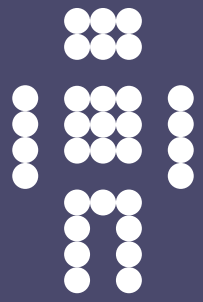
CFEngine



Package Management

A CFEngine Special Topics Handbook

CFEngine AS

CFEngine interfaces with operating system package management systems to offer best-effort convergent maintenance of software packages. Package management can be subtle, due to the diverse behaviours of different package managers.

Table of Contents

What is package management?

Package management is about managing software inventory. It includes ensuring that software is installed on computers, and in the correct versions. It includes patching and upgrading. Each operating system generally has its own approved package manager and software source. Usually this is supplied by the operating system provider.

Some package managers allow users to create their own software packages providing a uniform way of deploying software to systems. Packaging software is common on GNU/Linux, where well-known package formats include RPM and DPGK.

Strengths and weaknesses of package management

Packages were introduced to bring a rational approach to handling software dependencies. By dividing up applications and libraries into packages, one can share code efficiently and assign the responsibility of updating and versioning to different maintainers.

Package management is not a substitute for configuration management. It only delivers preconfigured files into a specific location. Packaged software cannot be customized to local needs without post-installation adaptation.

What does CFEngine bring to package management?

CFEngine does not try to fight against package mangers, but rather work with them. CFEngine integrates the idea of convergent maintenance with package installation, so that one can be certain of maintaining a desired state.

Package managers do not usually have the intelligence to be able to verify the actual state of software configuration. Rather they assume that once a package is installed, it will remain in a good state until an update is required.

If one reinstalls a package, changes get blown away in favour of the original matrix. Package installation is thus a 'destructive' installation mechanism. It overwrites whatever currently exists with a prefabricated (and therefore approximate) version of what you need. For generic software this is exactly what is required. However, for complex software such as web services, this is entirely insufficient to result in a working system.

CFEngine brings convergent methods to package management, and allows surgically precise customizations to be applied and maintained even after multiple package upgrades.

Package promises

To manage software, you write `packages` promises, analogous to any other kind of promise in CFEngine. It makes sense to use lists to install packages if you don't need to make complex specifications about versions. Keep it simple and package management will be a simple matter.

```
vars:
 "match_package" slist => {
                          "apache2",
                          "apache2-mod_php5",
                          "apache2-prefork",
                          "php5"
                          };
packages:

   "$(match_package)"
        package_policy => "add",
        package_method => yum;
```

Many users elect to install a basic 'stem cell' image for all machines in their environment, and then customize each machine to a specific purpose by adding or subtracting packages from this stem cell starting state. CFEngine can be used together with other tools like `Cobbler` or `rPath` to accomplish this in a comfortable way in your environment. If you are working in the cloud, this is the default approach to management. You begin from a basic image and then customize it by either hardening or extending the software inventory.

How CFEngine compares package versions

Cfengine uses a model for packages that is generic enough to support all the known package managers. It classifies packages into

Name The name of the packet is usually the name of the software itself, e.g. '`cfengine`'.

Version Versions of a particular piece of software are described in wildly different ways, causing a lot of confusion. For instance, a common model is to use major version number, minor version number and patch release number, e.g. 3.1.5. However, many maintainers slap on their own additions, such as 3.1.5-2 or 3.1.5-2.el5. Because these models are operating system, software and release specific, you have to know the versioning numbers used on your operating systems and refer to them properly. CFEngine cannot reliably guess these things for you.

Architecture The architecture describes the hardware platform for execution, e.g. '`x86_64`' or '`i586`'. This is important when package managers store multiple architectures in the same repository.

Example package promises

Let's look at some example cases to explain the behaviour of the interaction between CFEngine and the package managers.

Install latest package version example

Suppose there is a older version of `wget` installed on your machine.

```
redhat$ rpm -q wget
wget-1.10.2-7.el5
```

Now suppose you'd like to upgrade the package to the latest version available in a repository by using yum. We make a promise such the following;

```
bundle agent test001
{
 packages:
  redhat::
   "wget"
             package_policy => "addupdate",
             package_method => yum,
             package_select => ">=",
            package_version => "1.11.4-2.el5_4.1",
      package_architectures => { "x86_64" };

}
```

Now run this bundle:

```
redhat$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

redhat$ rpm -q wget
wget-1.11.4-2.el5_4.1
```

If there is no wget installed, CFE will install the lastest one for you.

```
redhat$ rpm -e wget
redhat$ rpm -q wget
package wget is not installed
redhat$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K
redhat$ rpm -q wget
wget-1.11.4-2.el5_4.1
```

Install specific package version example

To install a specific version, we can just adapt the promise. This example will use RPM as the YUM repository doesn't support multi-version packages.

```
bundle agent test002
{
 packages:
  redhat::
   "wget"
             package_policy => "addupdate",
             package_method => rpm("/root"),
```

```
             package_select => "==",
            package_version => "1.10.2-7.el5",
      package_architectures => { "x86_64" };

}
```

Now see before and after:

```
redhat$ ls -l /root
-rw-r--r-- 1 root root 595422 Apr  4  2007 wget-1.10.2-7.el5.x86_64.rpm
-rw-r--r-- 1 root root 596335 Nov  5  2009 wget-1.11.4-2.el5_4.1.x86_64.rpm

redhat$ rpm -q wget
package wget is not installed

redhat$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

redhat$ rpm -q wget
wget-1.10.2-7.el5
```

To upgrade the package to a newer version, just change '`package_version`' to a version you'd like;

```
bundle agent test003
{
 packages:
  redhat::
   "wget"
             package_policy => "addupdate",
             package_method => rpm("/root"),
             package_select => "==",
            package_version => "1.11.4-2.el5_4.1",
      package_architectures => { "x86_64" };

}
```

Now see the result:

```
redhat$ rpm -q wget
wget-1.10.2-7.el5

redhat$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

redhat$ rpm -q wget
wget-1.11.4-2.el5_4.1
```

Here is an example for Ubuntu, which supports both the APT and DPKG interfaces.

```
bundle agent test004
{
 packages:
  ubuntu::
   "wget"
             package_policy => "addupdate",
             package_method => apt,
             package_select => ">=",
            package_version => "1.12-1.1ubuntu2.1",
      package_architectures => { "*" };
}
```

Before and after:

```
ubuntu$ dpkg -l | grep wget
ii  wget  1.10.2-3ubuntu1.2 retrieves files from the web

ubuntu$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

ubuntu$ dpkg -l | grep wget
ii  wget  1.12-1.1ubuntu2.1 retrieves files from the web
```

Similarly, we can use the "dpkg" interface to install specific version of the software.

```
bundle agent test005
{
 packages:
  ubuntu::
   "wget"
             package_policy => "addupdate",
             package_method => dpkg("/root"),
             package_select => "==",
            package_version => "1.10.2-3ubuntu1.2",
      package_architectures => { "*" };
}
```

Before and after:

```
ubuntu$ dpkg -l | grep wget

ubuntu$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

ubuntu$ dpkg -l | grep wget
ii  wget                          1.10.2-3ubuntu1.2                retrieves files from
```

Uprading to a newer package version example

To upgrade to a newer version of apackage, we simply assign a newer version to package_version and change the policy to include updating.

```
bundle agent test006
{
 packages:
  ubuntu::
   "wget"
             package_policy => "addupdate",
             package_method => dpkg("/root"),
             package_select => "==",
            package_version => "1.12-1.1ubuntu2.1",
      package_architectures => { "*" };
}
```

Before and after the keeping of this promise:

```
ubuntu$ dpkg -l | grep wget
ii  wget                             1.10.2-3ubuntu1.2                retrieves files from
ubuntu$ cf-agent -f /tmp/test.cf -K
ubuntu$ dpkg -l | grep wget
ii  wget                             1.12-1.1ubuntu2.1                retrieves files from
```

Here is an example using the zypper package manager:

```
bundle agent test007
{
 packages:
  SuSE::
   "tcpdump"
             package_policy => "addupdate",
             package_method => zypper,
             package_select => ">=",
            package_version => "4.1.1-1.11",
      package_architectures => { "x86_64" };
}
```

Before and after running the agent:

```
suse$ rpm -q tcpdump
tcpdump-4.0.0-2.1.x86_64

suse$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

suse$ rpm -q tcpdump
tcpdump-4.1.1-1.11.x86_64
```

Finally, since SuSE uses RPM as a native format so we can use '`package_method rpm()`' from above.

```
bundle agent test008
{
 packages:
  SuSE::
   "tcpdump"
             package_policy => "addupdate",
             package_method => rpm("/root"),
             package_select => "==",
            package_version => "4.0.0-2.1",
      package_architectures => { "x86_64" };
}
```

```
suse$ ls -l /root
-rw-r--r-- 1 root root 571158 2009-10-19 20:36 tcpdump-4.0.0-2.1.x86_64.rpm
-rw-r--r-- 1 root root 318279 2010-07-05 23:37 tcpdump-4.1.1-1.11.x86_64.rpm

suse$ rpm -q tcpdump
package tcpdump is not installed

suse$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

suse$ rpm -q tcpdump
tcpdump-4.0.0-2.1.x86_64
```

Changing to a new version:

```
bundle agent test009
{
 packages:
  SuSE::
   "tcpdump"
             package_policy => "addupdate",
             package_method => rpm("/root"),
             package_select => "==",
            package_version => "4.1.1-1.11",
      package_architectures => { "x86_64" };
```

```
}
```

Before and after:

```
suse$ rpm -q tcpdump
tcpdump-4.0.0-2.1.x86_64

suse$ /var/cfengine/bin/cf-agent -f /tmp/test.cf -K

suse$ rpm -q tcpdump
tcpdump-4.1.1-1.11.x86_64
```

Package management next steps

The CFEngine standard library contains package manager methods for all major operating systems and managers. Check out the reference documentation too to learn about extended features of package integration. Visit also the community forum to hear about reach experiences.