and configure the required directories and configuration files needed for your site's cfengine framework. We'll describe a simple cfengine site configuration later in this chapter, but the example is intended for demonstration purposes only. In Chapter 5, we'll configure a complete cfengine framework and build on it in later chapters.

For now, simply be aware that by default `cfagent` (or any other cfengine program) doesn't automatically configure a system in a way that satisfies all the prerequisites for running cfengine. You'll have to handle all the details yourself. If you're chomping at the bit to see how it's done, check out the `cf.preconf` preconfiguration script from Chapter 5.

Identifying Systems with Classes

The concept of classes is at the heart of cfengine. Each system belongs to one or more classes. Or, to put it another way: each time `cfagent` runs, many classes are defined based on a variety of different kinds of information. Each action in the configuration file can be limited only to certain classes. So, any given action could be performed on one host only, on hosts running a specific operating system, or on every host. Cfengine uses both built-in and custom classes, both of which will be discussed within this section.

Categorizing Predefined Classes

The host itself determines many of the classes that are defined—its architecture, hostname, IP address(es), and operating system. Several classes are also defined based on the current date and time.

To determine which standard classes are defined on any given system, run this command:

```
# cfagent -p -v | grep Defined
Defined Classes = ( 192_168_1 192_168_1_1 64_bit Day25 Debian_GNU_Linux_4_0__n__l_
Hr00 Hr00_Q4 June Min55_00 Min57_Q4 VMware Wednesday Yr2008 addr_ any campin_net
cfengine_2 cfengine_2_1 cfengine_2_1_20 compiled_on_linux_gnu debian debian_4
```

```
debian_4_0 fe80__290_27ff_fee8_9510 fe80__290_27ff_fee8_9511 ipv4_192 ipv4_192_168
ipv4_192_168_1 ipv4_192_168_1_1 linux linux_2_6_18_5_amd64 linux_x86_64
linux_x86_64_2_6_18_5_amd64
linux_x86_64_2_6_18_5_amd64__1_SMP_Sat_Dec_22_20_43_59_UTC_2007 net
net_iface_eth0 net_iface_eth1 net_iface_lo net_iface_ppp0
net_iface_ppp1 foo foo_campin_net foo_int x86_64 )
```

As you can see, this example system contains quite a number of predefined classes. They fall into a few categories:

Operating System: `debian debian_4 debian_4_0`

Kernel: `linux linux_2_6_18_5_amd64`

Architecture: `x86_64 linux_x86_64 linux_2_6_18_5_amd64`

Hostname: `campin_net foo foo_campin_net`

IP Address: `ipv4_192 ipv4_192_168 192_168_1 192_168_1_1`

Date/Time: `Wednesday Yr2008 Hr00 Hr00_Q4 June Min55_00 Min57 Day25`

Every system is a member of the `any` class, for obvious reasons. As you can see, cfengine provides good granularity with its default class assignments. We cannot possibly list all the classes that could be assigned to your systems, so you will have to check the list on each of your systems (or, at least, each type of system on your network).

The time-related classes probably require some additional explanation. The `Min55_00` class specifies the current five-minute range. The `Q4` class is always set in the last quarter of the hour. The `Hr00_Q4` class says you are currently in the last quarter of the midnight hour (it's time for bed).

Defining Custom Classes

Custom classes are defined in the `classes` section of the `cfagent.conf` configuration file. Here is an example:

```
classes:
   # Check to see if X11R6 is installed
   X11R6                                    = ( '/usr/bin/test -d /usr/X11R6' )

   # Use built-in directory check for a local program directory
   have_usr_pkg_ganglia_3_0_7        = ( IsDir(/usr/pkg/ganglia-3.0.7) )
```

```
# Mail servers must be explicitly defined at our site
mail = ( mail1 mail2 )

# DNS and web servers are obvious by their configuration files
dns = ( '/usr/bin/test -f /etc/named.conf' )
web = ( '/usr/bin/test -f /etc/httpd/conf/httpd.conf' )

# We have a large number of web servers, let's break them out into
# groups of ten for when we implement changes across them
first_ten_webs        = ( RegCmp("webserver[0-9]i","${host}") )
second_ten_webs   = ( RegCmp("webserver1[0-9]i","${host}") )
third_ten_webs         = ( RegCmp("webserver2[0-9]i","${host}") )
fourth_ten_webs    = ( RegCmp("webserver3[0-9]i","${host}") )
fifth_ten_webs        = ( RegCmp("webserver4[0-9]i","${host}") )
# ...and so on...

# Any critical servers are a member of this class
critical = ( mail dns web )
```

When a class definition contains a quoted string, that is a command to be executed. If the command returns an exit code of `0`, then this system will be part of that class.

Class definitions can (and often do) list other classes. If a system is a member of any of the listed classes, then it is also a member of the new class.

Some cfengine commands can define new classes in certain situations. If, for example, a particular drive is too full, a command can define a class and print a warning. Other parts of the configuration file might take further action if that class is defined. We'll explore this quite a bit in later chapters.

Finding More Information About Cfengine

You can download cfengine from its web site: `http://www.cfengine.org/`. The web site includes a tutorial, a comprehensive reference manual, mailing lists, and archives. Two additional web sites are useful. The first, `http://www.cfengine.com`, is a commercial venture started by the cfengine author and some colleagues. It contains enhanced documentation in return for a little information about how you use cfengine. The second is `http://www.cfwiki.org`, a community-run site with a lot of useful tips and tricks for dealing with cfengine, generally from very experienced cfengine users.

You should also examine the large number of sample configuration files included with the cfengine distribution.

Learning the Basic Setup

Within this section, we'll illustrate and discuss a simple cfengine setup that will provide a good framework for customization and expansion. These simple configuration files will not make many changes to your systems, but they will still show some of the power of cfengine.

This simple setup includes one central server and one other host. With cfengine, all hosts are set up identically (even with only slight differences on the server), so you could extend this example to any number of systems. We would recommend, though, that you start out with just two systems. Once you get cfengine up and running on those systems, expanding the system to other hosts is easy enough. In later chapters we'll completely overhaul this basic configuration to scale up to complete site-wide management.

Setting Up the Network

Before starting with cfengine, you should make sure your network is properly prepared. Using cfengine with dynamic IP addresses is difficult because cfengine utilizes two-way authentication during network communications. Even if you use the Dynamic Host Configuration Protocol (DHCP) to assign addresses to some or all of your systems, it should always assign the same IP address to systems that you'll control with cfengine. In other words, it doesn't matter which method you use to assign the IP addresses, as long as the IP address for each system to be managed stays consistent. Cfengine has configuration directives allowing it to understand that hosts on certain IP ranges use dynamic IP addresses, but this defeats the two-way trust mechanism. You should avoid dynamic IP addresses if possible.

The next task is to make sure your Domain Name System (DNS) is properly configured for your hosts. Each host should have a hostname, and a DNS lookup of that hostname should return that host's IP address. In addition, if that IP address is looked up in DNS, the same hostname should be returned.

If this setup is not possible, we recommend that you add every host to the `/etc/hosts` file on every system, although if your DNS isn't properly configured you'll have pain in other areas. If you are using a multihomed host, you must pay attention to which IP address will be used when your host is communicating with other cfengine hosts.

Running Necessary Processes

In the simplest setup, you can use cfengine by running `cfagent` on each system manually. You will, however, benefit more from running one or two daemons on each system.

The cfexecd Daemon

Although you could, theoretically, run `cfagent` only on demand, you're better off running it automatically on a regular basis. This is when `cfexecd` comes in handy; it runs as a daemon and executes `cfagent` on a regular, predefined schedule. You can modify this schedule by adding time classes to the `schedule` setting in the `control` block in `cfagent.conf`. The default setting is `Min00_05`, which means `cfagent` will run in the first five minutes of every hour. To run twice per hour, for example, you could place the following line in the `control` section of `cfagent.conf`:

```
schedule = ( Min00_05 Min30_35 )
```

The `cfexecd` daemon does not have its own configuration file; it uses settings intended for `cfexecd` in the `cfagent.conf` file.

You can also run `cfexecd` on a regular basis using the system's cron daemon. You could add the following entry to the system crontab (usually `/etc/crontab`) to execute (and report) `cfagent` every hour:

```
0 * * * * root /usr/local/sbin/cfexecd -F
```

The `-F` switch tells the `cfexecd` program not to go into daemon mode because it is being run by cron.

For increased reliability, you can run `cfexecd` as a daemon and also run it from cron (maybe once per day). You can then, in `cfagent.conf`, check for the crontab entry and check whether the `cfexecd` daemon is running. The following lines, if placed in `cfagent.conf`, perform these checks and correct any problems:

```
editfiles:
   { /etc/crontab
      AppendIfNoSuchLine "0 * * * * root /var/cfengine/bin/cfexecd -F"
   }

processes:
   "cfexecd" restart "/var/cfengine/bin/cfexecd"
```

With this technique, if either of the methods is not working properly, the other method ultimately repairs the problem.

The cfservd Daemon

The `cfservd` daemon is not required on all systems. It needs to run on any cfengine file servers, which, in our case, is the central configuration server only. It also allows you to execute `cfagent` from other systems remotely. If you want this functionality (which

we recommend), then `cfservd` needs to be running on every system. In either case, you should always check whether it's running by using the following entry in `cfagent.conf`:

```
processes:
   "cfservd" restart "/var/cfengine/bin/cfservd"
```

Running `cfservd` on all hosts presents little risk, and it allows added flexibility when you later need to retrieve information from client systems using cfengine. Make sure that your access controls (the `admit` lines in `cfservd.conf`) don't allow access to unnecessary hosts. Only access explicitly defined in `cfservd.conf` is allowed, so `cfservd` is safe by default.

Creating Basic Configuration Files

You need to place your configuration files in the master configuration directory on the configuration server (as we'll explain in the next section). These common files will be used in their exact original form on every server in your network.

Example cfservd.conf

This is the configuration file for the `cfservd` daemon. It allows clients to transfer the master set of configuration files and also allows you to execute `cfagent` remotely using `cfrun`. Obviously only one system needs to allow access to the central configuration files (the server), but having `cfservd` allow access to those files doesn't hurt anything on other systems (because those systems don't have the files there to copy). All systems, however, can benefit from allowing remote `cfagent` execution, because it allows you to execute `cfagent` on demand from remote systems.

So, you can use the following `cfservd.conf` on all your systems:

```
control:
   domain                        = ( mydomain.com )
   AllowUsers                    = ( root )
   cfrunCommand                  = ( "/var/cfagent/bin/cfagent" )
   TrustKeysFrom                 = ( 10.1.1 )
   AllowConnectionsFrom          = ( 10.1.1 )
   AllowMultipleConnectionsFrom = ( 10.1.1 )

admit:
   /usr/local/var/cfengine/inputs *.mydomain.com
   /var/cfagent/bin/cfagent       *.mydomain.com
```

The cfrunCommand setting specifies the location of the cfagent binary to be run when a connection from cfrun is received. The admit section is important because it specifies which hosts have access to which files. You must grant access to the central configuration directory and the cfagent binary. You also need to grant access to any other files that clients need to transfer from this server.

Basic update.conf

You should keep the update.conf file as simple as possible and change it rarely, if ever. The cfagent command parses and executes the update.conf file before it does the same to cfagent.conf. If you put out a bad cfagent.conf, the next time the clients execute cfagent they get the new version because their update.conf file is still valid. Distributing a bad update.conf would be a bad idea, so we recommend testing any changes thoroughly before you place the file in the central configuration directory. We also recommend you include some comments in the file that warn about problems resulting from errors.

Again, the update.conf file is run on every host, including the server. The cfagent command is smart enough to copy the files locally (instead of over the network) when running on the configuration server. Several variables are defined in the control section and then used in the copy section. You can accomplish variable substitution with either the $(var) or ${var} sequence:

```
1 control:
2  actionsequence  = ( copy tidy )
3  domain          = ( mydomain.com )
4  workdir         = ( /var/cfengine )
5  policyhost      = ( server.mydomain.com )
6  master_cfinput  = ( /usr/local/var/cfengine/inputs )
7  cf_install_dir  = ( /usr/local/sbin )
8
9 copy:
10      $(cf_install_dir)/cfagent    dest=$(workdir)/bin/cfagent
11                                   mode=755
12                                   type=checksum
13
14      $(cf_install_dir)/cfservd    dest=$(workdir)/bin/cfservd
15                                   mode=755
16                                   type=checksum
17
```

```
18      $(cf_install_dir)/cfexecd    dest=$(workdir)/bin/cfexecd
19                                   mode=755
20                                   type=checksum
21
22      $(master_cfinput)            dest=$(workdir)/inputs
23                                   r=inf
24                                   mode=644
25                                   type=binary
26                                   exclude=*.lst
27                                   exclude=*~
28                                   exclude=#*
29                                   server=$(policyhost)
30                                   trustkey=true
31
32 tidy:
33   $(workdir)/outputs pattern=* age=7
```

Line 5: You should replace the string `server.mydomain.com` with the hostname of your configuration server.

Line 6: This is the directory on the master configuration server that contains the master configuration files. It requires the `admit` entry in `cfservd.conf`.

Line 7: This is the location, on every server, of the cfengine binaries.

Line 23: This specifies that you should copy the source directory recursively to the destination directory with no limit on the recursion depth.

Line 25: This option is misleading at first. It specifies that you should compare any local file byte by byte with the master copy to determine if an update is required.

Line 29: This option causes the files to be retrieved from the specified server.

Line 33: This command in the `tidy` section removes any `outputs/` directory files that have not been accessed in the last seven days.

The permissions (modes) on each file are checked on each run even if the file already exists.

Framework for cfagent.conf

The `cfagent.conf` file is the meat of the cfengine configuration. You should make any change on any system using this file (or files imported from this file, as demonstrated in later chapters). We'll keep the sample `cfagent.conf` file simple for demonstration purposes; don't use it as is in a real-world scenario.

If you call any scripts from cfengine and those scripts produce any output, that output will be displayed (when executed interactively) or logged and e-mailed (when executed from `cfexecd`). Executing `cfagent` every hour is typical, so any scripts should produce output only if there is a problem or if something changed and the administrator needs to be notified:

```
1 control:
2 actionsequence  = ( files directories tidy disable processes )
3 domain          = ( mydomain.com )
4 timezone        = ( EDT EST )
5 access          = ( root )
6 # Where cfexecd sends reports
7 smtpserver      = ( mail.mydomain.com )
8 sysadm          = ( root@mydomain.com )
9
10 files:
11 /etc/passwd mode=644 owner=root action=fixall
12 /etc/shadow mode=600 owner=root action=fixall
13 /etc/group  mode=644 owner=root action=fixall
14
15 directories:
16 /tmp mode=1777 owner=root group=root
17
18 tidy:
19 /tmp recurse=inf age=7 rmdirs=sub
20
21 disable:
22 /root/.rhosts
23 /etc/hosts.equiv
24
25 processes:
26 "cfservd" restart "/var/cfengine/bin/cfservd"
27 "cfexecd" restart "/var/cfengine/bin/cfexecd"
```

Line 2: The `actionsequence` command is very important and easy to overlook. You must list each section that you wish to process in this variable. If you add a new section but forget to add it to this list, it will not be executed.

Line 4: Cfengine will make sure the system is configured with one of the time zones in this list.

Line 10: This section checks the ownership and permissions of a few important system files and fixes any problems it finds.

Line 15: This section checks the permissions on the `/tmp/` directory and fixes them, if necessary. It also creates the directory if necessary.

Line 18: This section removes everything from the `/tmp/` directory that has not been accessed in the past seven days. It removes only subdirectories of `/tmp/`—not the directory itself.

Line 21: These files are disabled for security reasons, and renamed if found. If they are executable, the executable bit is unset.

Line 25: This section verifies that the `cfservd` and `cfexecd` daemons are running and starts them if they are not.

Creating the Configuration Server

The configuration server contains the master copy of the cfengine configuration files. It also processes those configuration files on a regular basis as all the client systems do. The server must run a properly configured `cfservd` so the client systems can retrieve the master configuration from the system.

The configuration server needs a special place to keep the master cfengine configuration files. In this example, that directory is `/usr/local/var/cfengine/inputs/`. It could be any directory, but not `/var/cfengine/inputs/` because the master host copies the files to that directory when executing, just like every other host.

Like all systems, the server should also run `cfexecd` either as a daemon or from cron (or, even better, both).

Now we'll discuss generating server keys. You need to run `cfkey` on the server system to create its public- and private-key files. These files will be in the `/var/cfengine/ppkeys/` directory (or `/var/lib/cfengine2/ppkeys` on Debian and its derivatives) and will be named `localhost.priv` and `localhost.pub`.

The server also needs each client's public key in the appropriate file, based on the client's IP address as described in the next section. You can populate the file automatically upon the first connection of a remote host, similar to the way most SSH clients prompt the user to store a remote SSH server's host key for validation during subsequent connections. The `TrustKeysFrom` value in `cfservd.conf` and the `TrustKey` value in `copy` statements control server and client trust settings. We believe that trusted initial key exchange is a good idea, so we'll use that technique throughout this book.

CFENGINE AND ROOT PRIVILEGES

Cfengine does not require root privileges. The demonstration configuration files in this chapter perform operations that require root privileges, such as enforcing restrictive permissions on the `/etc/passwd` file.

You are encouraged to run cfengine as a nonprivileged user. The `cfagent` program defaults to the `~/.cfagent` directory instead of `/var/cfengine` for a working directory. The privileges needed are entirely dependent on the actions taken in the configuration files.

Preparing the Client Systems

Each client system is relatively simple to configure. Once you install the cfengine binaries, you need to generate the host's public and private keys (as discussed in this section). You also must copy the `update.conf` file from the master server manually and place it in `/var/cfengine/inputs/`. Once this file is in place, you should manually run `cfagent` to download the remaining configuration files and complete system configuration. We'll explore automated ways to handle this later on, but for now we're keeping things simple.

Each client should run `cfexecd` either as a daemon or from cron. You probably also want to run `cfservd` on each client to allow remote execution of `cfagent` using `cfrun`. Assuming automatic `cfexecd` execution has already configured in the `cfagent.conf` file on the server, these daemons will be started after the first manual execution of `cfagent`. From that point on, cfengine will be self-sustaining. It will update its own config files, and you can even use it to change its own behavior. For example, you can configure initialization scripts to start cfengine, change its schedule using `cfexecd` (from `cfagent.conf`), or even upgrade the cfengine binaries themselves.

You need to run `cfkey` on each client system before you run `cfagent` for the first time. This creates `localhost.priv` and `localhost.pub` in `/var/cfengine/ppkeys/`. You don't need to copy the central server's public key to the client manually. If the server's IP address is `10.1.1.1`, then when `cfagent` is run and it sees the `trustkey=true` entry in the `copy` section of `update.conf`, it will copy the server's public key to `root-10.1.1.1.pub` in the `/var/cfengine/ppkeys/` directory on the client. From that point on, the key is trusted and expected not to change for the host at IP `10.1.1.1`. If the key does change, by default cfengine will break off communications with the host when the key validation fails.

Debugging cfengine

When you are trying to get cfengine up and running, you will probably face a few problems. Network problems are common, for example, when you transfer configuration files from the master server and initiate `cfagent` execution on remote systems with `cfrun`.

For any network problems, you should run both the server (`cfservd`) and the client (`cfrun` or `cfagent`) in debugging mode. You can accomplish this by using the `-d2` command-line argument. For `cfservd`, this switch not only provides debugging output, but it also prevents the daemon from detaching from the terminal.

When you are trying something new in your `cfagent.conf` file, you should always try it with the `--dry-run` switch to see what it would do without making any actual changes. The `-v` switch is also useful if you want to see what steps `cfagent` is taking. If it is not doing something you think it should be doing, the verbose output will probably tell you why.

If you are making frequent changes or trying to get a new function to work properly, you probably want to be able to run `cfagent` repeatedly on demand. By default, `cfagent` will not do anything more frequently than once per minute. This helps prevent both intentional and accidental denial-of-service attacks on your systems.

You can eliminate this feature for testing purposes by placing this line in the `control` section of `cfagent.conf`:

```
IfElapsed = ( 0 )
```

You might also find it helpful to run only a certain set of actions by using the `--just` command-line option. For example, to check only on running processes, you can run the command `cfagent --just processes`.

Creating Sections in cfagent.conf

Cfengine 2.2.7 offers 34 possible sections in `cfagent.conf`; we'll cover some of these sections in this chapter, some later in the book, and some not at all. For additional information, refer to the comprehensive reference manual on the cfengine web site at `http://www.cfengine.org/`. Plus, read Brendan Strejcek's blog post about picking and choosing from the available cfengine feature set: "Cfengine Best Practices" (find it at `http://meta-admin.blogspot.com/2005/12/cfengine-best-practices.html`). It is also prominently displayed on the `http://www.cfwiki.org` site.

Every section can contain one or more class specifications. For example, the `files` section could be:

```
files:
   /etc/passwd

   any::
      /etc/group
   redhat::
      /etc/redhat-release
   solaris::
      /etc/vfstab
```

Both `/etc/passwd` and `/etc/group` will be processed on all hosts (because the default group is `any` when none is specified). In addition, the `/etc/redhat-release` file will be checked only on systems running Red Hat Linux, and the `/etc/vfstab` will be checked only if the operating system is Sun Microsystems' Solaris.

You can use the period (`.`) to "and" groups together, whereas you can use the pipe character (`|`) to "or" groups together. The exclamation character (`!`) can invert a class and parentheses (`()`) can group classes. Here is a complex example:

```
files:
   (redhat|solaris).!Mon::
      /etc/passwd
```

In this case, the `/etc/passwd` file will be checked only if the operating system is Red Hat Linux or Solaris and today is not a Monday.

Using Classes in cfagent.conf

You can use the `classes` section to create user-defined classes, as we described earlier. Determine a system's class membership using a shell command:

```
classes:
   X11R6 = ( '/usr/bin/test -d /usr/X11R6' )
```

If the command returns `true` (exit code `0`), this machine will be a member of that class (for the current run). You can also define classes to contain specific hosts or any hosts that are members of another existing class:

```
classes:
   critical = ( host1 host2 web_servers )
```

Here are a few more possibilities that you could place in the `classes` section:

```
classes:
   notthis = ( !this )
   ip_in_range = ( IPRange(129.0.0.1-15) )
   ip_in_range = ( IPRange(129.0.0.1/24) )
```

The copy Section

The `copy` section is one of the most commonly utilized in cfengine. Cfengine can copy files between mounted filesystems on the client (whether local filesystems or remote shares), as well as from remote cfengine servers (systems running `cfservd`).

Cfengine `copy` operations prevent corruption or other errors in copied files by first copying the file to a file named *`file`*`.cfnew` on the local filesystem (where *`file`* is the name of the file being copied), then renaming the file quickly into place. This technique wards off problems resulting from partially copied or corrupted files being left in place (due to full disk, network interruption, system error, and so on).

You can choose from many configurable parameters for the `copy` section, but here are some used for the common scenario of copying an entire directory tree of programs:

```
copy:
        debian.i686::
                /usr/local/masterfiles/ganglia-3.0.7-debian-i686
                                dest=/usr/pkg/ganglia-3.0.7
                                mode=755
                                recurse=inf
                                owner=root
                                group=root
                                type=checksum
                                server=masterhost.mydomain.com
                                encrypt=true
```

This `copy` action is performed only on hosts with both the `debian` and `i686` classes defined. Cfengine sets these classes automatically based on system attributes. Take advantage of these automatic classes to ensure that binary files of the correct type are copied.

The directory `/usr/local/masterfiles/ganglia-3.0.7-debian-i686` is copied from a remote host named `masterhost.mydomain.com`. The remote host needs to run the `cfservd` daemon with access controls that allow the client system to access the source files. The setting `recurse=inf` specifies that the source directory be copied recursively—which means to recurse infinitely and copy all subdirectories, as well as all the files in the source directory. Instead of `inf`, you can specify a number to control the recursion depth.

The copy type is set to checksum, which means that an MD5 checksum is computed for each source and destination file, and that the destination file is updated only if the checksums don't match.

The owner and group are set to root on the destination, regardless of the ownership settings on the source files. We recommend explicitly setting ownership of copied files in every copy section, for security reasons. We set all the copied files to have the mode 755, which will be executable for all users but writable only by the owner.

We set the network communications to be encrypted with the setting encrypt=true, because we like to keep all our administrative traffic private.

The directories Section

The directories section checks for the presence of one or more directories. Here is an example:

```
directories:
   /etc mode=0755 owner=root group=root syslog=true
      inform=true
   /tmp mode=1777 owner=root group=root define=tmp_created
```

If either directory does not exist, it will be created. The section will also check for permissions and ownership and correct them, if necessary. In this example, the administrator will be informed (through mail or the terminal) and a syslog entry will be created if the /etc/ directory does not exist, or if it has incorrect permissions and/or ownership.

For the /tmp/ directory, the class tmp_created is defined if the directory was created. You could then use this class in another section of the configuration file to perform certain additional tasks.

The disable Section

The disable section causes cfengine to disable any number of files. This simple section disables two files that you probably don't want around because they allow access to the root account through Remote Shell (RSH) using only the source IP address as authentication:

```
disable:
   /root/.rhosts
   /etc/hosts.equiv
```

If either of these files exists, the section will disable it by renaming it with a .cfdisabled extension. It will also change the permissions to 0600, so you could use it to disable executables. At one point, for example, Solaris had a local root exploit with

the `eject` command. Until there was a patch available, you could have disabled that command using the following sequence:

```
disable:
   solaris::
      /usr/bin/eject inform=true syslog=true
```

This would not only disable the command, but it also would inform the administrator and make a log entry using syslog.

Suppose you want to remove `/etc/httpd/conf/httpd.conf` if it exists and create a symbolic link pointing to the master file `/usr/local/etc/httpd.conf`. The following command sequence can accomplish this task:

```
disable:
   /etc/httpd/conf/httpd.conf type=file define=link_httpd_conf
links:
   link_httpd_conf::
      /etc/httpd/conf/httpd.conf -> /usr/local/etc/httpd.conf
```

The `disable` section would remove the file only if it is a normal file (and not a link, for example). If the file is disabled, the `link_httpd_conf` class will be defined. Then, in the `links` section, a symbolic link will be created in its place.

Remember that cfengine does not execute these sections in any predefined order. The `actionsequence` setting in the `control` section controls the order of execution. So, for this example, make sure the file is disabled before the symbolic link is created:

```
control:
   actionsequence = ( disable links )
```

Managing the proper `actionsequence` order is something of an art form. As your cfengine configuration files grow in size, number, and complexity, you'll usually find both advantages and disadvantages to a particular ordering in your `actionsequence`. Once you find the order that works best in most cases, you'll want to keep the ordering in mind as you write configurations that set classes to trigger other actions, and try to line up your dependencies accordingly. To see what we mean, skip ahead to the section on NTP client configuration in Chapter 7. When NTP configuration files are copied:

1. A class called `restartntpd` is defined.
2. In the `shellcommands` section, the NTP daemon is restarted with a startup script.
3. If the `shellcommands` section doesn't come after the `copy` section in the `actionsequence`, then this sequence of events can't happen as planned.

You can also use the `disable` section to rotate log files. The following sequence rotates the web-server access log if it is larger than 5MB and keeps up to four files:

```
disable:
   /var/log/httpd/access_log size=>5mbytes rotate=4
```

The editfiles Section

The `editfiles` section can be the most complex section in a configuration file. You can choose from approximately 100 possible commands in this section. These commands allow you to check and modify text files (and, in a few cases, binary files).

We won't go into great detail on using `editfiles` because you should use this section rarely. We generally prefer copying complete files to clients. This way, we can keep our systems' file contents properly in a revision-control system. You can use a script to create the proper file contents for different systems in a central location, after which classes based on system attributes (e.g., Debian vs. Solaris, or web server vs. mail server) can copy the correct file into place.

That said, you'll encounter some situations where doing a direct file modification on clients can be appropriate and useful, such as maintaining a message of the day or updating files that were previously updated by a manual process and that differ from system to system. The examples here are for demonstration purposes.

Here is an example `editfiles` section:

```
editfiles:
   { /etc/crontab
      AppendIfNoSuchLine "0 * * * * root /usr/local/sbin/cfexecd -F"
   }
```

This command adds the specified line of text to `/etc/crontab`. This makes sure that cron runs `cfexecd` every hour. You also might want to make sure that other hosts can access and use the printers on your printer servers:

```
editfiles:
   PrintServer::
      { /etc/hosts.lpd
         AppendIfNoSuchLine "host1"
         AppendIfNoSuchLine "host2"
      }
```

In your environment, perhaps a standard port is used for another purpose. For example, you might want to rename port 23 to `myservice`. To do this, you could change its label in `/etc/services` on every host:

```
editfiles:
   { /etc/services
      ReplaceAll "^.*23/tcp.*$" With "myservice 23/tcp"
   }
```

If you are using `inetd` and want to disable the `telnet` application, for example, you could comment out those lines in `/etc/inetd.conf`:

```
editfiles:
   { /etc/inetd.conf
      HashCommentLinesContaining "telnet"
      DefineClasses "modified_inetd_conf"
   }
processes:
   modified_inetd_conf::
      "inetd" signal=hup
```

Any line containing the string `telnet` will be commented out with the `#` character (if not already commented). If you make such a change, the `inetd` process is sent the HUP signal in the `processes` section.

The files Section

The `files` section can process files (and directories) and check for valid ownership and permissions. It can also watch for changing files. Here is a simple example:

```
files:
   /etc/passwd mode=644 owner=root group=root action=fixall checksum=md5
   /etc/shadow mode=600 owner=root group=root action=fixall
   /etc/group mode=644 owner=root group=root action=fixall

   web_servers::
      /var/www/html r=inf mode=a+r action=fixall
```

We accomplish several tasks with these entries. On every system, the `files` section checks and fixes the ownership and permissions on `/etc/passwd`, `/etc/shadow`, and `/etc/group`. It also calculates and records the MD5 checksum of `/etc/passwd`.

On any system in the class `web_servers`, the permissions on `/var/www/html/` are checked because that is the standard web-content directory across all hosts at our site. Your site might (and probably will) differ. The directory is scanned recursively and all files and directories are made publicly readable. The execute bits on directories will also be set

according to the read bits; because we requested files to be publicly readable, directories will also be publicly executable.

The `checksum` option requires a little more explanation. The file's checksum is stored, so the administrator will be warned if it changes later. In fact, the administrator will be warned every hour unless you configure cfengine to update the checksum database. In that case, the administrator will be notified only once, and then the database will be modified. You can enable this preference by adding the following command in the `control` section:

```
control:
   ChecksumUpdates = ( on )
```

Here are some other options you might want to use in the `files` section:

`links`: You can set this option to `traverse` to follow symbolic links pointing to directories. Alternatively, you can set it to `tidy` to remove any dead symbolic links (links that do not point to valid files or directories).

`ignore`: You can specify the `ignore` option multiple times. Cfengine version 2 requires a pattern or a simple string, but not a regular expression. Cfengine version 3 repairs this inconsistency (look for more information on cfengine 3 at the end of this chapter). Any file or directory matching this pattern is ignored. For instance, `ignore="^\."` would ignore all hidden files and directories.

`include`: If any `include` options are listed, any files must match one of these regular expressions in order to be processed.

`exclude`: Any file matching any of the `exclude` regular expressions will not be processed by cfengine.

`define`: If the directives in this `files` section result in changes being made to any listed files, a new class will be defined. You can also list several classes, separated by colons.

`elsedefine`: Similar to define, but here new classes are defined if *no changes* are made to the listed files. You can also list several classes, separated by colons.

`syslog`: When set to `on`, cfengine will log any changes to the system log.

`inform`: When set to `on`, cfengine will log any changes to the screen (or send them by e-mail if `cfagent` is executed by `cfexecd`).

The links Section

With the `links` section, `cfagent` can create symbolic links:

```
links:
   /usr/tmp -> ../var/tmp
   /var/adm/messages  ->! /var/log/messages
   /usr/local/bin +> /usr/local/lib/perl/bin
```

In this example, the first command creates a symbolic link (if it doesn't already exist) from `/usr/tmp` to `../var/tmp` (relative to `/usr/tmp`).

The second command creates a symbolic link from `/var/adm/messages` to `/var/log/messages` *even if there is already a file* located at `/var/adm/messages`. The bang (`!`) overrides the default safety mechanism built into cfengine around creating symbolic links.

The third command creates one link in `/usr/local/bin/` pointing to each file in `/usr/local/lib/perl/bin/`. Using this technique, you could install applications in separate directories and create links to those binaries in the `/usr/local/bin/` directory.

The `links` section offers plenty of possible options, but they are rarely used in practice so we won't cover them in this book. See the cfengine reference manual for more information.

The processes Section

You can monitor and manipulate system processes in the `processes` section. Here is an example from earlier in this chapter:

```
processes:
   "cfservd" restart "/var/cfengine/bin/cfservd"
   "cfexecd" restart "/var/cfengine/bin/cfexecd"
```

For the `processes` section, cfengine runs the `ps` command with either the `-aux` or `-ef` switch (as appropriate for the specific system). This output is cached and the first part of each command in the `processes` section is interpreted as a regular expression against this output. If there are no matches, the `restart` command is executed.

You can specify the following options when using the `restart` facility: `owner`, `group`, `chroot`, `chdir`, and/or `umask`. These affect the execution environment of the new process as started by the `restart` command.

You can also send a signal to a process:

```
processes:
   "httpd" signal=hup
```

This signal would be sent on every execution to any processes matching the regular expression `httpd`, so you probably don't want to use it as is. It is also possible to specify limits on the number of processes that can match the regular expression. If you wanted to ensure there were no more than ten `httpd` processes running at any given time, for example, you would use this code:

```
processes:
   "httpd" action=warn matches=<10
```

The shellcommands Section

For some custom and/or complex operations, you will need to execute one or more external scripts from cfengine. You can accomplish this with the `shellcommands` section. Here is an example:

```
shellcommands:
   all::
      "/usr/bin/rdate -s ntp1" timeout=30
   redhat.Hr02_Q1::
      "/usr/local/sbin/log_packages" background=true
```

On all systems, the `rdate` command is executed to synchronize the system's clock. Cfengine terminates this command in 30 seconds if it has not completed. On systems running Red Hat Linux, a script runs between 2:00 a.m. and 2:15 a.m. to log the currently installed packages. This command is placed in the background and cfengine does not wait for it to complete. This way, cfengine can perform other tasks or exit while the command is still running.

You can specify the following options to control the environment in which the command is executed: `owner`, `group`, `chroot`, `chdir`, and/or `umask`.

If these scripts want to access the list of currently defined classes, they can look in the `CFALLCLASSES` environment variable. Each active class will be listed, separated by colons.

Scripts, by default, are ignored when cfengine is performing a dry run (with `--dry-run` specified). You can override this setting by specifying `preview=true`. The script should not, however, make any changes when the class `opt_dry_run` is defined.

Using cfrun

The `cfrun` command allows you to execute `cfagent` on any number of systems on the network. It requires a configuration file in the current directory named `cfrun.hosts` (or a file specified with the `-f` option). The `cfrun.hosts` file can be as simple as this:

```
domain=mydomain.com
server.mydomain.com
client1.mydomain.com
client2.mydomain.com
```

Apart from the domain setting, this file is just a list of every host, including the configuration server. You can also have the output logged to a series of files (instead of being displayed to the screen) by adding these options to the top of the file:

```
outputdir=/tmp/cfrun_output
maxchild=10
```

This code tells `cfrun` to fork up to ten processes and place the output for each host in a separate file in the specified directory. You can normally run `cfrun` without arguments. If you do want to specify arguments, use this format:

```
cfrun CFRUN_OPTIONS HOSTS -- CFAGENT_OPTIONS -- CLASSES
```

`CFRUN_OPTIONS` is, quite literally, optional, and can contain any number of options for the `cfrun` command. Next, you can specify an optional list of hostnames. If some hostnames are specified, only those hosts will be contacted. If no hosts are specified, every host in the `cfrun.hosts` file will be contacted.

After the first `--`, you must place all options you want to pass to the `cfagent` command run on each remote system. After the second `--` is an optional list of classes. If some classes are specified, only hosts that match one of these classes will execute `cfagent` (although each host is contacted because each host must decide if it matches one of the classes).

Looking Forward to Cfengine 3

Cfengine 3 has been in the design phase for several years. It is a complete rewrite of the cfengine suite, but more important, it involves a new way of thinking about system management.

Cfengine 3 is built around a concept called "Promise Theory." This concept might sound difficult to grasp, but it's actually quite intuitive. With cfengine 3, you'll describe the *desired state* of your systems instead of the *changes* to your systems. The desired state is expressed as a collection of promises, and in the words of the cfengine author Mark Burgess, allows us to focus on the good instead of the bad.

The Cfengine.org web site has a thorough introduction to cfengine 3, as well as source code to the current snapshot of cfengine 3 development: `http://www.cfengine.org/cfengine3.php`.

We encourage you to familiarize yourself with the next evolutionary steps in cfengine for two reasons:

1. Familiarity with the new concepts and syntax will make it easier to migrate from version 2 to version 3 when the time comes.

2. Experimenting with the current feature set and implementation allows you to suggest enhancements or bug fixes. Making suggestions helps the people working on cfengine 3 and "gives back" to the people who gave us cfengine in the first place.

Using cfengine in the Real World

In this chapter, we covered the core concepts of cfengine and demonstrated basic usage with a collection of artificial configuration files. This information arms you with the knowledge you need to work through the remainder of this book.

The use of demonstration configuration files and imaginary scenarios ends here. Throughout the rest of this book, we will operate on a real-world infrastructure that we build from scratch. Every configuration setting and modification that we undertake will be one more building block in the construction of a completely automated and fully functional UNIX infrastructure.