

# Neil H. Watson

## UNIX Consultant

[Home](#) | [Services](#) | [Contact](#) | [Blog](#) | [News](#) | [Resume](#)

## Blog

# Cfengine Cookbook

Neil H. Watson

## Abstract:

This white paper offers solutions to system administration problems using the configuration management tool Cfengine.

## Contents

- [Introduction](#)
- [Help with Cfengine](#)
  - [Problem](#)
  - [Solution](#)
- [Initial setup](#)
  - [Problem](#)
  - [Solution](#)
- [Bootstrapping clients](#)
  - [Problem](#)
  - [Solution](#)
  - [Gravy](#)
- [Version Control](#)
  - [Problem](#)
  - [Solution](#)
  - [Gravy](#)
- [DMZ mirror](#)
  - [Problem](#)
  - [Solution](#)
- [Organizing Cfengine files](#)
  - [Problem](#)
  - [Solution](#)
- [Custom classes](#)

- [Problem](#)
  - [Solution](#)
- [Package management](#)
  - [Problem](#)
  - [Solution](#)
  - [Gravy](#)
- [Distributing public keys](#)
  - [Problem](#)
  - [Solution](#)
  - [Gravy](#)
- [CRON tables](#)
  - [Problem](#)
  - [Solution](#)
- [DNS services](#)
  - [Problem](#)
  - [Solution](#)
- [Process control](#)
  - [Problem](#)
  - [Solution](#)
- [Editing files](#)
  - [Problem](#)
  - [Solution](#)
- [Editing complex host specific files](#)
  - [Problem](#)
  - [Solution](#)
- [Application deployment](#)
  - [Problem](#)
  - [Solution](#)
  - [Gravy](#)
- [Pushing changes immediately](#)
  - [Problem](#)
  - [Solution](#)
- [Running manual commands on all clients](#)
  - [Problem](#)
  - [Solution](#)
- [Redundancy and load balancing](#)
  - [Problem](#)
  - [Solution](#)
  - [Caveat](#)
- [Security policies](#)

- [Problem](#)
- [Solution](#)
  
- [Audits](#)
  - [Problem](#)
  - [Solution](#)
  
- [Disaster recovery](#)
  - [Problem](#)
  - [Solution](#)
  - [Gravy](#)
  
- [Working with free time](#)
  - [Problem](#)
  - [Solution](#)

## Introduction

Cfengine is a configuration management tool that can centrally manage the configuration of UNIX servers. When discussing configuration management a common limiting thought is that it applies only to like systems. Render farms and other distributed server groups are typically configured nearly identical. It is in this environment that many administrators think to deploy a tool like Cfengine.

Even heterogeneous systems all have common configurations. A typical group of UNIX servers, regardless of their different functions, have many things in common. Package installations, NTP clients, resolv.conf, log rotation, ssh keys, sudoers files and general temp directory cleanups can all be common among different hosts. Centralizing these things makes a sysadmin more efficient.

What about centralizing an application? Consider a typical commercial LAMP web server. The server must be configured with SSL certs, Apache virtual hosts, database connectors and new content. Using Cfengine all of these things could be defined externally from the server itself.

What you end up with is a blue print for your servers and server based services that is stored in one place, has history and is backed up. This blue print can be used again and again to replace or deploy new servers and server based services with minimal repetition.

Interested? The recipes contained in this tome represent a small sample of how Cfengine can be used to make a sysadmin's work day more productive and elevate him or her from sysadmin to super sysadmin. This is very handy during review time.

## Help with Cfengine

### Problem

You know nothing about Cfengine.

### Solution

Cfengine is configuration management system that allows a sysadmin to define the ideal state of a host. As a Cfengine agent runs on the host it makes changes to ensure that the host matches the defined state.

It is not the purpose of this paper to teach the reader how to set up their own Cfengine service. To learn more about Cfengine and how to implement it please refer to the following sites:

An introduction to Cfengine

<http://www.onlamp.com/pub/a/onlamp/2004/04/15/Cfengine.html>

Cfengine website

<http://www.Cfengine.org>

Sys Admin magazine

<http://www.samag.com/documents/s=9936/sam0601e/0601e.htm>

A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine

[http://www.sage.org/pubs/16\\_cfengine/](http://www.sage.org/pubs/16_cfengine/)

## Initial setup

### Problem

You want to configure an initial setup.

### Solution

Cfengine is a client server application and needs configuration for both its client and its server. These files are called cfagent.conf and cfservd.conf respectively. The file update.conf is a special file used by the client. This file is parsed first before the client opens cfagent.conf. Using this method it is possible to create a stable, basic method to update cfagent.conf. This is desirable as complex cfagent.conf files can sometimes develop syntax errors by the sysadmin's own mistake. This can render the cfagent.conf file inoperable. By ensuring that update.conf is stable and simple, errors in the cfagent.conf file are easy to correct.

The following is an example of a client, in the network 172.16.1.0/24 is configured to communicate with one server with the IP address of 172.16.1.66. This is presented as an example but not with a full explanation. Such explanations are better left to Cfengine's official documentation.

```
#
# update.conf
#
# This file hold initial configuration information for cfengine clients.
#
# DO NOT EDIT THIS FILE UNLESS YOU REALLY KNOW WHAT YOU ARE DOING
#
# This file should hold JUST enough information to create a working
# client. This ensures that if other configuration information is in
# error this file will always work.
```

control:

```

actionsequence = ( directories copy links shellcommands tidy )

domain = ( example.com )
trustkey = ( true )

#
# Master server configuration
policyhost = ( 172.16.48.66 )

#
# Other files for distribution
masterfiles = ( /var/cfengine/masterfiles )

#
# Master configuration files
masterinputs = ( ${masterfiles}/config/inputs )

#
# Other Variables
workdir          = ( /var/cfengine )
cf_install_dir  = ( /usr/local/sbin )

#
# Ease traffic flow by deferring cleint connections by a random interval.
SplayTime = ( 5 )

```

directories:

```

    /var/cfengine/bin

```

```

#
# Now we want to distribute the update.conf and cfengine binary files to
# the client.

```

copy:

```

#
# Add cfengine .conf files.
${masterinputs}/conf/
    dest=${workdir}/inputs/
    r=inf mode=600
    type=checksum
    ignore=.svn
    server=${policyhost}

#
# Add .Cfengine import files
${masterinputs}/imports
    dest=${workdir}/inputs/
    r=inf mode=600
    type=checksum
    ignore=.svn

```

```
server=${policyhost}
```

```
#  
# Copy binaries for Redhat AS3  
redhat_as_3::  
    ${masterfiles}/config/bin/redhat_as_3  
    dest=${cf_install_dir}  
    r=inf  
    mode=755  
    ignore=.svn  
    backup=false  
    type=checksum  
    server=${policyhost}  
    # define this class to indicate that files were copied  
    define=bin_update
```

```
#  
# Copy binaries for Redhat AS4  
redhat_as_4::  
    ${masterfiles}/config/bin/redhat_as_4  
    dest=${cf_install_dir}  
    r=inf  
    mode=755  
    ignore=.svn  
    backup=false  
    type=checksum  
    server=${policyhost}  
    # define this class to indicate that files were copied  
    define=bin_update
```

```
links:  
    /var/cfengine/bin/cfagent -> /usr/local/sbin/cfagent
```

```
shellcommands:
```

```
#  
# If binaries were updated above then restart.  
bin_update::  
    "/bin/sh -c '/sbin/service cfexecd.sv restart'"  
    "/bin/sh -c '/sbin/service cfservd.sv restart'"
```

```
tidy:
```

```
#  
# Clean up old working files  
${workdir}/outputs pattern=* age=7
```

```
# EOF
```

```
#  
# cfagent.conf  
#
```

```
control:
  domain = ( example.com )

#
# Other files for distribution
masterfiles = ( /var/cfengine/masterfiles )

#
# Master server configuration
server = ( 172.16.48.66 )

workdir = ( /var/cfengine )

actionsequence = (
  directories
  resolve
  disable
  copy
  links
  editfiles
  files
  shellcommands
  processes
  packages
  tidy )

#
# For package management
DefaultPkgMgr = ( rpm )
RPMInstallCommand = ( "/usr/sbin/up2date --nox %s" )

#
# Tidy old entries from resolv.conf
EmptyResolvConf = ( true )

#
# When should the cfexecd client wake up
schedule = ( Min05_10 Min25_30 Min45_50 )

#
# Log any actions that Cfengine takes.
any::
  Syslog = ( true )
  SyslogFacility = ( LOG_DAEMON )

#
# Here we import or include separate config files
import:

any::
  # Always import classes FIRST
  classes.cf

config::
```

```
masterfiles.cf
```

```
any::
  bind.cf
  cfservd.cf
  cfexecd.cf
  cron.cf
  db2.cf
  env.cf
  httpd.cf
  iptables.cf
  mast.cf
  misc.cf
  ntp.cf
  openssh.cf
  packages.cf
  resolv.cf
  sendmail.cf
  serial.cf
  stunnel.cf
  sudo.cf
  syslog.cf
  tomcat.cf
```

```
# EOF
```

```
#
# cfservd.conf
#
```

```
# This file configures cfservd, the server daemon.
```

```
control:
```

```
domain = ( example.com )

#
# Trust keys, one time, from my network
TrustKeysFrom = ( 172.16.1.0/24 )
trustkey = ( true )

#
# These options allow remote cfrun commands
AllowUsers = ( root )
cfrunCommand = ( "/usr/local/sbin/cfagent" )
```

```
any::

#
# Traffic control to prevent clients for overloading
# The server with simultaneous requests.
IfElapsed = ( 1 )
ExpireAfter = ( 15 )
MaxConnections = ( 50 )
```

```
# This allows the server to update itself using cfrun
AllowMultipleConnectionsFrom = ( 172.16.1.66 )
```

```
grant:
```

```
#
# Grant host access for clients to access master files
/var/cfengine/masterfiles/ 172.16.1.0/24

#
# Grant access for server to allow cfrun remote command.
/usr/local/sbin/cfagent 172.16.1.66
```

```
# EOF
```

Note that the imported \*.Cfengine files are not included in this example. These will be used as the Cfengine service expands to control services in the network.

## Bootstrapping clients

### Problem

You want to bootstrap a host to be a Cfengine client.

### Solution

As with most things in the UNIX world there is more than one way to accomplish this. The manual method would involve creating the directory hierarchy under /var/cfengine. Then the cfagent, cfkey and update.conf files would be installed on the system. A key is generated using the cfkey program. Finally the cfagent program is run for the first time.

A more automated approach might involve using GNU Make. A makefile could be used to bundle a tar ball on the Cfengine server. The tar ball is then copied to the would be client and extracted. A makefile in the tar ball is used on the client to configure, install and start the agent. The example below assumes that all Cfengine binaries and configuration files are stored in a hierarchy, preferably a version control repository, as shown below.

```
./inputs/
./inputs/imports
./bin/
./bin/redhat_as_3
./bin/redhat_as_4

#
# makefile for bootstrapping Cfengine
#

# This file will build a boot strap client.

SHELL=/bin/bash
AWK=/bin/awk
```

```

SVN=/usr/local/bin/svn
PUBLICKEY=root-172.16.1.66.pub

WORKDIR=/var/cfengine
BINDIR=/usr/local/sbin
TARFILES=makefile          bin/${VERSION}/cfagent          bin/${VERSION}/
cfkey          bin/${VERSION}/cfexecd          inputs/conf/update.conf          init.
d/cfexecd.sv          ppkeys/${PUBLICKEY}          REVISION.txt

INSTALLFILES=          ${BINDIR}/cfagent          ${BINDIR}/cfkey
${BINDIR}/cfexecd          ${WORKDIR}/inputs/update.conf
${WORKDIR}/bin          /etc/rc.d/init.d/cfexecd.sv          /etc/rc.d/rc3.
d/S51cfexecd.sv

# Vpath works like PATH.  The inputs directory will be
# searched for any .conf files that are referenced.
vpath %.conf inputs
vpath %.sv init.d

.PHONY: help
help:
    make --print-data-base --question |          ${AWK} '/^[^.%][-A-Za-z0-
9]*:/'          { print substr($$1, 1, length($$1)-1) }' |          sort
# USAGE:
# make client VERSION=[redhat_as_3 | redhat_as_4]
# make install
# -- makes client tarball to be copied to new client.
#
# Copy tarball to client system.
# make install: install tarball on client.

# Make client tarball to be copied to client.
.PHONY: client

# Tarball file that will be copied to the client.
client: ${TARFILES}
    tar -cvzpf cfengine-bs-v$(shell cat REVISION.txt).tgz $^
    rm REVISION.txt

# server public key
${PUBLICKEY}:
    cp /var/cfengine/ppkeys/${PUBLICKEY} ${PUBLICKEY}

REVISION.txt:
    ${SVN} info | ${AWK} '/^Revision.*' { print $$2 }' > $@

# Install client
.PHONY: install
install: ${INSTALLFILES}
    ln -s ${BINDIR}/cfexecd ${WORKDIR}/bin/cfexecd
    ln -s ${BINDIR}/cfagent ${WORKDIR}/bin/cfagent
    ${BINDIR}/cfkey
    cp ppkeys/* ${WORKDIR}/ppkeys
    chown root:root ${WORKDIR}/ppkeys/*

```

```

chmod 600 ${WORKDIR}/ppkeys/*
service cfexecd.sv start
# Run cfagent to receive first updates.

```

```

${BINDIR}/cfagent:
cp bin/redhat_as_*/cfagent $@
chown root:root $@
chmod 755 $@

```

```

${BINDIR}/cfkey:
cp bin/redhat_as_*/cfkey $@
chown root:root $@
chmod 755 $@

```

```

${BINDIR}/cfexecd:
cp bin/redhat_as_*/cfexecd $@
chown root:root $@
chmod 755 $@

```

```

${WORKDIR}/inputs/update.conf: ${WORKDIR}/inputs
cp inputs/*_0/update.conf $@
chown root:root $@
chmod 644 $@

```

```

${WORKDIR}/ppkeys: ${WORKDIR}
mkdir ${WORKDIR}/ppkeys
chown root:root $@
chmod 700 $@

```

```

${WORKDIR}/bin: ${WORKDIR}
mkdir ${WORKDIR}/bin
chown root:root $@
chmod 700 $@

```

```

${WORKDIR}/inputs: ${WORKDIR}
mkdir ${WORKDIR}/inputs
chown root:root $@
chmod 700 $@

```

```

${WORKDIR}:
mkdir $@
chown root:root $@
chmod 755 $@

```

```

/etc/rc.d/init.d/cfexecd.sv: ${BINDIR}/cfexecd
cp init.d/cfexecd.sv /etc/rc.d/init.d
chmod 755 $@

```

```

/etc/rc.d/rc3.d/S51cfexecd.sv: /etc/rc.d/init.d/cfexecd.sv
ln -s $< $@

```

```

# EOF

```

Using this makefile a client tar ball is created with ``make client". The tar ball is copied to the client host and extracted. Inside is a makefile. The command ``make install" will install and start the client.

# Gravy

Add Cfengine to your automated installs, like kickstart, by having the installer download and install the tar ball.

## Version Control

### Problem

How can I combine Cfengine with version control to keep history and have an audit trail available?

### Solution

In recipe [3](#) we noted that the Cfengine master files were stored in `/var/cfengine/masterfiles`. It is this location that Cfengine has access to revision controlled files. In this example we'll use Subversion. Subversion is a revision control system similar to CVS. In this instance one or more Subversion repositories can be hosted locally on the Cfengine master host or on a remote host. The method of file retrieval is the same. In this example Cfengine checks the most recent copy of a repository into a preexisting working copy. Cfengine considers this working copy to be its set of master files.

```
#
# masterfiles.cf
#
# This ensures that the masterfiles are current on policy servers

copy:

# Update copy of masterfiles on dmz from internal server.
dmz_mirror::
    ${masterfiles}/config
    dest=${masterfiles}/config
    r=inf mode=600
    type=checksum
    # Skip SVN control files.
    ignore=.svn
    # Skip internal only files.
    ignore=internal
    ignore=172_16_*
    # Skips docs
    ignore=docs
    # Do not keep old files
    purge=true
    backup=false
    server=${server}

shellcommands:

# Update cfengine repositories
config_server::
    "/bin/sh -c 'cd ${masterfiles}/config; /usr/local/bin/svn update'"
```

This example performs two tasks. The first is to copy selective master files from the internal master server to a mirror (see recipe [6](#)). The second task is a shell command that updates the current working copy of Cfengine's master files from the Cfengine master files Subversion repository. Note that during your initial setup of this working copy a manual check out will need to be performed for the first time. Afterward this update should work automatically.

## Gravy

Cfengine can use this method to distribute other files such as DNS files. This can allow your hostmaster access to his DNS repository but still have Cfengine distribute the changes. See recipe [12.1](#) for more details.

## DMZ mirror

### Problem

Cfengine client hosts query a master server for updates. This ``pull'' method of communication can break firewall policies.

### Solution

Cfengine is often described as using a ``pull'' method for client server communication. In other words the client initiates a connection with the master server. Since the master server is typically located on an internal network and some clients can be located on DMZ network security policies can be broken. The way to minimize this security exception is to create a DMZ mirror.

A DMZ mirror is a Cfengine server located on the DMZ network. This server pulls selected master files from the internal master server and in turn acts as the master server for DMZ clients.

Consider this addition to the masterfiles.cf file demonstrated in recipe [5](#).

```
copy:

# Update copy of masterfiles on dmz from internal server.
dmz_mirror::
    ${masterfiles}/config
    dest=${masterfiles}/config
    r=inf mode=600
    type=checksum
    # Skip SVN control files.
    ignore=.svn
    # Skip internal only files.
    ignore=internal
    ignore=172_16_*
    # Skips docs
    ignore=docs
    # Do not keep old files
    purge=true
    backup=false
```

```
server=${server}
```

In this copy action master files are copied or downloaded to the DMZ mirror server. Files that are relevant to internal hosts only or sensitive documentation are skipped using the ignore clause. Now clients located on the DMZ network can query the DMZ server for relevant changes without allowing them access to the internal network. The DMZ mirror server still needs access to the internal network to contact the master server but this is now limited to a single host that can be hardened.

## Organizing Cfengine files

### Problem

As a Cfengine configuration grows how should it be organized to keep it maintainable?

### Solution

A good way to organize a Cfengine configuration files is by service. When Cfengine is configured to manage a service it is often convenient to keep the configuration in a separate file that is imported into the cfagent.conf file. For example, consider a sudoers file.

In the cfagent.conf file import the sudoers related actions.

```
import:
  any::
  sudoers.cf
```

The sudoers file would look like this.

```
#
# sudo.cf
#
copy:
  ${masterfiles}/config/sudo/sudoers
    dest=/etc/sudoers
    server=${server}
    owner=root
    group=root
    encrypt=true
    mode=440
    type=checksum
```

This instructs the client to copy the sudoers file from the Cfengine server to the destination on the client. Because the sudoers file contains sensitive information the copy operation is encrypted. Since the class is not listed before the copy Cfengine defaults to the any class. Thus the sudoers file is maintained on all Cfengine clients.

Now we see that the sudoers service is organized into two parts. The first is the configuration file sudoers.cf that is imported into the cfagent.conf file. The second is the sudoers files which is listed in the master files hierarchy. Expanding on this could lead to cfagent.conf imports like this:

```
import:
```

```
any::
  bind.cf
  cron.cf
  httpd.cf
  iptables.cf
  misc.cf
  ntp.cf
  openssh.cf
  resolv.cf
  sendmail.cf
  serial.cf
  stunnel.cf
  sudo.cf
  syslog.cf
  tomcat.cf
```

The master files hierarchy on the Cfengine server might look like this:

```
./bin
./cron
./httpd
./init.d
./inputs
./iptables
./misc
./ntp
./openssh
./ppkeys
./stunnel
./sudo
./syslog
./tomcat
```

## Custom classes

### Problem

How do I group certain hosts into a single class for an action?

### Solution

Note that I mentioned a `classes.cf` file in recipe [7](#). This file is used to configure custom classes.

```
classes:

  any::

    # DB2 servers
    db2 = ( db2srv1 db2srv2 db2test db2mirror )
```

```
# DNS slaves
dnsslave = ( ipv4_10_0_0_1 ipv4_10_0_0_2
             ipv4_10_0_0_3 ipv4_10_0_0_4 )

# group of servers to patch
update01 = ( dnsslave db2 )
```

The class ``db2" includes four hostnames. The class ``dnsslave" includes four IP addresses. Note that the underscore is used instead of a period in the addresses. Cfengine interprets the period as a logical 'and' in classes. Thus `dnsslave.Hr20` is interpreted by Cfengine to mean any host of the `dnsslave` class and of the hour twenty ( eight PM ) class.

## Package management

### Problem

You want to ensure that certain packages are installed in some or all of your hosts.

### Solution

Cfengine is able to manage packages for multiple UNIX distributions including Solaris, Red Hat and Debian. In this example we'll configure Cfengine for use with Red Hat hosts. First, Cfengine needs to know what package manager to use by default. This is placed in the control section of the `cfagent.conf` file:

```
DefaultPkgMgr = ( rpm )
RPMInstallCommand = ( "/usr/sbin/up2date --nox %s" )
```

You may want to have a `packages.cf` file that is imported by `cfagent.conf`. In that file we configure which packages are managed and how.

```
packages:
```

```
any::
  screen action=install
  sysstat action=install
  vim-enhanced action=install
  rpm-devel action=install
  rpm-build action=install
  compat-db action=install
  ntp action=install
```

The above action ensures that the named RPMs are installed if they are not present. Note that versions are not important in this example. However, it is possible to instruct Cfengine to maintain the packages based the package version.

You could have a `packages` section in any other service related import file. For example, if you had an `ntp.cf` file to handle network time configuration you could include a `packages` section to ensure that any needed NTP packages were installed.

# Gravy

Being a lazy sysadmin I don't like to manually update basic, non-masked RPMs on certain Red Hat hosts that I manage. I configure Cfengine to perform these updates automatically for me.

```
update01.Monday.Hr20::
  "/usr/sbin/urpdate --nox -u"
  ifelapsed=60
  expireafter=60
```

The class ``update01" is a class that contains an arbitrary group of hosts that I would like to update. The compound class is interpreted as any host that is a member of the ``update01" class and is a member of the day class ``Monday" and is a member of the time class ``Hr20". This means that all ``update01" host will be updated, using `urpdate`, on Mondays at 2000 hours. The ``ifelapsed" statement tells Cfengine to run this command only if at least sixty minutes has elapsed since the last time it was run. This ensures that `urpdate` will not run multiple times during the eight o'clock hour even if Cfengine runs more than once per hour. The ``expireafter" statement tells Cfengine to kill this job if it takes more than sixty minutes to complete. This is good housekeeping and helps guard against any run away jobs.

## Distributing public keys

### Problem

You want to distribute your public SSH key to all hosts.

### Solution

This is a relatively simple task using Cfengine's files and directories actions.

directories:

```
# Create .ssh directorise
# Neil Watson
/home/nhwatson/.ssh
  mode=700
  owner=nhwatson
  group=nhwatson
```

copy:

```
# Copy authorized_keys file.
# Neil Watson
${masterfiles}/config/openssh/nhwatson/authorized_keys
  dest=/home/nhwatson/.ssh/authorized_keys
  server=${server}
  mode=644
  owner=nhwatson
  group=nhwatson
  type=checksum
```

In this configuration we first ensure that the user's `.ssh` directory is created and is set to the correct mode of `0700`. In the copy action we copy a stored public key to the remote host and ensure that the mode is set to `0644`.

## Gravy

This method could be expanded to include an entire user environment. Files like `.profile`, `.vimrc` and even whole directories such as `.vim` could all be managed and distributed using this method.

## CRON tables

### Problem

Have you ever had someone ask you to change a production cron table entry but, that someone cannot describe what host the cron table on is or for what user? To avoid this you'd like to centralize all cron tables to Cfengine.

### Solution

If your crons are very convoluted and dependant upon one another you may want to consider using an enterprise scheduler. However, Cfengine can still be helpful in organizing cron tables.

Cfengine can work as a scheduler without using cron at all. Consider this example.

```
#
# crons.cf
#
classes:
    firstweekday = ( (Day1|Day2|Day3).!(Saturday|Sunday) )
import:

    # Jobs that run at 0000 hours
    Hr00:: dailyjobs.cf

    Hr05::
        # Nightly ELT jobs.
        etl.cf
        # Nightly reports
        reports.cf

    # Jobs that run every Monday
    Monday:: mondayjobs.cf

    # Jobs that run the first weekday of the month
    firstweekday:: firstweekday.cf
```

Organizing in this way ensures that Cfengine did not have to import and parse any files that are not due. For example, on Tuesdays the `mondayjobs.cf` file would not be imported.

The imported file will contain the information Cfengine needs to determine what job to run and on which client. For example:

```
#
# dailyjobs.cf
#
actionsequence = ( copy shellcommands )

# Ensure that script is up to date
copy:
  webapps01::
    ${masterfiles}/config/jobs/webapps01/clean.sh
    dest=/home/secacct/bin/clean.sh
    mode=700
    backup=false
    owner=sevacct
    group=secacct
    type=checksum

# Run up to date script
shellcommands:
  tor_lx_webapps01::
    "/home/devacct/bin/clean.sh"
    owner=sevacct
    group=sevacct
    ifelapsed=60
```

The ``ifelapsed=60" clause ensures that the shell command is only executed once per hour even if cfagent is run more than once. Using this method Cfengine controls both when the script is run and the contents of the script .

One caveat to this set up is reliability. If cron jobs are for production work then the Cfengine server may need to be redundant or even highly available in order to ensure that these jobs are run consistently. If a Cfengine cluster is not available it is suggested that Cfengine be used to distribute cron table files instead. This still offers a central copy of all jobs but allows the highly available production servers to run them autonomously. For example:

```
# cron.cf
  tor_lx_svn::
    # cron tables.
    ${masterfiles}/config/cron/internal/tor_lx_svn
    dest=/var/spool/cron/
    server=${server}
    r=inf
    ignore=.svn
    owner=root
    group=root
    encrypt=true
    mode=600
    type=checksum

    # script for root's crontab
    ${masterfiles}/infotech/subversion/svn_backup.sh
    dest=/usr/local/bin/svn_backup.sh
```

```

server=${server}
owner=root
group=root
mode=755
type=checksum

# script for hostmaster's crontab
${masterfiles}/infotech/dns/whois.pl
dest=/usr/local/bin/whois.pl
server=${server}
owner=root
group=root
mode=755
type=checksum

```

In above example the cron tables, typically found in `/var/spool/cron` are copied from the master files location to the target host. Additionally, certain cron entries use custom scripts. These scripts are also copied by Cfengine from its master files location.

Regardless of which method is used, finding a specific cron entry is now simplified. Recursive text searches are now possible.

## DNS services

### Problem

You want use Cfengine to manage your DNS services.

### Solution

Since the Bind DNS daemon uses normal text files Cfengine is a natural choice to manage these files. Additionally since Cfengine using version control all changes to Bind files are recorded for audits and roll backs.

Expanding on recipe [5](#) we can add a new repository strictly for Bind files and configure Cfengine to keep its working copy current. Using a separate repository allows a separation of duties amongst the sysadmin team. Now certain team members will be able to manage DNS services without needing access to Cfengine's own configuration.

```

copy:

# Update copy of masterfiles on dmz from internal server.
dmz_mirror::

# Update copy of config masterfiles on dmz from internal server.
${masterfiles}/bind
dest=${masterfiles}/bind
r=inf mode=600
type=checksum
# Skip SVN control files.
ignore=.svn

```

```

# Skip bcp files
ignore=bcp
# Skip internal only files.
ignore=internal
ignore=172_16_*
# Skips docs
ignore=docs
# Do not keep old files
purge=true
backup=false
server=${server}

```

shellcommands:

```

# Update cfengine repositories
config_server::
    "/bin/sh -c 'cd ${masterfiles}/bind; /usr/local/bin/svn update'"

```

Now we create Cfengine rules for managing Bind configurations.

```

#
# dns.cf
#
#####
# External bind services.

```

copy:

```

#####
# named.conf files
ext_dns_master::
    ${masterfiles}/bind/trunk/dmz/master/etc/named.conf
    dest=/etc/named.conf
    server=${server}
    owner=root
    group=named
    mode=640
    type=checksum
    define=dns_reload

```

```

ext_dns_slave::
    ${masterfiles}/bind/trunk/dmz/slave/etc/named.conf
    dest=/etc/named.conf
    server=${server}
    owner=root
    group=named
    mode=640
    type=checksum
    define=dns_reload

```

```

#####
# record files
ext_dns_master::
    ${masterfiles}/bind/trunk/dmz/master/var/named/data

```

```
dest=/var/named/data
server=${server}
recurse=inf
include=*.hosts
include=*.ptr
include=*.rev
include=db.*
owner=named
group=named
mode=640
type=checksum
define=dns_reload
```

```
ns2::
```

```
# loopback
```

```
  ${masterfiles}/bind/trunk/dmz/slave/var/named/slaves/db.127.0.0.ns2
    dest=/var/named/slaves/db.127.0.0
    server=${server}
    owner=root
    group=named
    mode=640
    type=checksum
    define=dns_reload
```

```
ns3::
```

```
# loopback
```

```
  ${masterfiles}/bind/trunk/dmz/slave/var/named/slaves/db.127.0.0.ns3
    dest=/var/named/slaves/db.127.0.0
    server=${server}
    owner=root
    group=named
    mode=640
    type=checksum
    define=dns_reload
```

```
ext_dns_master::
```

```
# root hint file
```

```
  ${masterfiles}/bind/trunk/dmz/db.cache
    dest=/var/named/data/db.cache
    server=${server}
    owner=root
    group=named
    mode=640
    type=checksum
    define=dns_reload
```

```
ext_dns_slave::
```

```
# root hint file
```

```
  ${masterfiles}/bind/trunk/dmz/db.cache
    dest=/var/named/slaves/db.cache
    server=${server}
    owner=root
    group=named
    mode=640
```

```

type=checksum
define=dns_reload

```

```
shellcommands:
```

```

dns_reload::
    "/bin/sh -c '/sbin/service named reload'"

```

In this configuration there is one master Bind service and two slaves. All three are controlled by Cfengine which will ensure that the files are current and that Bind reloads the files when they change.

## Process control

### Problem

You want to ensure that a service is running on a host.

### Solution

The processes action allows Cfengine to test for running processes, start them or send any number of signals (e. g. kill, HUP, term). For example, suppose you wish for Cfengine to monitor a web server and ensure that its web service is always running.

```
processes:
```

```

webserver::

    # ensure apache is running
    "/usr/sbin/httpd"
    matches=>4
    signal=kill
    restart "/sbin/service httpd restart"

```

The above example runs on any host that is a member of the ``webserver" class. Cfengine examines the list of running processes and looks for the regular expression ``/usr/sbin/httpd". If Cfengine finds four or less running processes then they are killed and the restart command is run.

The above processes example shows how to ensure a process is running. Suppose you want to ensure that a process is not running? Suppose a group of developers show ``initiative" and install a telnet service on their development hosts. Telnet is strictly against our security policy. Educating the developers is a good step but Cfengine can help ensure that ``accidents" don't happen.

```
processes:
```

```

development::

    # ensure no telnet services are running
    "telnet" match=<1 signal=kill

```

Above we target any ``development" class hosts. Here Cfengine will attempt to match one or more processes that match the ``telnet" regular expression and kill any that it finds.

## Editing files

### Problem

You need to edit some files in place without replacing them.

### Solution

Sometimes a configuration file provided by the package install is close enough to what you need that only a few minor changes are required. Having Cfengine manage the entire file is a little inefficient in such cases. Fortunately Cfengine is able to edit text files for you. This can allow you to change just a few part of the file. Consider this collection of examples:

```
editfiles:
```

```
any::
    # Prevent ctrl-alt-del from rebooting the system
    { /etc/inittab
        CommentLinesMatching "ca::ctrlaltdel:/sbin/shutdown -t3 -r now"
    }
```

In the tangle of KVM wires have you ever though that you had issued a ctrl-alt-delete to Windows host only to realize with horror that it was a Linux host and that host is now rebooting? This editfiles clause ensures that our hosts do not respond to a ctrl-alt-delete. It does so by commenting, that is inserting a ``#" character in front, in the /etc/inittab file any line matching what is found in quotations.

```
editfiles:
```

```
any::
    # Rotated logs should be compressed
    { /etc/logrotate.conf
        ReplaceFirst "^[[:space:]]*#[[:space:]]*compress[[:space:]]*$"
            With "compress"
    }
```

Many default log rotation configurations do not compress old log files. If space is at a premium compression is a good way to go. This clause uses a POSIX regular expression to replace the first instance of the commented compress line with an uncommented compress line in the /etc/logrotate.conf file.

```
editfiles:
```

```
any.!x86_64::
    # Run sysstat every 5 minutes
    # Note that these rules are different for 32bit versus 64 bit
    # hosts.
```

```

{ /etc/cron.d/sysstat
  ReplaceFirst "^*/10.*sa1.*" With "*/5 * * * * root /usr/lib/sa/sa1 1 1"
  Backup "false"
}

```

In this example the sysstat cron job is changed so that the sysstat data collector runs every five minutes instead of every ten. Note that the string to search for uses a regular expression but the replacement string does not. Also note that the class string omits 64bit hosts.

editfiles:

```

x86_64::
# Run sysstat every 5 minutes
# Note that these rules are different for 32bit versus 64 bit
# hosts.
{ /etc/cron.d/sysstat
  ReplaceFirst "^*/10.*sa1.*" With "*/5 * * * * root /usr/lib64/sa/sa1
1 1"
  Backup "false"
}

```

This example is the almost the same as the previous one but in this case it is for 64bit hosts.

From these four examples we can see that Cfengine has the ability to comment lines, perform a search and replace and use regular expressions. Cfengine's ability to edit files is extensive and often complex. Studying the official Cfengine manual is recommended.

## Editing complex host specific files

### Problem

You have file (e.g. snmpd.conf, sudoers) that is common for many hosts but with specific differences for many others.

### Solution

A simple solution would be to maintain a separate configuration file for each host. If much of the file is same for each host this becomes inefficient when changes are made. The goal of edit once, deploy everywhere would be lost. A better option would be to maintain a common template file. This file could be copied to target hosts and then edited for each class of host. Additionally, some services allow ``included'' import files. Those could be maintained for specific hosts and copied where needed.

Using the copy action a template file could be centrally maintained and distributed.

```

# snmp.cf
copy:

snmp::
  ${masterfiles}/config/snmp/snmpd-temp.conf
  dest=/etc/snmp/snmpd-temp.conf
  server=${server}

```

```

owner=root
group=root
encrypt=true
type=checksum
mode=400

```

Here we copy a template `snmpd.conf` file, called `snmpd-temp.conf`. Also note that we encrypt this copy operation since the `snmpd.conf` file contains sensitive community strings.

Now we perform edits based on specific classes. Here we will use another feature of Cfengine's editfiles action. This feature is called `expandvariables`.

control:

```

# Define load for normal hosts
snmp.!highload::
    load = ( load 2 2 2 )

# Define load for high load hosts
highload::
    load = ( load 5 4 4 )

```

editfiles:

```

# Create SNMP file from template file
snmp::
{
    /etc/snmp/snmpd.conf
    EmptyEntireFilePlease
    AutoCreate
    Backup "false"
    InsertFile "/etc/snmp/snmpd-temp.conf"
    ExpandVariables "true"
    DefineClasses "snmpdrestart"
}

```

files:

```

/etc/snmp/snmpd.conf mode=0600 owner=root group=root

```

shellcommands:

```

snmpdrestart::
    "/bin/sh -c '/sbin/service snmpd restart'"

```

The example shows four distinct parts. The first part, `control`, defines Cfengine variables `load` to a value based on class. In this case we want the `snmpd.conf` file to have different load numbers to consider if the host is of the `highload` class versus any other SNMP class of host.

If you know a little about the Net-SNMP configuration file you know that there is a `load` stanza that looks like the variable names we've defined. However, in our template file we've listed this part as a Cfengine style variable called `load`. You'll see why shortly.

The next part is the `editfiles` action. In this part we identify the file, empty it to start fresh, we ensure that Cfengine will create the file even if it doesn't exist and we do not backup the old file. Next we insert the template file into the empty file that was just created. Now here is the magic. `ExpandVariables "true"`

tells Cfengine to read the file we've just created. When Cfengine finds any Cfengine style variables (e.g. `load`) it expands them to their current values. In this example we defined the current value of the `load` variable in the first step to be either `load 2 2` or `load 5 4`. Lastly we define a new class called `snmpdrestart` each time our target file (`snmpd.conf`) is edited.

Next is a files action. The `editfiles` action will result in a file with a mode of 644. An `snmpd.conf` file, which contains community strings and passwords should be less visible. Thus we use the files action to set the mode to 600 as well as set the owner and group.

The final part is a `shellcommands` action. This action restarts the `snmpd` service if the `snmpdrestart` class is defined. In reality this means that when the `snmpd.conf` file is changed the `snmpd` daemon is restarted.

Consider another example for monitoring disk space.

```
control:
  # Define disk space
  snmp.oracle::
    diskoptoracle = ( "disk /opt/oracle 200000" )

editfiles:
  # Create SNMP file from template file
  snmp::
    {
      /etc/snmp/snmpd.conf
      EmptyEntireFilePlease
      AutoCreate
      Backup "false"
      InsertFile "/etc/snmp/snmpd-skel.conf"
      ExpandVariables "true"
      DefineClasses "snmpdrestart"
    }

files:
  /etc/snmp/snmpd.conf mode=0600 owner=root group=root

shellcommands:
  snmpdrestart::
    "/bin/sh -c '/sbin/service snmpd restart'"
```

Here we configure the SNMP daemon to monitor the available disk space of an Oracle partition. Note that the rest of this example is the same as the first. Thus in practice only the variables in the control section and perhaps your class definitions elsewhere need to be altered. The `ExpandVariables` action will work with any variables you have defined.

One last thing to consider is economy. Suppose you have Cfengine scheduled to run several times per hour. It might not be efficient to have Cfengine create this `snmpd.conf` file and restart the daemon for each run. If you like you could instruct Cfengine, using a time class, to perform this edit once an hour or even less.

```
editfiles:

  # Create SNMP file from template file
  snmp.Min15_20::
    {
      /etc/snmp/snmpd.conf
```

```

    EmptyEntireFilePlease
    AutoCreate
    Backup "false"
    IfElapsed 60
    InsertFile "/etc/snmp/snmpd-skel.conf"
    ExpandVariables "true"
    DefineClasses "snmpdrestart"
}

```

Here we've added a time class of ``15\_20" minutes ensure that this is run only at 15 to 20 minutes after the hour. Also the ``IfElapsed 60" line tells Cfengine not to perform this operation if it has already been performed less than 60 minutes ago.

## Application deployment

### Problem

In recipe [9](#) we looked at how to deploy RPMs using Cfengine. Suppose you have a custom application, like a web site, that needs to be deployed and maintained in production and possibly development and QA settings.

### Solution

Many web or middleware applications are simply a collection of files that are copied from one server to another as they move from development to QA and to production. J2EE applications should use WAR or EAR files for deployments. Applications that are built in different languages may not have any type of deployment tools. In such cases, Cfengine's copy action is a good method for ensuring that applications are kept current on the target servers.

Suppose there is a web application that consists of a collection of HTML, PDF, CGI and Apache configuration files. This application goes through the standard development, QA and production cycles. Cfengine could be configured to copy these files to each of the separate development, QA and production servers. The magic here is in which master files Cfengine uses. A typical version control repository will consist of a trunk and various branches. The trunk is typically considered production and the branches can consist of development, QA and perhaps personal developer branches.

copy:

```

#####
# Production web services
#####
web01::
    ${masterfiles}/mywebapp/trunk/www
    dest=/var/www/mywebapp
    server=${server}
    r=inf
    ignore=.svn
    owner=root
    group=root
    mode=644
    type=checksum

```

```

backup=false
purge=true

```

```

${masterfiles}/mywebapp/trunk/etc/httpd/conf
dest=/etc/httpd/conf
server=${server}
r=inf
ignore=.svn
purge=true
backup=false
owner=root
group=root
mode=640
type=checksum
# This will be used for shellcommand below.
define=httpd_reload

```

```

${masterfiles}/mywebapp/trunk/etc/httpd/conf.d
dest=/etc/httpd/conf.d
server=${server}
r=inf
ignore=.svn
purge=true
backup=false
owner=root
group=root
mode=640
type=checksum
# This will be used for shellcommand below.
define=httpd_reload

```

```
#####
```

```
# development web services
```

```
#####
```

```
devweb01::
```

```

${masterfiles}/mywebapp/branches/dev/www
dest=/var/www/mywebapp
server=${server}
r=inf
ignore=.svn
owner=root
group=root
mode=644
type=checksum
backup=false
purge=true

```

```

${masterfiles}/mywebapp/branches/dev/etc/httpd/conf
dest=/etc/httpd/conf
server=${server}
r=inf
ignore=.svn
purge=true
backup=false

```

```

owner=root
group=root
mode=640
type=checksum
# This will be used for shellcommand below.
define=httpd_reload

```

```

${masterfiles}/mywebapp/branches/dev/etc/httpd/conf.d
dest=/etc/httpd/conf.d
server=${server}
r=inf
ignore=.svn
purge=true
backup=false
owner=root
group=root
mode=640
type=checksum
# This will be used for shellcommand below.
define=httpd_reload

```

```
#####
```

```
# QA web services
```

```
#####
```

```
qaweb01::
```

```

${masterfiles}/mywebapp/branches/qa/www
dest=/var/www/mywebapp
server=${server}
r=inf
ignore=.svn
owner=root
group=root
mode=644
type=checksum
backup=false
purge=true

```

```

${masterfiles}/mywebapp/branches/qa/etc/httpd/conf
dest=/etc/httpd/conf
server=${server}
r=inf
ignore=.svn
purge=true
backup=false
owner=root
group=root
mode=640
type=checksum
# This will be used for shellcommand below.
define=httpd_reload

```

```

${masterfiles}/mywebapp/branches/qa/etc/httpd/conf.d
dest=/etc/httpd/conf.d
server=${server}

```

```

r=inf
ignore=.svn
purge=true
backup=false
owner=root
group=root
mode=640
type=checksum
# This will be used for shellcommand below.
define=httpd_reload

```

shellcommands:

```

# Restat httpd services if configuration is changed
httpd_reload::
    "/bin/sh -c '/sbin/service httpd reload'"

```

In this example production, development and QA Apache configurations and content are maintained and copied as needed. Notice that Apache is reloaded automatically when needed. Notice also that legacy files will be purged at the target location. This ensures that only the files that we want there will reside there. Promotion from development to QA or QA to production is done by simply merging branches in the repository and waiting for Cfengine to notice the changes and update them on the target hosts. Additionally revision control and Cfengine logging will leave an audit trail of who promoted code and when.

## Gravy

Managing Apache SSL keys can be tedious. If you store your key files in a repository like the master files in the above example, Cfengine can distribute them as needed.

# Pushing changes immediately

## Problem

You've made emergency changes and need Cfengine to deploy them now.

## Solution

Typically Cfengine is a gradual convergent service. Over scheduled runs Cfengine will bring a host up to date. Sometimes you need Cfengine to fix a problem immediately. In such cases there are two options. The first is to log onto the target host and run the cfagent program manually. This is done as root. The length of time it takes to complete this operation can vary wildly depending on what has to be done and how busy the master server is so be patient. You may even have to run cfagent more than once if there are action dependencies.

Since Cfengine is a system where the clients pull from the server the client cannot be instructed remotely from the server. However, it is possible to run a Cfengine server process on each target client. The master server can then use Cfengine's cfrun command to contact the client's server process and instruct it to run cfagent.

In recipe [3](#) we actually enabled our setup to use cfrun already. In the cfservd.conf file of that recipe there are four lines that reference the cfrun command:

..

```
# These options allow remote cfrun commands
AllowUsers = ( root )
cfrunCommand = ( "/usr/local/sbin/cfagent" )
```

..

```
# This allows the server to update itself using cfrun
AllowMultipleConnectionsFrom = ( 172.16.1.66 )
```

..

```
#
# Grant access for server to allow cfrun remote command.
/usr/local/sbin/cfagent      172.16.1.66
```

This file controls the server component of Cfengine called cfservd. Our last step is to define a file called the cfrun.hosts

```
domain=example.com
```

```
# Hosts that cfrun is allowed to connect to.
# KEEP THIS UP TO DATE.
```

```
172.16.1.140
172.16.1.172
172.16.1.177
172.16.1.203
172.16.1.204
172.16.1.215
172.16.1.239
172.16.1.35
172.16.1.64
172.16.48.66
192.168.81.43
```

This file tells the server which hosts it should connect to when a cfrun command is issued. If a host is not in this file cfrun will not make a connection attempt. Once this is in place it should be simple to force a client to contact the server by running the command

```
cfrun 172.16.1.35
```

from the master server where 172.16.1.35 is the IP address of the remote client. Multiple IP addresses could also be listed. If no IP addresses are listed then cfrun will contact all hosts listed in the cfrun.hosts file. Since this activates the cfagent process on the remote client it will take a variable amount of time to run. For more information on cfrun please consult the official Cfengine documentation.

## Running manual commands on all clients

### Problem

Occasionally you wish to run a manual command on all clients. This command is run at your whim with no set schedule. How do you avoid logging on to each host manually?

## Solution

The traditional solution may be to create a custom SSH script that logs on to each machine sequentially and executes the command. Why go through that hassle when Cfengine can already do this for you? Suppose that as part of your disaster recovery plan requires that you be able to shut down all of your hosts in short order. During a power outage you may have only a few minutes to perform this shutdown before the backup batteries run dry. In such a situation a manual operation will take to long.

The first step is to define a shell command within the cfagent.conf file.

```
shellcommands:
    emergency_shutdown::
        "/sbin/halt"
```

Now we have a special class called emergency\_shutdown. What hosts are a member of that class? None until you define it. Recall in recipe [17](#) that we used cfrun to affect a ``push" action. We will do so again here but we will also define a class for all hosts at the same time.

```
cfrun -D emergency_shutdown
```

The above command instructs all remote clients, listed in the cfrun.hosts file, to run immediately. Additionally all clients will be members of the class emergency\_shutdown. The result is that all clients run the halt command.

## Redundancy and load balancing

### Problem

Your Cfengine deployment has scaled to many hosts. You would like to add more master hosts for redundancy and load balancing.

### Solution

Cfengine does not currently possess much in the way of redundancy or load balancing but, there are ways to achieve at least partial results. Strategies is a way for Cfengine to choose randomly from a group of classes. Each class in the group can be assigned a weight in the random selection process.

```
strategies:
    { random_policy_host
        policyhost1: "1"
        policyhost2: "1"
    }

control:
    policyhost1::
        policyhost = ( 172.16.48.66 )

    policyhost2::
        policyhost = ( 172.16.48.67 )
```

In this example the strategies section defines to classes, ``policyhost1" and ``policyhost2". Each class is defined

a weight of one. In the control section each of the random classes defines a variable called ``policyhost''. Cfengine will randomly choose to define the server variable to 172.16.48.66 or 172.16.48.67. Since each class is weighted as one there is a 50 percent chance for each assignment. In previous recipes Cfengine code referred to a `{policyhost}` variable. In the recipe [3](#) ``policyhost'' was defined as a single IP address. Using the strategies method ``policyhost'' is defined randomly each time cfagent runs. Thus during each run about half of the Cfengine clients should contact each policy host.

If one of your policy hosts should go down this example will ensure that the client will contact the running host half of the time. Your frequency of updates is about half of what it was before but, at least updates are still available. You could add more hosts to allow better redundancy and load balancing. You could also alter the weight of the random classes to account for more powerful or more reliable policy hosts.

Another redundancy option to consider is that of the client's cfagent process. This is typically controlled by the daemon cfexecd which will execute the client according to the schedule defined in the agent configuration. Suppose the cfexecd daemon dies? Running cfexecd from a cron job will ensure even if the daemon dies the agent will still be run. Additionally, the agent itself can be configured to check for the cfexecd daemon process and start it if required. Using this method you'll be ensured that both the running daemon and the cron job will check up on each other.

## Caveat

Since the client and policy host authenticate via key exchange be sure that the clients have the keys of all policy hosts and that the policy hosts each have all of the client keys.

# Security policies

## Problem

You need to maintain a tedious and constantly changing security policy across some or all of your hosts.

## Solution

Many aspects of a security policy define how a service or host should be configured. A typical security policy may touch upon PAM files, log file permissions, log history retention, home directory permissions and SSH configuration to name a few. Cfengine can be used to maintain all of these examples and more. In using Cfengine you are assured that hosts meet the current policy requirements. New hosts will have the policy automatically applied. Policy changes need only be defined in Cfengine which will then apply them automatically to all current and future hosts.

# Audits

## Problem

A security auditor wants to know how you track host configuration changes. Suppose a new configuration was deployed but failed. You want to determine when the deployment happened and what exactly was changed.

## Solution

All configurations are stored in Cfengine's master files location. The master files are actually a working copy of a Subversion, or other revision control repository. It is beyond the scope of this paper to explain the details of version control. However, with version control one can easily determine every line that has been changed, when that change was checked in and by whom.

Additionally, using Cfengine's Syslog facility (see section [3](#)) Cfengine will log to a host's syslog service any changes it makes or errors it encounters. Thus not only will you know how and when a file was changed you'll also know when Cfengine deployed it.

## Disaster recovery

### Problem

You need to recover a host which has had a catastrophic failure.

### Solution

The good news is that if this host is managed by Cfengine then your job may already be done. Imagine that all the previous recipes have been used to manage this failed host. Imagine that you have expanded these recipes to encompass other configurations that you wish to manage. With the failure of the host only a fresh install is an option. After the OS is installed and the network configured, boot strap Cfengine onto the host. Since this new host is replacing an old host its Cfengine classes will not change. Now Cfengine will automatically restore all the configurations, settings and services that it was managing on the failed host.

### Gravy

Since your configurations are all stored in a repository and copied to target hosts using Cfengine there may be no need to backup target hosts. As long as the revision control repositories are backed up you should be safe. Any file lost on a target host that exist in the repository will not only be available for restoration but will even be restored automatically by Cfengine.

In some cases a hot spare host will be setup at a disaster recovery site. This host is configured to mirror a production host and take over in the event of failure. Often replicating changes to the off site host is done via SAN replication or proprietary replication services. Cfengine can do this with little effort. Since the production host is managed by Cfengine it is easy to add the spare host by a compound class or by defining your own classes. See the example below.

copy:

```
#####
# named.conf files for production AND disaster recovery hosts
ext_dns_master|ext_dns_master_dr::
    ${masterfiles}/bind/trunk/dmz/master/etc/named.conf
    dest=/etc/named.conf
    server=${server}
    owner=root
    group=named
```

```
mode=640
type=checksum
define=dns_reload
```

# Working with free time

## Problem

Now that Cfengine is managing your host's you find yourself with much more free time.

## Solution

Is this really a problem? I could suggest taking a vacation or working part time but a more practical suggestion would be to talk to your boss. If you have free time you or your boss will no doubt have a list of "nice to have" projects that have been in waiting. Having automated the day to day tasks you can become a contractor within your own company. Start those new projects. Offer your services to other departments for their projects. Make your boss, your department and you a more valuable asset to the organization. Be sure to remind your boss of this at review time.

## About this document ...

### Cfengine Cookbook

This document was generated using the [LaTeX2HTML](#) translator Version 2002-2-1 (1.71)

Copyright © 1993, 1994, 1995, 1996, Nikos Drakos, Computer Based Learning Unit, University of Leeds.  
Copyright © 1997, 1998, 1999, [Ross Moore](#), Mathematics Department, Macquarie University, Sydney.

The command line arguments were:

```
latex2html -html_version 4.0 -no_subdir -no_navigation -split 0 cfcookbook.tex
```

The translation was initiated by Neil Watson on 2008-10-17

---

Neil Watson 2008-10-17

---

@>/

[Contact](#)