**Common Examples¶**

The examples in this section show functionality that is common across all resources.

**Use the :nothing action**

service 'memcached' **do**

action :nothing

**end**

**Use the ignore\_failure common attribute**

gem\_package 'syntax' **do**

action :install

ignore\_failure true

**end**

**Use the retries common attribute**

service 'apache' **do**

action [ :enable, :start ]

retries 3

**end**

**Create a file, but not if an attribute has a specific value**

The following example shows how to use the not\_if condition to create a file based on a template and using the presence of an attribute value on the node to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if { node['some\_value'] }

**end**

**Create a file with a Ruby block, but not if “/etc/passwd” exists**

The following example shows how to use the not\_if condition to create a file based on a template and then Ruby code to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if **do**

File.exist?('/etc/passwd')

**end**

**end**

**Create a file with Ruby block that has curly braces, but not if “/etc/passwd” exists**

The following example shows how to use the not\_if condition to create a file based on a template and using a Ruby block (with curly braces) to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if { File.exist?('/etc/passwd') }

**end**

**Create a file using a string, but not if “/etc/passwd” exists**

The following example shows how to use the not\_if condition to create a file based on a template and using a string to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if 'test -f /etc/passwd'

**end**

**Install a file from a remote location using bash**

The following is an example of how to install the foo123 module for Nginx. This module adds shell-style functionality to an Nginx configuration file and does the following:

* Declares three variables
* Gets the Nginx file from a remote location
* Installs the file using Bash to the path specified by the src\_filepath variable

*# the following code sample is similar to the ``upload\_progress\_module``*

*# recipe in the ``nginx`` cookbook:*

*# https://github.com/chef-cookbooks/nginx*

src\_filename = "foo123-nginx-module-v**#{**

node['nginx']['foo123']['version']

**}**.tar.gz"

src\_filepath = "**#{**Chef::Config['file\_cache\_path']**}**/**#{**src\_filename**}**"

extract\_path = "**#{**

Chef::Config['file\_cache\_path']

**}**/nginx\_foo123\_module/**#{**

node['nginx']['foo123']['checksum']

**}**"

remote\_file 'src\_filepath' **do**

source node['nginx']['foo123']['url']

checksum node['nginx']['foo123']['checksum']

owner 'root'

group 'root'

mode '0755'

**end**

bash 'extract\_module' **do**

cwd ::File.dirname(src\_filepath)

code <<-EOH

mkdir -p #{extract\_path}

tar xzf #{src\_filename} -C #{extract\_path}

mv #{extract\_path}/\*/\* #{extract\_path}/

EOH

not\_if { ::File.exist?(extract\_path) }

**end**

**Create a file, but only if an attribute has a specific value**

The following example shows how to use the only\_if condition to create a file based on a template and using the presence of an attribute on the node to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

only\_if { node['some\_value'] }

**end**

**Create a file with a Ruby block, but only if “/etc/passwd” does not exist**

The following example shows how to use the only\_if condition to create a file based on a template, and then use Ruby to specify a condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

only\_if **do** ! File.exist?('/etc/passwd') **end**

**end**

**Create a file using a string, but only if “/etc/passwd” exists**

The following example shows how to use the only\_if condition to create a file based on a template and using a string to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

only\_if 'test -f /etc/passwd'

**end**

**Delay notifications**

template '/etc/nagios3/configures-nagios.conf' **do**

*# other parameters*

notifies :run, 'execute[test-nagios-config]', :delayed

**end**

**Notify immediately**

By default, notifications are :delayed, that is they are queued up as they are triggered, and then executed at the very end of a chef-client run. To run an action immediately, use :immediately:

template '/etc/nagios3/configures-nagios.conf' **do**

*# other parameters*

notifies :run, 'execute[test-nagios-config]', :immediately

**end**

and then the chef-client would immediately run the following:

execute 'test-nagios-config' **do**

command 'nagios3 --verify-config'

action :nothing

**end**

**Enable a service after a restart or reload**

service 'apache' **do**

supports :restart => true, :reload => true

action :enable

**end**

**Notify multiple resources**

template '/etc/chef/server.rb' **do**

source 'server.rb.erb'

owner 'root'

group 'root'

mode '0755'

notifies :restart, 'service[chef-solr]', :delayed

notifies :restart, 'service[chef-solr-indexer]', :delayed

notifies :restart, 'service[chef-server]', :delayed

**end**

**Notify in a specific order**

To notify multiple resources, and then have these resources run in a certain order, do something like the following:

execute 'foo' **do**

command '...'

notifies :create, 'template[baz]', :immediately

notifies :install, 'package[bar]', :immediately

notifies :run, 'execute[final]', :immediately

**end**

template 'baz' **do**

...

notifies :run, 'execute[restart\_baz]', :immediately

**end**

package 'bar'

execute 'restart\_baz'

execute 'final' **do**

command '...'

**end**

where the sequencing will be in the same order as the resources are listed in the recipe: execute 'foo', template 'baz', execute [restart\_baz], package 'bar', and execute 'final'.

**Reload a service**

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

notifies :reload, 'service[apache]', :immediately

**end**

**Restart a service when a template is modified**

template '/etc/www/configures-apache.conf' **do**

notifies :restart, 'service[apache]', :immediately

**end**

**Send notifications to multiple resources**

To send notifications to multiple resources, just use multiple attributes. Multiple attributes will get sent to the notified resources in the order specified.

template '/etc/netatalk/netatalk.conf' **do**

notifies :restart, 'service[afpd]', :immediately

notifies :restart, 'service[cnid]', :immediately

**end**

service 'afpd'

service 'cnid'

**Execute a command using a template**

The following example shows how to set up IPv4 packet forwarding using the **execute** resource to run a command named forward\_ipv4 that uses a template defined by the **template** resource:

execute 'forward\_ipv4' **do**

command 'echo > /proc/.../ipv4/ip\_forward'

action :nothing

**end**

template '/etc/file\_name.conf' **do**

source 'routing/file\_name.conf.erb'

notifies :run, 'execute[forward\_ipv4]', :delayed

**end**

where the command property for the **execute** resource contains the command that is to be run and the sourceproperty for the **template** resource specifies which template to use. The notifies property for the **template**specifies that the execute[forward\_ipv4] (which is defined by the **execute** resource) should be queued up and run at the end of the chef-client run.

**Restart a service, and then notify a different service**

The following example shows how start a service named example\_service and immediately notify the Nginx service to restart.

service 'example\_service' **do**

action :start

notifies :restart, 'service[nginx]', :immediately

**end**

**Restart one service before restarting another**

This example uses the :before notification to restart the php-fpm service before restarting nginx:

service `nginx` **do**

action :restart

notifies :restart, `service[php-fpm]`, :before

**end**

With the :before notification, the action specified for the nginx resource will not run until action has been taken on the notified resource (php-fpm).

**Notify when a remote source changes**

remote\_file '/tmp/couch.png' **do**

source 'http://couchdb.apache.org/img/sketch.png'

action :nothing

**end**

http\_request 'HEAD http://couchdb.apache.org/img/sketch.png' **do**

message ''

url 'http://couchdb.apache.org/img/sketch.png'

action :head

**if** File.exist?('/tmp/couch.png')

headers 'If-Modified-Since' => File.mtime('/tmp/couch.png').httpdate

**end**

notifies :create, 'remote\_file[/tmp/couch.png]', :immediately

**end**

**Prevent restart and reconfigure if configuration is broken**

Use the :nothing action (common to all resources) to prevent the test from starting automatically, and then use the subscribes notification to run a configuration test when a change to the template is detected:

execute 'test-nagios-config' **do**

command 'nagios3 --verify-config'

action :nothing

subscribes :run, 'template[/etc/nagios3/configures-nagios.conf]', :immediately

**end**

**Reload a service using a template**

To reload a service that is based on a template, use the **template** and **service** resources together in the same recipe, similar to the following:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

**end**

service 'apache' **do**

action :enable

subscribes :reload, 'template[/tmp/somefile]', :immediately

**end**

where the subscribes notification is used to reload the service whenever the template is modified.

**Stash a file in a data bag**

The following example shows how to use the **ruby\_block** resource to stash a BitTorrent file in a data bag so that it can be distributed to nodes in the organization.

*# the following code sample comes from the ``seed`` recipe*

*# in the following cookbook: https://github.com/mattray/bittorrent-cookbook*

ruby\_block 'share the torrent file' **do**

block **do**

f = File.open(node['bittorrent']['torrent'],'rb')

*#read the .torrent file and base64 encode it*

enc = Base64.encode64(f.read)

data = {

'id'=>bittorrent\_item\_id(node['bittorrent']['file']),

'seed'=>node.ipaddress,

'torrent'=>enc

}

item = Chef::DataBagItem.new

item.data\_bag('bittorrent')

item.raw\_data = data

item.save

**end**

action :nothing

subscribes :create, "bittorrent\_torrent[**#{**node['bittorrent']['torrent']**}**]", :immediately

**end**

**Relative Paths**

template "**#{**ENV['HOME']**}**/chef-getting-started.txt" **do**

source 'chef-getting-started.txt.erb'

mode '0755'

**end**

**apt\_package¶**

Use the **apt\_package** resource to manage packages on Debian and Ubuntu platforms.

**Install a package using package manager**

apt\_package 'name of package' **do**

action :install

**end**

**Install a package using local file**

apt\_package 'jwhois' **do**

action :install

source '/path/to/jwhois.deb'

**end**

**Install without using recommend packages as a dependency**

package 'apache2' **do**

options '--no-install-recommends'

**end**

**apt\_update¶**

Use the **apt\_update** resource to manage APT repository updates on Debian and Ubuntu platforms.

**Update the Apt repository at a specified interval**

apt\_update 'all platforms' **do**

frequency 86400

action :periodic

**end**

**Update the Apt repository at the start of a chef-client run**

apt\_update 'update'

**bash¶**

Use the **bash** resource to execute scripts using the Bash interpreter. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence.

**Note**

The **bash** script resource (which is based on the **script** resource) is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

**Use a named provider to run a script**

bash 'install\_something' **do**

user 'root'

cwd '/tmp'

code <<-EOH

wget http://www.example.com/tarball.tar.gz

tar -zxf tarball.tar.gz

cd tarball

./configure

make

make install

EOH

**end**

**Install a file from a remote location using bash**

The following is an example of how to install the foo123 module for Nginx. This module adds shell-style functionality to an Nginx configuration file and does the following:

* Declares three variables
* Gets the Nginx file from a remote location
* Installs the file using Bash to the path specified by the src\_filepath variable

*# the following code sample is similar to the ``upload\_progress\_module``*

*# recipe in the ``nginx`` cookbook:*

*# https://github.com/chef-cookbooks/nginx*

src\_filename = "foo123-nginx-module-v**#{**

node['nginx']['foo123']['version']

**}**.tar.gz"

src\_filepath = "**#{**Chef::Config['file\_cache\_path']**}**/**#{**src\_filename**}**"

extract\_path = "**#{**

Chef::Config['file\_cache\_path']

**}**/nginx\_foo123\_module/**#{**

node['nginx']['foo123']['checksum']

**}**"

remote\_file 'src\_filepath' **do**

source node['nginx']['foo123']['url']

checksum node['nginx']['foo123']['checksum']

owner 'root'

group 'root'

mode '0755'

**end**

bash 'extract\_module' **do**

cwd ::File.dirname(src\_filepath)

code <<-EOH

mkdir -p #{extract\_path}

tar xzf #{src\_filename} -C #{extract\_path}

mv #{extract\_path}/\*/\* #{extract\_path}/

EOH

not\_if { ::File.exist?(extract\_path) }

**end**

**Install an application from git using bash**

The following example shows how Bash can be used to install a plug-in for rbenv named ruby-build, which is located in git version source control. First, the application is synchronized, and then Bash changes its working directory to the location in which ruby-build is located, and then runs a command.

git "**#{**Chef::Config[:file\_cache\_path]**}**/ruby-build" **do**

repository 'git://github.com/sstephenson/ruby-build.git'

reference 'master'

action :sync

**end**

bash 'install\_ruby\_build' **do**

cwd '#{Chef::Config[:file\_cache\_path]}/ruby-build'

user 'rbenv'

group 'rbenv'

code <<-EOH

./install.sh

EOH

environment 'PREFIX' => '/usr/local'

**end**

To read more about ruby-build, see here: https://github.com/sstephenson/ruby-build.

**Store certain settings**

The following recipe shows how an attributes file can be used to store certain settings. An attributes file is located in the attributes/ directory in the same cookbook as the recipe which calls the attributes file. In this example, the attributes file specifies certain settings for Python that are then used across all nodes against which this recipe will run.

Python packages have versions, installation directories, URLs, and checksum files. An attributes file that exists to support this type of recipe would include settings like the following:

default['python']['version'] = '2.7.1'

**if** python['install\_method'] == 'package'

default['python']['prefix\_dir'] = '/usr'

**else**

default['python']['prefix\_dir'] = '/usr/local'

**end**

default['python']['url'] = 'http://www.python.org/ftp/python'

default['python']['checksum'] = '80e387...85fd61'

and then the methods in the recipe may refer to these values. A recipe that is used to install Python will need to do the following:

* Identify each package to be installed (implied in this example, not shown)
* Define variables for the package version and the install\_path
* Get the package from a remote location, but only if the package does not already exist on the target system
* Use the **bash** resource to install the package on the node, but only when the package is not already installed

*# the following code sample comes from the ``oc-nginx`` cookbook on |github|: https://github.com/cookbooks/oc-nginx*

version = node['python']['version']

install\_path = "**#{**node['python']['prefix\_dir']**}**/lib/python**#{**version.split(/(^\d+\.\d+)/)[1]**}**"

remote\_file "**#{**Chef::Config[:file\_cache\_path]**}**/Python-**#{**version**}**.tar.bz2" **do**

source "**#{**node['python']['url']**}**/**#{**version**}**/Python-**#{**version**}**.tar.bz2"

checksum node['python']['checksum']

mode '0755'

not\_if { ::File.exist?(install\_path) }

**end**

bash 'build-and-install-python' **do**

cwd Chef::Config[:file\_cache\_path]

code <<-EOF

tar -jxvf Python-#{version}.tar.bz2

(cd Python-#{version} && ./configure #{configure\_options})

(cd Python-#{version} && make && make install)

EOF

not\_if { ::File.exist?(install\_path) }

**end**

**batch¶**

Use the **batch** resource to execute a batch script using the cmd.exe interpreter on Windows. The **batch** resource creates and executes a temporary file (similar to how the **script** resource behaves), rather than running the command inline. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence.

**Unzip a file, and then move it**

To run a batch file that unzips and then moves Ruby, do something like:

batch 'unzip\_and\_move\_ruby' **do**

code <<-EOH

7z.exe x #{Chef::Config[:file\_cache\_path]}/ruby-1.8.7-p352-i386-mingw32.7z

-oC:\\source -r -y

xcopy C:\\source\\ruby-1.8.7-p352-i386-mingw32 C:\\ruby /e /y

EOH

**end**

batch 'echo some env vars' **do**

code <<-EOH

echo %TEMP%

echo %SYSTEMDRIVE%

echo %PATH%

echo %WINDIR%

EOH

**end**

or:

batch 'unzip\_and\_move\_ruby' **do**

code <<-EOH

7z.exe x #{Chef::Config[:file\_cache\_path]}/ruby-1.8.7-p352-i386-mingw32.7z

-oC:\\source -r -y

xcopy C:\\source\\ruby-1.8.7-p352-i386-mingw32 C:\\ruby /e /y

EOH

**end**

batch 'echo some env vars' **do**

code 'echo %TEMP%\\necho %SYSTEMDRIVE%\\necho %PATH%\\necho %WINDIR%'

**end**

**bff\_package¶**

Use the **bff\_package** resource to manage packages for the AIX platform using the installp utility. When a package is installed from a local file, it must be added to the node using the **remote\_file** or **cookbook\_file** resources.

**Note**

A Backup File Format (BFF) package may not have a .bff file extension. The chef-client will still identify the correct provider to use based on the platform, regardless of the file extension.

New in Chef Client 12.0.

**Install a package**

The **bff\_package** resource is the default package provider on the AIX platform. The base **package** resource may be used, and then when the platform is AIX, the chef-client will identify the correct package provider. The following examples show how to install part of the IBM XL C/C++ compiler.

Using the base **package** resource:

package 'xlccmp.13.1.0' **do**

source '/var/tmp/IBM\_XL\_C\_13.1.0/usr/sys/inst.images/xlccmp.13.1.0'

action :install

**end**

Using the **bff\_package** resource:

bff\_package 'xlccmp.13.1.0' **do**

source '/var/tmp/IBM\_XL\_C\_13.1.0/usr/sys/inst.images/xlccmp.13.1.0'

action :install

**end**

**breakpoint¶**

Use the **breakpoint** resource to add breakpoints to recipes. Run the chef-shell in chef-client mode, and then use those breakpoints to debug recipes. Breakpoints are ignored by the chef-client during an actual chef-client run. That said, breakpoints are typically used to debug recipes only when running them in a non-production environment, after which they are removed from those recipes before the parent cookbook is uploaded to the Chef server.

**A recipe without a breakpoint**

yum\_key node['yum']['elrepo']['key'] **do**

url node['yum']['elrepo']['key\_url']

action :add

**end**

yum\_repository 'elrepo' **do**

description 'ELRepo.org Community Enterprise Linux Extras Repository'

key node['yum']['elrepo']['key']

mirrorlist node['yum']['elrepo']['url']

includepkgs node['yum']['elrepo']['includepkgs']

exclude node['yum']['elrepo']['exclude']

action :create

**end**

**The same recipe with breakpoints**

breakpoint "before yum\_key node['yum']['repo\_name']['key']" **do**

action :break

**end**

yum\_key node['yum']['repo\_name']['key'] **do**

url node['yum']['repo\_name']['key\_url']

action :add

**end**

breakpoint "after yum\_key node['yum']['repo\_name']['key']" **do**

action :break

**end**

breakpoint "before yum\_repository 'repo\_name'" **do**

action :break

**end**

yum\_repository 'repo\_name' **do**

description 'description'

key node['yum']['repo\_name']['key']

mirrorlist node['yum']['repo\_name']['url']

includepkgs node['yum']['repo\_name']['includepkgs']

exclude node['yum']['repo\_name']['exclude']

action :create

**end**

breakpoint "after yum\_repository 'repo\_name'" **do**

action :break

**end**

where the name of each breakpoint is an arbitrary string. In the previous examples, the names are used to indicate if the breakpoint is before or after a resource, and then also to specify which resource.

**chef\_gem¶**

Use the **chef\_gem** resource to install a gem only for the instance of Ruby that is dedicated to the chef-client. When a gem is installed from a local file, it must be added to the node using the **remote\_file** or **cookbook\_file**resources.

The **chef\_gem** resource works with all of the same properties and options as the **gem\_package** resource, but does not accept the gem\_binary property because it always uses the CurrentGemEnvironment under which the chef-client is running. In addition to performing actions similar to the **gem\_package** resource, the **chef\_gem**resource does the following:

* Runs its actions immediately, before convergence, allowing a gem to be used in a recipe immediately after it is installed
* Runs Gem.clear\_paths after the action, ensuring that gem is aware of changes so that it can be required immediately after it is installed

**Install a gems file for use in recipes**

To install a gem while the chef-client is configuring the node (the “converge phase”), set the compile\_timeproperty to false:

chef\_gem 'right\_aws' **do**

compile\_time false

action :install

**end**

To install a gem while the resource collection is being built (the “compile phase”), set the compile\_time property to true:

chef\_gem 'right\_aws' **do**

compile\_time true

action :install

**end**

New in Chef Client 12.1.

**Install MySQL for Chef**

apt\_update

node.override['build\_essential']['compiletime'] = true

include\_recipe 'build-essential'

include\_recipe 'mysql::client'

node['mysql']['client']['packages'].each **do** |mysql\_pack|

resources("package[**#{**mysql\_pack**}**]").run\_action(:install)

**end**

chef\_gem 'mysql'

**chef\_handler¶**

Use the **chef\_handler** resource to enable handlers during a chef-client run. The resource allows arguments to be passed to the chef-client, which then applies the conditions defined by the custom handler to the node attribute data collected during the chef-client run, and then processes the handler based on that data.

The **chef\_handler** resource is typically defined early in a node’s run-list (often being the first item). This ensures that all of the handlers will be available for the entire chef-client run.

**New in Chef Client 14.0.**

**Enable the CloudkickHandler handler**

The following example shows how to enable the CloudkickHandler handler, which adds it to the default handler path and passes the oauth key/secret to the handler’s initializer:

chef\_handler "CloudkickHandler" **do**

source "**#{**node['chef\_handler']['handler\_path']**}**/cloudkick\_handler.rb"

arguments [node['cloudkick']['oauth\_key'], node['cloudkick']['oauth\_secret']]

action :enable

**end**

**Enable handlers during the compile phase**

chef\_handler "Chef::Handler::JsonFile" **do**

source "chef/handler/json\_file"

arguments :path => '/var/chef/reports'

action :nothing

**end**.run\_action(:enable)

**Handle only exceptions**

chef\_handler "Chef::Handler::JsonFile" **do**

source "chef/handler/json\_file"

arguments :path => '/var/chef/reports'

supports :exception => true

action :enable

**end**

**Cookbook Versions (a custom handler)**

Community member juliandunn created a custom report handler that logs all of the cookbooks and cookbook versions that were used during the chef-client run, and then reports after the run is complete. This handler requires the **chef\_handler** resource (which is available from the **chef\_handler** cookbook).

cookbook\_versions.rb:

The following custom handler defines how cookbooks and cookbook versions that are used during the chef-client run will be compiled into a report using the Chef::Log class in the chef-client:

require 'chef/log'

**module** **Opscode**

**class** CookbookVersionsHandler < Chef::Handler

**def** report

cookbooks = run\_context.cookbook\_collection

Chef::Log.info('Cookbooks and versions run: #{cookbooks.keys.map {|x| cookbooks[x].name.to\_s + ' ' + cookbooks[x].version} }')

**end**

**end**

**end**

default.rb:

The following recipe is added to the run-list for every node on which a list of cookbooks and versions will be generated as report output after every chef-client run.

include\_recipe 'chef\_handler'

cookbook\_file "**#{**node['chef\_handler']['handler\_path']**}**/cookbook\_versions.rb" **do**

source 'cookbook\_versions.rb'

owner 'root'

group 'root'

mode '0755'

action :create

**end**

chef\_handler 'Opscode::CookbookVersionsHandler' **do**

source "**#{**node['chef\_handler']['handler\_path']**}**/cookbook\_versions.rb"

supports :report => true

action :enable

**end**

This recipe will generate report output similar to the following:

[2013-11-26T03:11:06+00:00] INFO: Chef Run complete **in** 0.300029878 seconds

[2013-11-26T03:11:06+00:00] INFO: Running report handlers

[2013-11-26T03:11:06+00:00] INFO: Cookbooks **and** versions run: ["chef\_handler 1.1.4", "cookbook\_versions\_handler 1.0.0"]

[2013-11-26T03:11:06+00:00] INFO: Report handlers complete

**JsonFile Handler**

The json\_file handler is available from the **chef\_handler** cookbook and can be used with exceptions and reports. It serializes run status data to a JSON file. This handler may be enabled in one of the following ways.

By adding the following lines of Ruby code to either the client.rb file or the solo.rb file, depending on how the chef-client is being run:

require 'chef/handler/json\_file'

report\_handlers << Chef::Handler::JsonFile.new(:path => '/var/chef/reports')

exception\_handlers << Chef::Handler::JsonFile.new(:path => '/var/chef/reports')

By using the **chef\_handler** resource in a recipe, similar to the following:

chef\_handler 'Chef::Handler::JsonFile' **do**

source 'chef/handler/json\_file'

arguments :path => '/var/chef/reports'

action :enable

**end**

After it has run, the run status data can be loaded and inspected via Interactive Ruby (IRb):

irb(main):001:0> require 'rubygems' => true

irb(main):002:0> require 'json' => true

irb(main):003:0> require 'chef' => true

irb(main):004:0> r = JSON.parse(IO.read('/var/chef/reports/chef-run-report-20110322060731.json')) => ... output truncated

irb(main):005:0> r.keys => ['end\_time', 'node', 'updated\_resources', 'exception', 'all\_resources', 'success', 'elapsed\_time', 'start\_time', 'backtrace']

irb(main):006:0> r['elapsed\_time'] => 0.00246

**Register the JsonFile handler**

chef\_handler "Chef::Handler::JsonFile" **do**

source "chef/handler/json\_file"

arguments :path => '/var/chef/reports'

action :enable

**end**

**ErrorReport Handler**

The error\_report handler is built into the chef-client and can be used for both exceptions and reports. It serializes error report data to a JSON file. This handler may be enabled in one of the following ways.

By adding the following lines of Ruby code to either the client.rb file or the solo.rb file, depending on how the chef-client is being run:

require 'chef/handler/error\_report'

report\_handlers << Chef::Handler::ErrorReport.new()

exception\_handlers << Chef::Handler::ErrorReport.new()

By using the chef\_handler resource in a recipe, similar to the following:

chef\_handler 'Chef::Handler::ErrorReport' **do**

source 'chef/handler/error\_report'

action :enable

**end**

**chocolatey\_package¶**

Use the **chocolatey\_package** resource to manage packages using Chocolatey on the Microsoft Windows platform.

**Install a package**

chocolatey\_package 'name of package' **do**

action :install

**end**

**Install a package with options**

This example uses Chocolatey’s --checksum option:

chocolatey\_package 'name of package' **do**

options '--checksum 1234567890'

action :install

**end**

**cookbook\_file¶**

Use the **cookbook\_file** resource to transfer files from a sub-directory of COOKBOOK\_NAME/files/ to a specified path located on a host that is running the chef-client. The file is selected according to file specificity, which allows different source files to be used based on the hostname, host platform (operating system, distro, or as appropriate), or platform version. Files that are located in the COOKBOOK\_NAME/files/default sub-directory may be used on any platform.

**Transfer a file**

cookbook\_file 'file.txt' **do**

mode '0755'

**end**

**Handle cookbook\_file and yum\_package resources in the same recipe**

When a **cookbook\_file** resource and a **package** resource are both called from within the same recipe, use the flush\_cache attribute to dump the in-memory Yum cache, and then use the repository immediately to ensure that the correct package is installed:

cookbook\_file '/etc/yum.repos.d/custom.repo' **do**

source 'custom'

mode '0755'

**end**

package 'only-in-custom-repo' **do**

action :install

flush\_cache [ :before ]

**end**

**Install repositories from a file, trigger a command, and force the internal cache to reload**

The following example shows how to install new Yum repositories from a file, where the installation of the repository triggers a creation of the Yum cache that forces the internal cache for the chef-client to reload:

execute 'create-yum-cache' **do**

command 'yum -q makecache'

action :nothing

**end**

ruby\_block 'reload-internal-yum-cache' **do**

block **do**

Chef::Provider::Package::Yum::YumCache.instance.reload

**end**

action :nothing

**end**

cookbook\_file '/etc/yum.repos.d/custom.repo' **do**

source 'custom'

mode '0755'

notifies :run, 'execute[create-yum-cache]', :immediately

notifies :create, 'ruby\_block[reload-internal-yum-cache]', :immediately

**end**

**Use a case statement**

The following example shows how a case statement can be used to handle a situation where an application needs to be installed on multiple platforms, but where the install directories are different paths, depending on the platform:

cookbook\_file 'application.pm' **do**

path **case** node['platform']

**when** 'centos','redhat'

'/usr/lib/version/1.2.3/dir/application.pm'

**when** 'arch'

'/usr/share/version/core\_version/dir/application.pm'

**else**

'/etc/version/dir/application.pm'

**end**

source "application-**#{**node['languages']['perl']['version']**}**.pm"

owner 'root'

group 'root'

mode '0755'

**end**

**Manage dotfiles**

The following example shows using the **directory** and **cookbook\_file** resources to manage dotfiles. The dotfiles are defined by a JSON data structure similar to:

"files": {

".zshrc": {

"mode": '0755',

"source": "dot-zshrc"

},

".bashrc": {

"mode": '0755',

"source": "dot-bashrc"

},

".bash\_profile": {

"mode": '0755',

"source": "dot-bash\_profile"

},

}

and then the following resources manage the dotfiles:

**if** u.has\_key?('files')

u['files'].each **do** |filename, file\_data|

directory "**#{**home\_dir**}**/**#{**File.dirname(filename)**}**" **do**

recursive true

mode '0755'

**end** **if** file\_data['subdir']

cookbook\_file "**#{**home\_dir**}**/**#{**filename**}**" **do**

source "**#{**u['id']**}**/**#{**file\_data['source']**}**"

owner 'u['id']'

group 'group\_id'

mode 'file\_data['mode']'

ignore\_failure true

backup 0

**end**

**end**

**cron¶**

Use the **cron** resource to manage cron entries for time-based job scheduling. Properties for a schedule will default to \* if not provided. The **cron** resource requires access to a crontab program, typically cron.

**Warning**

The **cron** resource should only be used to modify an entry in a crontab file. Use the **cookbook\_file** or **template** resources to add a crontab file to the cron.d directory. The cron\_d lightweight resource (found in the cron cookbook) is another option for managing crontab files.

**Run a program at a specified interval**

cron 'noop' **do**

hour '5'

minute '0'

command '/bin/true'

**end**

**Run an entry if a folder exists**

cron 'ganglia\_tomcat\_thread\_max' **do**

command "/usr/bin/gmetric

-n 'tomcat threads max'

-t uint32

-v '/usr/local/bin/tomcat-stat

--thread-max'"

only\_if **do** File.exist?('/home/jboss') **end**

**end**

**Run every Saturday, 8:00 AM**

The following example shows a schedule that will run every hour at 8:00 each Saturday morning, and will then send an email to “admin@example.com” after each run.

cron 'name\_of\_cron\_entry' **do**

minute '0'

hour '8'

weekday '6'

mailto 'admin@example.com'

action :create

**end**

**Run only in November**

The following example shows a schedule that will run at 8:00 PM, every weekday (Monday through Friday), but only in November:

cron 'name\_of\_cron\_entry' **do**

minute '0'

hour '20'

day '\*'

month '11'

weekday '1-5'

action :create

**end**

**csh¶**

Use the **csh** resource to execute scripts using the csh interpreter. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_ifand only\_if to guard this resource for idempotence.

**Note**

The **csh** script resource (which is based on the **script** resource) is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

No examples.

**directory¶**

Use the **directory** resource to manage a directory, which is a hierarchy of folders that comprises all of the information stored on a computer. The root directory is the top-level, under which the rest of the directory is organized. The **directory** resource uses the name property to specify the path to a location in a directory. Typically, permission to access that location in the directory is required.

**Create a directory**

directory '/tmp/something' **do**

owner 'root'

group 'root'

mode '0755'

action :create

**end**

**Create a directory in Microsoft Windows**

directory "C:**\\**tmp**\\**something.txt" **do**

rights :full\_control, "DOMAIN**\\**User"

inherits false

action :create

**end**

or:

directory 'C:\tmp\something.txt' **do**

rights :full\_control, 'DOMAIN\User'

inherits false

action :create

**end**

**Note**

The difference between the two previous examples is the single- versus double-quoted strings, where if the double quotes are used, the backslash character (\) must be escaped using the Ruby escape character (which is a backslash).

**Create a directory recursively**

%w{dir1 dir2 dir3}.each **do** |dir|

directory "/tmp/mydirs/**#{**dir**}**" **do**

mode '0755'

owner 'root'

group 'root'

action :create

recursive true

**end**

**end**

**Delete a directory**

directory '/tmp/something' **do**

recursive true

action :delete

**end**

**Set directory permissions using a variable**

The following example shows how read/write/execute permissions can be set using a variable named user\_home, and then for owners and groups on any matching node:

user\_home = "/**#{**node[:matching\_node][:user]**}**"

directory user\_home **do**

owner 'node[:matching\_node][:user]'

group 'node[:matching\_node][:group]'

mode '0755'

action :create

**end**

where matching\_node represents a type of node. For example, if the user\_home variable specified {node[:nginx]...}, a recipe might look similar to:

user\_home = "/**#{**node[:nginx][:user]**}**"

directory user\_home **do**

owner 'node[:nginx][:user]'

group 'node[:nginx][:group]'

mode '0755'

action :create

**end**

**Set directory permissions for a specific type of node**

The following example shows how permissions can be set for the /certificates directory on any node that is running Nginx. In this example, permissions are being set for the owner and group properties as root, and then read/write permissions are granted to the root.

directory "**#{**node[:nginx][:dir]**}**/shared/certificates" **do**

owner 'root'

group 'root'

mode '0755'

recursive true

**end**

**Reload the configuration**

The following example shows how to reload the configuration of a chef-client using the **remote\_file** resource to:

* using an if statement to check whether the plugins on a node are the latest versions
* identify the location from which Ohai plugins are stored
* using the notifies property and a **ruby\_block** resource to trigger an update (if required) and to then reload the client.rb file.

directory 'node[:ohai][:plugin\_path]' **do**

owner 'chef'

recursive true

**end**

ruby\_block 'reload\_config' **do**

block **do**

Chef::Config.from\_file('/etc/chef/client.rb')

**end**

action :nothing

**end**

**if** node[:ohai].key?(:plugins)

node[:ohai][:plugins].each **do** |plugin|

remote\_file node[:ohai][:plugin\_path] +"/**#{**plugin**}**" **do**

source plugin

owner 'chef'

notifies :run, 'ruby\_block[reload\_config]', :immediately

**end**

**end**

**end**

**Manage dotfiles**

The following example shows using the **directory** and **cookbook\_file** resources to manage dotfiles. The dotfiles are defined by a JSON data structure similar to:

"files": {

".zshrc": {

"mode": '0755',

"source": "dot-zshrc"

},

".bashrc": {

"mode": '0755',

"source": "dot-bashrc"

},

".bash\_profile": {

"mode": '0755',

"source": "dot-bash\_profile"

},

}

and then the following resources manage the dotfiles:

**if** u.has\_key?('files')

u['files'].each **do** |filename, file\_data|

directory "**#{**home\_dir**}**/**#{**File.dirname(filename)**}**" **do**

recursive true

mode '0755'

**end** **if** file\_data['subdir']

cookbook\_file "**#{**home\_dir**}**/**#{**filename**}**" **do**

source "**#{**u['id']**}**/**#{**file\_data['source']**}**"

owner 'u['id']'

group 'group\_id'

mode 'file\_data['mode']'

ignore\_failure true

backup 0

**end**

**end**

**dpkg\_package¶**

Use the **dpkg\_package** resource to manage packages for the dpkg platform. When a package is installed from a local file, it must be added to the node using the **remote\_file** or **cookbook\_file** resources.

**Install a package**

dpkg\_package 'wget\_1.13.4-2ubuntu1.4\_amd64.deb' **do**

source '/foo/bar/wget\_1.13.4-2ubuntu1.4\_amd64.deb'

action :install

**end**

**dsc\_resource¶**

The **dsc\_resource** resource allows any DSC resource to be used in a Chef recipe, as well as any custom resources that have been added to your Windows PowerShell environment. Microsoft frequently adds new resources to the DSC resource collection.

**Open a Zip file**

dsc\_resource 'example' **do**

resource :archive

property :ensure, 'Present'

property :path, 'C:\Users\Public\Documents\example.zip'

property :destination, 'C:\Users\Public\Documents\ExtractionPath'

**end**

**Manage users and groups**

dsc\_resource 'demogroupadd' **do**

resource :group

property :groupname, 'demo1'

property :ensure, 'present'

**end**

dsc\_resource 'useradd' **do**

resource :user

property :username, 'Foobar1'

property :fullname, 'Foobar1'

property :password, ps\_credential('P@assword!')

property :ensure, 'present'

**end**

dsc\_resource 'AddFoobar1ToUsers' **do**

resource :Group

property :GroupName, 'demo1'

property :MembersToInclude, ['Foobar1']

**end**

**Create a test message queue**

The following example creates a file on a node (based on one that is located in a cookbook), unpacks the MessageQueue.zip Windows PowerShell module, and then uses the **dsc\_resource** to ensure that Message Queuing (MSMQ) sub-features are installed, a test queue is created, and that permissions are set on the test queue:

cookbook\_file 'cMessageQueue.zip' **do**

path "**#{**Chef::Config[:file\_cache\_path]**}\\**MessageQueue.zip"

action :create\_if\_missing

**end**

windows\_zipfile "**#{**ENV['PROGRAMW6432']**}\\**WindowsPowerShell**\\**Modules" **do**

source "**#{**Chef::Config[:file\_cache\_path]**}\\**MessageQueue.zip"

action :unzip

**end**

dsc\_resource 'install-sub-features' **do**

resource :windowsfeature

property :ensure, 'Present'

property :name, 'msmq'

property :IncludeAllSubFeature, true

**end**

dsc\_resource 'create-test-queue' **do**

resource :cPrivateMsmqQueue

property :ensure, 'Present'

property :name, 'Test\_Queue'

**end**

dsc\_resource 'set-permissions' **do**

resource :cPrivateMsmqQueuePermissions

property :ensure, 'Present'

property :name, 'Test\_Queue\_Permissions'

property :QueueNames, 'Test\_Queue'

property :ReadUsers, node['msmq']['read\_user']

**end**

**dsc\_script¶**

Many DSC resources are comparable to built-in Chef resources. For example, both DSC and Chef have **file**, **package**, and **service** resources. The **dsc\_script** resource is most useful for those DSC resources that do not have a direct comparison to a resource in Chef, such as the Archive resource, a custom DSC resource, an existing DSC script that performs an important task, and so on. Use the **dsc\_script** resource to embed the code that defines a DSC configuration directly within a Chef recipe.

New in Chef Client 12.2. Changed in Chef Client 12.6.

**Specify DSC code directly**

DSC data can be specified directly in a recipe:

dsc\_script 'emacs' **do**

code <<-EOH

Environment 'texteditor'

{

Name = 'EDITOR'

Value = 'c:\\emacs\\bin\\emacs.exe'

}

EOH

**end**

**Specify DSC code using a Windows Powershell data file**

Use the command property to specify the path to a Windows PowerShell data file. For example, the following Windows PowerShell script defines the DefaultEditor:

Configuration 'DefaultEditor'

{

Environment 'texteditor'

{

Name = 'EDITOR'

Value = 'c:\emacs\bin\emacs.exe'

}

}

Use the following recipe to specify the location of that data file:

dsc\_script 'DefaultEditor' **do**

command 'c:\dsc\_scripts\emacs.ps1'

**end**

**Pass parameters to DSC configurations**

If a DSC script contains configuration data that takes parameters, those parameters may be passed using the flags property. For example, the following Windows PowerShell script takes parameters for the EditorChoice and EditorFlags settings:

$choices = @{'emacs' = 'c:\emacs\bin\emacs';'vi' = 'c:\vim\vim.exe';'powershell' = 'powershell\_ise.exe'}

Configuration 'DefaultEditor'

{

[**CmdletBinding**()]

**param**

(

$EditorChoice,

$EditorFlags = ''

)

Environment 'TextEditor'

{

Name = 'EDITOR'

Value = "$($choices[$EditorChoice]) $EditorFlags"

}

}

Use the following recipe to set those parameters:

dsc\_script 'DefaultEditor' **do**

flags ({ :EditorChoice => 'emacs', :EditorFlags => '--maximized' })

command 'c:\dsc\_scripts\editors.ps1'

**end**

**Use custom configuration data**

Configuration data in DSC scripts may be customized from a recipe. For example, scripts are typically customized to set the behavior for Windows PowerShell credential data types. Configuration data may be specified in one of three ways:

* By using the configuration\_data attribute
* By using the configuration\_data\_script attribute
* By specifying the path to a valid Windows PowerShell data file

The following example shows how to specify custom configuration data using the configuration\_data property:

dsc\_script 'BackupUser' **do**

configuration\_data <<-EOH

@{

AllNodes = @(

@{

NodeName = "localhost";

PSDscAllowPlainTextPassword = $true

})

}

EOH

code <<-EOH

$user = 'backup'

$password = ConvertTo-SecureString -String "YourPass$(random)" -AsPlainText -Force

$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $user, $password

User $user

{

UserName = $user

Password = $cred

Description = 'Backup operator'

Ensure = "Present"

Disabled = $false

PasswordNeverExpires = $true

PasswordChangeRequired = $false

}

EOH

**end**

The following example shows how to specify custom configuration data using the configuration\_name property. For example, the following Windows PowerShell script defines the vi configuration:

Configuration 'emacs'

{

Environment 'TextEditor'

{

Name = 'EDITOR'

Value = 'c:\emacs\bin\emacs.exe'

}

}

Configuration 'vi'

{

Environment 'TextEditor'

{

Name = 'EDITOR'

Value = 'c:\vim\bin\vim.exe'

}

}

Use the following recipe to specify that configuration:

dsc\_script 'EDITOR' **do**

configuration\_name 'vi'

command 'C:\dsc\_scripts\editors.ps1'

**end**

**Using DSC with other Chef resources**

The **dsc\_script** resource can be used with other resources. The following example shows how to download a file using the **remote\_file** resource, and then uncompress it using the DSC Archive resource:

remote\_file "**#{**Chef::Config[:file\_cache\_path]**}\\**DSCResourceKit620082014.zip" **do**

source 'http://gallery.technet.microsoft.com/DSC-Resource-Kit-All-c449312d/file/124481/1/DSC%20Resource%20Kit%20Wave%206%2008282014.zip'

**end**

dsc\_script 'get-dsc-resource-kit' **do**

code <<-EOH

Archive reskit

{

ensure = 'Present'

path = "#{Chef::Config[:file\_cache\_path]}\\DSCResourceKit620082014.zip"

destination = "#{ENV['PROGRAMW6432']}\\WindowsPowerShell\\Modules"

}

EOH

**end**

**env¶**

Use the **windows\_env** resource to manage environment keys in Microsoft Windows. After an environment key is set, Microsoft Windows must be restarted before the environment key will be available to the Task Scheduler.

This resource was previously called the **env** resource; its name was updated in Chef Client 14.0 to reflect the fact that only Windows is supported. Existing cookbooks using env will continue to function, but should be updated to use the new name.

**Set an environment variable**

windows\_env 'ComSpec' **do**

value "C:**\\**Windows**\\**system32**\\**cmd.exe"

**end**

**execute¶**

Use the **execute** resource to execute a single command. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_ifand only\_if to guard this resource for idempotence.

**Run a command upon notification**

execute 'slapadd' **do**

command 'slapadd < /tmp/something.ldif'

creates '/var/lib/slapd/uid.bdb'

action :nothing

**end**

template '/tmp/something.ldif' **do**

source 'something.ldif'

notifies :run, 'execute[slapadd]', :immediately

**end**

**Run a touch file only once while running a command**

execute 'upgrade script' **do**

command 'php upgrade-application.php && touch /var/application/.upgraded'

creates '/var/application/.upgraded'

action :run

**end**

**Run a command which requires an environment variable**

execute 'slapadd' **do**

command 'slapadd < /tmp/something.ldif'

creates '/var/lib/slapd/uid.bdb'

action :run

environment ({'HOME' => '/home/myhome'})

**end**

**Delete a repository using yum to scrub the cache**

*# the following code sample thanks to gaffneyc @ https://gist.github.com/918711*

execute 'clean-yum-cache' **do**

command 'yum clean all'

action :nothing

**end**

file '/etc/yum.repos.d/bad.repo' **do**

action :delete

notifies :run, 'execute[clean-yum-cache]', :immediately

notifies :create, 'ruby\_block[reload-internal-yum-cache]', :immediately

**end**

**Install repositories from a file, trigger a command, and force the internal cache to reload**

The following example shows how to install new Yum repositories from a file, where the installation of the repository triggers a creation of the Yum cache that forces the internal cache for the chef-client to reload:

execute 'create-yum-cache' **do**

command 'yum -q makecache'

action :nothing

**end**

ruby\_block 'reload-internal-yum-cache' **do**

block **do**

Chef::Provider::Package::Yum::YumCache.instance.reload

**end**

action :nothing

**end**

cookbook\_file '/etc/yum.repos.d/custom.repo' **do**

source 'custom'

mode '0755'

notifies :run, 'execute[create-yum-cache]', :immediately

notifies :create, 'ruby\_block[reload-internal-yum-cache]', :immediately

**end**

**Prevent restart and reconfigure if configuration is broken**

Use the :nothing action (common to all resources) to prevent the test from starting automatically, and then use the subscribes notification to run a configuration test when a change to the template is detected:

execute 'test-nagios-config' **do**

command 'nagios3 --verify-config'

action :nothing

subscribes :run, 'template[/etc/nagios3/configures-nagios.conf]', :immediately

**end**

**Notify in a specific order**

To notify multiple resources, and then have these resources run in a certain order, do something like the following:

execute 'foo' **do**

command '...'

notifies :create, 'template[baz]', :immediately

notifies :install, 'package[bar]', :immediately

notifies :run, 'execute[final]', :immediately

**end**

template 'baz' **do**

...

notifies :run, 'execute[restart\_baz]', :immediately

**end**

package 'bar'

execute 'restart\_baz'

execute 'final' **do**

command '...'

**end**

where the sequencing will be in the same order as the resources are listed in the recipe: execute 'foo', template 'baz', execute [restart\_baz], package 'bar', and execute 'final'.

**Execute a command using a template**

The following example shows how to set up IPv4 packet forwarding using the **execute** resource to run a command named forward\_ipv4 that uses a template defined by the **template** resource:

execute 'forward\_ipv4' **do**

command 'echo > /proc/.../ipv4/ip\_forward'

action :nothing

**end**

template '/etc/file\_name.conf' **do**

source 'routing/file\_name.conf.erb'

notifies :run, 'execute[forward\_ipv4]', :delayed

**end**

where the command property for the **execute** resource contains the command that is to be run and the sourceproperty for the **template** resource specifies which template to use. The notifies property for the **template**specifies that the execute[forward\_ipv4] (which is defined by the **execute** resource) should be queued up and run at the end of the chef-client run.

**Add a rule to an IP table**

The following example shows how to add a rule named test\_rule to an IP table using the **execute** resource to run a command using a template that is defined by the **template** resource:

execute 'test\_rule' **do**

command 'command\_to\_run

--option value

...

--option value

--source #{node[:name\_of\_node][:ipsec][:local][:subnet]}

-j test\_rule'

action :nothing

**end**

template '/etc/file\_name.local' **do**

source 'routing/file\_name.local.erb'

notifies :run, 'execute[test\_rule]', :delayed

**end**

where the command property for the **execute** resource contains the command that is to be run and the sourceproperty for the **template** resource specifies which template to use. The notifies property for the **template**specifies that the execute[test\_rule] (which is defined by the **execute** resource) should be queued up and run at the end of the chef-client run.

**Stop a service, do stuff, and then restart it**

The following example shows how to use the **execute**, **service**, and **mount** resources together to ensure that a node running on Amazon EC2 is running MySQL. This example does the following:

* Checks to see if the Amazon EC2 node has MySQL
* If the node has MySQL, stops MySQL
* Installs MySQL
* Mounts the node
* Restarts MySQL

*# the following code sample comes from the ``server\_ec2``*

*# recipe in the following cookbook:*

*# https://github.com/chef-cookbooks/mysql*

**if** (node.attribute?('ec2') && ! FileTest.directory?(node['mysql']['ec2\_path']))

service 'mysql' **do**

action :stop

**end**

execute 'install-mysql' **do**

command "mv **#{**node['mysql']['data\_dir']**}** **#{**node['mysql']['ec2\_path']**}**"

not\_if **do** FileTest.directory?(node['mysql']['ec2\_path']) **end**

**end**

[node['mysql']['ec2\_path'], node['mysql']['data\_dir']].each **do** |dir|

directory dir **do**

owner 'mysql'

group 'mysql'

**end**

**end**

mount node['mysql']['data\_dir'] **do**

device node['mysql']['ec2\_path']

fstype 'none'

options 'bind,rw'

action [:mount, :enable]

**end**

service 'mysql' **do**

action :start

**end**

**end**

where

* the two **service** resources are used to stop, and then restart the MySQL service
* the **execute** resource is used to install MySQL
* the **mount** resource is used to mount the node and enable MySQL

**Use the platform\_family? method**

The following is an example of using the platform\_family? method in the Recipe DSL to create a variable that can be used with other resources in the same recipe. In this example, platform\_family? is being used to ensure that a specific binary is used for a specific platform before using the **remote\_file** resource to download a file from a remote location, and then using the **execute** resource to install that file by running a command.

**if** platform\_family?('rhel')

pip\_binary = '/usr/bin/pip'

**else**

pip\_binary = '/usr/local/bin/pip'

**end**

remote\_file "**#{**Chef::Config[:file\_cache\_path]**}**/distribute\_setup.py" **do**

source 'http://python-distribute.org/distribute\_setup.py'

mode '0755'

not\_if { File.exist?(pip\_binary) }

**end**

execute 'install-pip' **do**

cwd Chef::Config[:file\_cache\_path]

command <<-EOF

# command for installing Python goes here

EOF

not\_if { File.exist?(pip\_binary) }

**end**

where a command for installing Python might look something like:

*#{node['python']['binary']} distribute\_setup.py*

*#{::File.dirname(pip\_binary)}/easy\_install pip*

**Control a service using the execute resource**

**Warning**

This is an example of something that should NOT be done. Use the **service** resource to control a service, not the **execute**resource.

Do something like this:

service 'tomcat' **do**

action :start

**end**

and NOT something like this:

execute 'start-tomcat' **do**

command '/etc/init.d/tomcat6 start'

action :run

**end**

There is no reason to use the **execute** resource to control a service because the **service** resource exposes the start\_command property directly, which gives a recipe full control over the command issued in a much cleaner, more direct manner.

**Use the search recipe DSL method to find users**

The following example shows how to use the search method in the Recipe DSL to search for users:

*# the following code sample comes from the openvpn cookbook: https://github.com/chef-cookbooks/openvpn*

search("users", "\*:\*") **do** |u|

execute "generate-openvpn-**#{**u['id']**}**" **do**

command "./pkitool **#{**u['id']**}**"

cwd '/etc/openvpn/easy-rsa'

environment(

'EASY\_RSA' => '/etc/openvpn/easy-rsa',

'KEY\_CONFIG' => '/etc/openvpn/easy-rsa/openssl.cnf',

'KEY\_DIR' => node['openvpn']['key\_dir'],

'CA\_EXPIRE' => node['openvpn']['key']['ca\_expire'].to\_s,

'KEY\_EXPIRE' => node['openvpn']['key']['expire'].to\_s,

'KEY\_SIZE' => node['openvpn']['key']['size'].to\_s,

'KEY\_COUNTRY' => node['openvpn']['key']['country'],

'KEY\_PROVINCE' => node['openvpn']['key']['province'],

'KEY\_CITY' => node['openvpn']['key']['city'],

'KEY\_ORG' => node['openvpn']['key']['org'],

'KEY\_EMAIL' => node['openvpn']['key']['email']

)

not\_if { File.exist?("**#{**node['openvpn']['key\_dir']**}**/**#{**u['id']**}**.crt") }

**end**

%w{ conf ovpn }.each **do** |ext|

template "**#{**node['openvpn']['key\_dir']**}**/**#{**u['id']**}**.**#{**ext**}**" **do**

source 'client.conf.erb'

variables :username => u['id']

**end**

**end**

execute "create-openvpn-tar-**#{**u['id']**}**" **do**

cwd node['openvpn']['key\_dir']

command <<-EOH

tar zcf #{u['id']}.tar.gz \

ca.crt #{u['id']}.crt #{u['id']}.key \

#{u['id']}.conf #{u['id']}.ovpn \

EOH

not\_if { File.exist?("**#{**node['openvpn']['key\_dir']**}**/**#{**u['id']**}**.tar.gz") }

**end**

**end**

where

* the search will use both of the **execute** resources, unless the condition specified by the not\_if commands are met
* the environments property in the first **execute** resource is being used to define values that appear as variables in the OpenVPN configuration
* the **template** resource tells the chef-client which template to use

**Enable remote login for macOS**

execute 'enable ssh' **do**

command '/usr/sbin/systemsetup -setremotelogin on'

not\_if '/usr/sbin/systemsetup -getremotelogin | /usr/bin/grep On'

action :run

**end**

**Execute code immediately, based on the template resource**

By default, notifications are :delayed, that is they are queued up as they are triggered, and then executed at the very end of a chef-client run. To run an action immediately, use :immediately:

template '/etc/nagios3/configures-nagios.conf' **do**

*# other parameters*

notifies :run, 'execute[test-nagios-config]', :immediately

**end**

and then the chef-client would immediately run the following:

execute 'test-nagios-config' **do**

command 'nagios3 --verify-config'

action :nothing

**end**

**Sourcing a file**

The **execute** resource cannot be used to source a file (e.g. command 'source filename'). The following example will fail because source is not an executable:

execute 'foo' **do**

command 'source /tmp/foo.sh'

**end**

Instead, use the **script** resource or one of the **script**-based resources (**bash**, **csh**, **perl**, **python**, or **ruby**). For example:

bash 'foo' **do**

code 'source /tmp/foo.sh'

**end**

**Run a Knife command**

execute 'create\_user' **do**

command <<-EOM.gsub(/\s+/, ' ').strip!

knife user create #{user}

--admin

--password password

--disable-editing

--file /home/vagrant/.chef/user.pem

--config /tmp/knife-admin.rb

EOM

**end**

**Run install command into virtual environment**

The following example shows how to install a lightweight JavaScript framework into Vagrant:

execute "install q and zombiejs" **do**

cwd "/home/vagrant"

user "vagrant"

environment ({'HOME' => '/home/vagrant', 'USER' => 'vagrant'})

command "npm install -g q zombie should mocha coffee-script"

action :run

**end**

**Run a command as a named user**

The following example shows how to run bundle install from a chef-client run as a specific user. This will put the gem into the path of the user (vagrant) instead of the root user (under which the chef-client runs):

execute '/opt/chefdk/embedded/bin/bundle install' **do**

cwd node['chef\_workstation']['bundler\_path']

user node['chef\_workstation']['user']

environment ({

'HOME' => "/home/**#{**node['chef\_workstation']['user']**}**",

'USER' => node['chef\_workstation']['user']

})

not\_if 'bundle check'

**end**

**file¶**

Use the **file** resource to manage files directly on a node.

**Create a file**

file '/tmp/something' **do**

owner 'root'

group 'root'

mode '0755'

action :create

**end**

**Create a file in Microsoft Windows**

To create a file in Microsoft Windows, be sure to add an escape character—\—before the backslashes in the paths:

file 'C:\\tmp\\something.txt' **do**

rights :read, 'Everyone'

rights :full\_control, 'DOMAIN\\User'

action :create

**end**

**Remove a file**

file '/tmp/something' **do**

action :delete

**end**

**Set file modes**

file '/tmp/something' **do**

mode '0755'

**end**

**Delete a repository using yum to scrub the cache**

*# the following code sample thanks to gaffneyc @ https://gist.github.com/918711*

execute 'clean-yum-cache' **do**

command 'yum clean all'

action :nothing

**end**

file '/etc/yum.repos.d/bad.repo' **do**

action :delete

notifies :run, 'execute[clean-yum-cache]', :immediately

notifies :create, 'ruby\_block[reload-internal-yum-cache]', :immediately

**end**

**Add the value of a data bag item to a file**

The following example shows how to get the contents of a data bag item named impossible\_things, create a .pem file located at some/directory/path/, and then use the content attribute to update the contents of that file with the value of the impossible\_things data bag item:

private\_key = data\_bag\_item('impossible\_things', private\_key\_name)['private\_key']

file "some/directory/path/**#{**private\_key\_name**}**.pem" **do**

content private\_key

owner 'root'

group 'group'

mode '0755'

**end**

**Write a YAML file**

The following example shows how to use the content property to write a YAML file:

file "**#{**app['deploy\_to']**}**/shared/config/settings.yml" **do**

owner "app['owner']"

group "app['group']"

mode '0755'

content app.to\_yaml

**end**

**Write a string to a file**

The following example specifies a directory, and then uses the content property to add a string to the file created in that directory:

status\_file = '/path/to/file/status\_file'

file status\_file **do**

owner 'root'

group 'root'

mode '0755'

content 'My favourite foremost coastal Antarctic shelf, oh Larsen B!'

**end**

**Create a file from a copy**

The following example shows how to copy a file from one directory to another, locally on a node:

file '/root/1.txt' **do**

content IO.read('/tmp/1.txt')

action :create

**end**

where the content attribute uses the Ruby IO.read method to get the contents of the /tmp/1.txt file.

**freebsd\_package¶**

Use the **freebsd\_package** resource to manage packages for the FreeBSD platform.

**Install a package**

freebsd\_package 'name of package' **do**

action :install

**end**

**gem\_package¶**

Use the **gem\_package** resource to manage gem packages that are only included in recipes. When a package is installed from a local file, it must be added to the node using the **remote\_file** or **cookbook\_file** resources.

**Install a gems file from the local file system**

gem\_package 'right\_aws' **do**

source '/tmp/right\_aws-1.11.0.gem'

action :install

**end**

**Use the ignore\_failure common attribute**

gem\_package 'syntax' **do**

action :install

ignore\_failure true

**end**

**git¶**

Use the **git** resource to manage source control resources that exist in a git repository. git version 1.6.5 (or higher) is required to use all of the functionality in the **git** resource.

**Use the git mirror**

git '/opt/mysources/couch' **do**

repository 'git://git.apache.org/couchdb.git'

revision 'master'

action :sync

**end**

**Use different branches**

To use different branches, depending on the environment of the node:

**if** node.chef\_environment == 'QA'

branch\_name = 'staging'

**else**

branch\_name = 'master'

**end**

git '/home/user/deployment' **do**

repository 'git@github.com:gitsite/deployment.git'

revision branch\_name

action :sync

user 'user'

group 'test'

**end**

where the branch\_name variable is set to staging or master, depending on the environment of the node. Once this is determined, the branch\_name variable is used to set the revision for the repository. If the git status command is used after running the example above, it will return the branch name as deploy, as this is the default value. Run the chef-client in debug mode to verify that the correct branches are being checked out:

$ sudo chef-client -l debug

**Install an application from git using bash**

The following example shows how Bash can be used to install a plug-in for rbenv named ruby-build, which is located in git version source control. First, the application is synchronized, and then Bash changes its working directory to the location in which ruby-build is located, and then runs a command.

git "**#{**Chef::Config[:file\_cache\_path]**}**/ruby-build" **do**

repository 'git://github.com/sstephenson/ruby-build.git'

reference 'master'

action :sync

**end**

bash 'install\_ruby\_build' **do**

cwd '#{Chef::Config[:file\_cache\_path]}/ruby-build'

user 'rbenv'

group 'rbenv'

code <<-EOH

./install.sh

EOH

environment 'PREFIX' => '/usr/local'

**end**

To read more about ruby-build, see here: https://github.com/sstephenson/ruby-build.

**Upgrade packages from git**

The following example uses the **git** resource to upgrade packages:

*# the following code sample comes from the ``source`` recipe*

*# in the ``libvpx-cookbook`` cookbook:*

*# https://github.com/enmasse-entertainment/libvpx-cookbook*

git "**#{**Chef::Config[:file\_cache\_path]**}**/libvpx" **do**

repository node[:libvpx][:git\_repository]

revision node[:libvpx][:git\_revision]

action :sync

notifies :run, 'bash[compile\_libvpx]', :immediately

**end**

**Pass in environment variables**

git '/opt/mysources/couch' **do**

repository 'git://git.apache.org/couchdb.git'

revision 'master'

environment 'VAR' => 'whatever'

action :sync

**end**

**group¶**

Use the **group** resource to manage a local group.

**Append users to groups**

group 'www-data' **do**

action :modify

members 'maintenance'

append true

**end**

**Add a user to group on the Windows platform**

group 'Administrators' **do**

members ['domain\foo']

append true

action :modify

**end**

**homebrew\_package¶**

Use the **homebrew\_package** resource to manage packages for the macOS platform.

New in Chef Client 12.0.

**Install a package**

homebrew\_package 'name of package' **do**

action :install

**end**

**Specify the Homebrew user with a UUID**

homebrew\_package 'emacs' **do**

homebrew\_user 1001

**end**

**Specify the Homebrew user with a string**

homebrew\_package 'vim' **do**

homebrew\_user 'user1'

**end**

**http\_request¶**

Use the **http\_request** resource to send an HTTP request (GET, PUT, POST, DELETE, HEAD, or OPTIONS) with an arbitrary message. This resource is often useful when custom callbacks are necessary.

**Send a GET request**

http\_request 'some\_message' **do**

url 'http://example.com/check\_in'

**end**

The message is sent as http://example.com/check\_in?message=some\_message.

Changed in Chef Client 12.0 to deprecate the hard-coded query string from earlier versions. Cookbooks that rely on this string need to be updated to manually add it to the URL as it is passed to the resource.

**Send a POST request**

To send a POST request as JSON data, convert the message to JSON and include the correct content-type header. For example:

http\_request 'posting data' **do**

action :post

url 'http://example.com/check\_in'

message ({:some => 'data'}.to\_json)

headers({'AUTHORIZATION' => "Basic **#{**

Base64.encode64('username:password')**}**",

'Content-Type' => 'application/data'

})

**end**

**Transfer a file only when the remote source changes**

remote\_file '/tmp/couch.png' **do**

source 'http://couchdb.apache.org/img/sketch.png'

action :nothing

**end**

http\_request 'HEAD http://couchdb.apache.org/img/sketch.png' **do**

message ''

url 'http://couchdb.apache.org/img/sketch.png'

action :head

**if** File.exist?('/tmp/couch.png')

headers 'If-Modified-Since' => File.mtime('/tmp/couch.png').httpdate

**end**

notifies :create, 'remote\_file[/tmp/couch.png]', :immediately

**end**

**ifconfig¶**

Use the **ifconfig** resource to manage interfaces on \*nix systems.

**Configure a network interface**

ifconfig "33.33.33.80" **do**

bootproto "dhcp"

device "eth1"

**end**

will create the following interface:

vagrant@default-ubuntu-1204:~$ cat /etc/network/interfaces.d/ifcfg-eth1

iface eth1 inet dhcp

**Specify a boot protocol**

ifconfig '192.186.0.1' **do**

device 'eth0'

**end**

**Specify a static IP address**

ifconfig "33.33.33.80" **do**

device "eth1"

**end**

will create the following interface:

iface eth1 inet static

address 33.33.33.80

**Update a static IP address with a boot protocol**

ifconfig "33.33.33.80" **do**

bootproto "dhcp"

device "eth1"

**end**

will update the interface from static to dhcp:

iface eth1 inet dhcp

address 33.33.33.80

**ips\_package¶**

Use the **ips\_package** resource to manage packages (using Image Packaging System (IPS)) on the Solaris 11 platform.

**Install a package**

ips\_package 'name of package' **do**

action :install

**end**

**ksh¶**

Use the **ksh** resource to execute scripts using the Korn shell (ksh) interpreter. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence. New in Chef Client 12.6.

**Note**

The **ksh** script resource (which is based on the **script** resource) is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

No examples.

**link¶**

Use the **link** resource to create symbolic or hard links.

**Create symbolic links**

The following example will create a symbolic link from /tmp/file to /etc/file:

link '/tmp/file' **do**

to '/etc/file'

**end**

**Create hard links**

The following example will create a hard link from /tmp/file to /etc/file:

link '/tmp/file' **do**

to '/etc/file'

link\_type :hard

**end**

**Delete links**

The following example will delete the /tmp/file symbolic link and uses the only\_if guard to run the test -Lcommand, which verifies that /tmp/file is a symbolic link, and then only deletes /tmp/file if the test passes:

link '/tmp/file' **do**

action :delete

only\_if 'test -L /tmp/file'

**end**

**Create multiple symbolic links**

The following example creates symbolic links from two files in the /vol/webserver/cert/ directory to files located in the /etc/ssl/certs/ directory:

link '/vol/webserver/cert/server.crt' **do**

to '/etc/ssl/certs/ssl-cert-name.pem'

**end**

link '/vol/webserver/cert/server.key' **do**

to '/etc/ssl/certs/ssl-cert-name.key'

**end**

**Create platform-specific symbolic links**

The following example shows installing a filter module on Apache. The package name is different for different platforms, and for the Red Hat Enterprise Linux family, a symbolic link is required:

include\_recipe 'apache2::default'

**case** node['platform\_family']

**when** 'debian'

...

**when** 'suse'

...

**when** 'rhel', 'fedora'

...

link '/usr/lib64/httpd/modules/mod\_apreq.so' **do**

to '/usr/lib64/httpd/modules/mod\_apreq2.so'

only\_if 'test -f /usr/lib64/httpd/modules/mod\_apreq2.so'

**end**

link '/usr/lib/httpd/modules/mod\_apreq.so' **do**

to '/usr/lib/httpd/modules/mod\_apreq2.so'

only\_if 'test -f /usr/lib/httpd/modules/mod\_apreq2.so'

**end**

**end**

...

For the entire recipe, see https://github.com/onehealth-cookbooks/apache2/blob/68bdfba4680e70b3e90f77e40223dd535bf22c17/recipes/mod\_apreq2.rb.

**log¶**

Use the **log** resource to create log entries. The **log** resource behaves like any other resource: built into the resource collection during the compile phase, and then run during the execution phase. (To create a log entry that is not built into the resource collection, use Chef::Log instead of the **log** resource.)

**Note**

By default, every log resource that executes will count as an updated resource in the updated resource count at the end of a Chef run. You can disable this behavior by adding count\_log\_resource\_updates false to your Chef client.rb configuration file.

**Set default logging level**

log 'a string to log'

**Set debug logging level**

log 'a debug string' **do**

level :debug

**end**

**Add a message to a log file**

log 'message' **do**

message 'This is the message that will be added to the log.'

level :info

**end**

**macports\_package¶**

Use the **macports\_package** resource to manage packages for the macOS platform.

**Install a package**

macports\_package 'name of package' **do**

action :install

**end**

**mdadm¶**

Use the **mdadm** resource to manage RAID devices in a Linux environment using the mdadm utility. The **mdadm**resource will create and assemble an array, but it will not create the config file that is used to persist the array upon reboot. If the config file is required, it must be done by specifying a template with the correct array layout, and then by using the **mount** resource to create a file systems table (fstab) entry.

**Create and assemble a RAID 0 array**

The mdadm command can be used to create RAID arrays. For example, a RAID 0 array named /dev/md0 with 10 devices would have a command similar to the following:

$ mdadm --create /dev/md0 --level=0 --raid-devices=10 /dev/s01.../dev/s10

where /dev/s01 .. /dev/s10 represents 10 devices (01, 02, 03, and so on). This same command, when expressed as a recipe using the **mdadm** resource, would be similar to:

mdadm '/dev/md0' **do**

devices [ '/dev/s01', ... '/dev/s10' ]

level 0

action :create

**end**

(again, where /dev/s01 .. /dev/s10 represents devices /dev/s01, /dev/s02, /dev/s03, and so on).

**Create and assemble a RAID 1 array**

mdadm '/dev/md0' **do**

devices [ '/dev/sda', '/dev/sdb' ]

level 1

action [ :create, :assemble ]

**end**

**Create and assemble a RAID 5 array**

The mdadm command can be used to create RAID arrays. For example, a RAID 5 array named /dev/sd0 with 4, and a superblock type of 0.90 would be similar to:

mdadm '/dev/sd0' **do**

devices [ '/dev/s1', '/dev/s2', '/dev/s3', '/dev/s4' ]

level 5

metadata '0.90'

chunk 32

action :create

**end**

**mount¶**

Use the **mount** resource to manage a mounted file system.

**Mount a labeled file system**

mount '/mnt/volume1' **do**

device 'volume1'

device\_type :label

fstype 'xfs'

options 'rw'

**end**

**Mount a local block drive**

mount '/mnt/local' **do**

device '/dev/sdb1'

fstype 'ext3'

**end**

**Mount a non-block file system**

mount '/mount/tmp' **do**

pass 0

fstype 'tmpfs'

device '/dev/null'

options 'nr\_inodes=999k,mode=755,size=500m'

action [:mount, :enable]

**end**

**Mount and add to the file systems table**

mount '/export/www' **do**

device 'nas1prod:/export/web\_sites'

fstype 'nfs'

options 'rw'

action [:mount, :enable]

**end**

**Mount a remote file system**

mount '/export/www' **do**

device 'nas1prod:/export/web\_sites'

fstype 'nfs'

options 'rw'

**end**

**Mount a remote folder in Microsoft Windows**

mount 'T:' **do**

action :mount

device '\\\\hostname.example.com\\folder'

**end**

**Unmount a remote folder in Microsoft Windows**

mount 'T:' **do**

action :umount

device '\\\\hostname.example.com\\D$'

**end**

**Stop a service, do stuff, and then restart it**

The following example shows how to use the **execute**, **service**, and **mount** resources together to ensure that a node running on Amazon EC2 is running MySQL. This example does the following:

* Checks to see if the Amazon EC2 node has MySQL
* If the node has MySQL, stops MySQL
* Installs MySQL
* Mounts the node
* Restarts MySQL

*# the following code sample comes from the ``server\_ec2``*

*# recipe in the following cookbook:*

*# https://github.com/chef-cookbooks/mysql*

**if** (node.attribute?('ec2') && ! FileTest.directory?(node['mysql']['ec2\_path']))

service 'mysql' **do**

action :stop

**end**

execute 'install-mysql' **do**

command "mv **#{**node['mysql']['data\_dir']**}** **#{**node['mysql']['ec2\_path']**}**"

not\_if **do** FileTest.directory?(node['mysql']['ec2\_path']) **end**

**end**

[node['mysql']['ec2\_path'], node['mysql']['data\_dir']].each **do** |dir|

directory dir **do**

owner 'mysql'

group 'mysql'

**end**

**end**

mount node['mysql']['data\_dir'] **do**

device node['mysql']['ec2\_path']

fstype 'none'

options 'bind,rw'

action [:mount, :enable]

**end**

service 'mysql' **do**

action :start

**end**

**end**

where

* the two **service** resources are used to stop, and then restart the MySQL service
* the **execute** resource is used to install MySQL
* the **mount** resource is used to mount the node and enable MySQL

**ohai¶**

Use the **ohai** resource to reload the Ohai configuration on a node. This allows recipes that change system attributes (like a recipe that adds a user) to refer to those attributes later on during the chef-client run.

**Reload Ohai**

ohai 'reload' **do**

action :reload

**end**

**Reload Ohai after a new user is created**

ohai 'reload\_passwd' **do**

action :nothing

plugin 'etc'

**end**

user 'daemonuser' **do**

home '/dev/null'

shell '/sbin/nologin'

system true

notifies :reload, 'ohai[reload\_passwd]', :immediately

**end**

ruby\_block 'just an example' **do**

block **do**

*# These variables will now have the new values*

puts node['etc']['passwd']['daemonuser']['uid']

puts node['etc']['passwd']['daemonuser']['gid']

**end**

**end**

**openbsd\_package¶**

Use the **openbsd\_package** resource to manage packages for the OpenBSD platform.

**Install a package**

openbsd\_package 'name of package' **do**

action :install

**end**

New in Chef Client 12.1.

**osx\_profile¶**

Use the **osx\_profile** resource to manage configuration profiles (.mobileconfig files) on the macOS platform. The **osx\_profile** resource installs profiles by using the uuidgen library to generate a unique ProfileUUID, and then using the profiles command to install the profile on the system.

**One liner to install profile from cookbook file**

The profiles command will be used to install the specified configuration profile.

osx\_profile 'com.company.screensaver.mobileconfig'

**Install profile from cookbook file**

The profiles command will be used to install the specified configuration profile. It can be in sub-directory within a cookbook.

osx\_profile 'Install screensaver profile' **do**

profile 'screensaver/com.company.screensaver.mobileconfig'

**end**

**Install profile from a hash**

The profiles command will be used to install the configuration profile, which is provided as a hash.

profile\_hash = {

'PayloadIdentifier' => 'com.company.screensaver',

'PayloadRemovalDisallowed' => false,

'PayloadScope' => 'System',

'PayloadType' => 'Configuration',

'PayloadUUID' => '1781fbec-3325-565f-9022-8aa28135c3cc',

'PayloadOrganization' => 'Chef',

'PayloadVersion' => 1,

'PayloadDisplayName' => 'Screensaver Settings',

'PayloadContent'=> [

{

'PayloadType' => 'com.apple.ManagedClient.preferences',

'PayloadVersion' => 1,

'PayloadIdentifier' => 'com.company.screensaver',

'PayloadUUID' => '73fc30e0-1e57-0131-c32d-000c2944c108',

'PayloadEnabled' => true,

'PayloadDisplayName' => 'com.apple.screensaver',

'PayloadContent' => {

'com.apple.screensaver' => {

'Forced' => [

{

'mcx\_preference\_settings' => {

'idleTime' => 0,

}

}

]

}

}

}

]

}

osx\_profile 'Install screensaver profile' **do**

profile profile\_hash

**end**

**Remove profile using identifier in resource name**

The profiles command will be used to remove the configuration profile specified by the provided identifierproperty.

osx\_profile 'com.company.screensaver' **do**

action :remove

**end**

**Remove profile by identifier and user friendly resource name**

The profiles command will be used to remove the configuration profile specified by the provided identifierproperty.

osx\_profile 'Remove screensaver profile' **do**

identifier 'com.company.screensaver'

action :remove

**end**

**package¶**

Use the **package** resource to manage packages. When the package is installed from a local file (such as with RubyGems, dpkg, or RPM Package Manager), the file must be added to the node using the **remote\_file** or **cookbook\_file** resources.

**Install a gems file for use in recipes**

chef\_gem 'right\_aws' **do**

action :install

**end**

require 'right\_aws'

**Install a gems file from the local file system**

gem\_package 'right\_aws' **do**

source '/tmp/right\_aws-1.11.0.gem'

action :install

**end**

**Install a package**

package 'tar' **do**

action :install

**end**

**Install a package version**

package 'tar' **do**

version '1.16.1-1'

action :install

**end**

**Install a package with options**

package 'debian-archive-keyring' **do**

action :install

options '--force-yes'

**end**

**Install a package with a response\_file**

Use of a response\_file is only supported on Debian and Ubuntu at this time. Custom resources must be written to support the use of a response\_file, which contains debconf answers to questions normally asked by the package manager on installation. Put the file in /files/default of the cookbook where the package is specified and the chef-client will use the **cookbook\_file** resource to retrieve it.

To install a package with a response\_file:

package 'sun-java6-jdk' **do**

response\_file 'java.seed'

**end**

**Install a package using a specific provider**

package 'tar' **do**

action :install

source '/tmp/tar-1.16.1-1.rpm'

provider Chef::Provider::Package::Rpm

**end**

**Install a specified architecture using a named provider**

yum\_package 'glibc-devel' **do**

arch 'i386'

**end**

**Purge a package**

package 'tar' **do**

action :purge

**end**

**Remove a package**

package 'tar' **do**

action :remove

**end**

**Upgrade a package**

package 'tar' **do**

action :upgrade

**end**

**Use the ignore\_failure common attribute**

gem\_package 'syntax' **do**

action :install

ignore\_failure true

**end**

**Use the provider common attribute**

package 'some\_package' **do**

provider Chef::Provider::Package::Rubygems

**end**

**Avoid unnecessary string interpolation**

Do this:

package 'mysql-server' **do**

version node['mysql']['version']

action :install

**end**

and not this:

package 'mysql-server' **do**

version "**#{**node['mysql']['version']**}**"

action :install

**end**

**Install a package in a platform**

The following example shows how to use the **package** resource to install an application named app and ensure that the correct packages are installed for the correct platform:

package 'app\_name' **do**

action :install

**end**

**case** node[:platform]

**when** 'ubuntu','debian'

package 'app\_name-doc' **do**

action :install

**end**

**when** 'centos'

package 'app\_name-html' **do**

action :install

**end**

**end**

**Install sudo, then configure /etc/sudoers/ file**

The following example shows how to install sudo and then configure the /etc/sudoers file:

*# the following code sample comes from the ``default`` recipe in the ``sudo`` cookbook: https://github.com/chef-cookbooks/sudo*

package 'sudo' **do**

action :install

**end**

**if** node['authorization']['sudo']['include\_sudoers\_d']

directory '/etc/sudoers.d' **do**

mode '0755'

owner 'root'

group 'root'

action :create

**end**

cookbook\_file '/etc/sudoers.d/README' **do**

source 'README'

mode '0440'

owner 'root'

group 'root'

action :create

**end**

**end**

template '/etc/sudoers' **do**

source 'sudoers.erb'

mode '0440'

owner 'root'

group platform?('freebsd') ? 'wheel' : 'root'

variables(

:sudoers\_groups => node['authorization']['sudo']['groups'],

:sudoers\_users => node['authorization']['sudo']['users'],

:passwordless => node['authorization']['sudo']['passwordless']

)

**end**

where

* the **package** resource is used to install sudo
* the if statement is used to ensure availability of the /etc/sudoers.d directory
* the **template** resource tells the chef-client where to find the sudoers template
* the variables property is a hash that passes values to template files (that are located in the templates/directory for the cookbook

**Use a case statement to specify the platform**

The following example shows how to use a case statement to tell the chef-client which platforms and packages to install using cURL.

package 'curl'

**case** node[:platform]

**when** 'redhat', 'centos'

package 'package\_1'

package 'package\_2'

package 'package\_3'

**when** 'ubuntu', 'debian'

package 'package\_a'

package 'package\_b'

package 'package\_c'

**end**

**end**

where node[:platform] for each node is identified by Ohai during every chef-client run. For example:

package 'curl'

**case** node[:platform]

**when** 'redhat', 'centos'

package 'zlib-devel'

package 'openssl-devel'

package 'libc6-dev'

**when** 'ubuntu', 'debian'

package 'openssl'

package 'pkg-config'

package 'subversion'

**end**

**end**

**Use symbols to reference attributes**

Symbols may be used to reference attributes:

package 'mysql-server' **do**

version node[:mysql][:version]

action :install

**end**

instead of strings:

package 'mysql-server' **do**

version node['mysql']['version']

action :install

**end**

**Use a whitespace array to simplify a recipe**

The following examples show different ways of doing the same thing. The first shows a series of packages that will be upgraded:

package 'package-a' **do**

action :upgrade

**end**

package 'package-b' **do**

action :upgrade

**end**

package 'package-c' **do**

action :upgrade

**end**

package 'package-d' **do**

action :upgrade

**end**

and the next uses a single **package** resource and a whitespace array (%w):

%w{package-a package-b package-c package-d}.each **do** |pkg|

package pkg **do**

action :upgrade

**end**

**end**

where |pkg| is used to define the name of the resource, but also to ensure that each item in the whitespace array has its own name.

**Specify the Homebrew user with a UUID**

homebrew\_package 'emacs' **do**

homebrew\_user 1001

**end**

**Specify the Homebrew user with a string**

homebrew\_package 'vim' **do**

homebrew\_user 'user1'

**end**

**pacman\_package¶**

Use the **pacman\_package** resource to manage packages (using pacman) on the Arch Linux platform.

**Install a package**

pacman\_package 'name of package' **do**

action :install

**end**

**paludis\_package¶**

Use the **paludis\_package** resource to manage packages for the Paludis platform.

**Install a package**

paludis\_package 'name of package' **do**

action :install

**end**

New in Chef Client 12.1.

**perl¶**

Use the **perl** resource to execute scripts using the Perl interpreter. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_ifand only\_if to guard this resource for idempotence.

**Note**

The **perl** script resource (which is based on the **script** resource) is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

No examples.

**portage\_package¶**

Use the **portage\_package** resource to manage packages for the Gentoo platform.

**Install a package**

portage\_package 'name of package' **do**

action :install

**end**

**powershell\_script¶**

Use the **powershell\_script** resource to execute a script using the Windows PowerShell interpreter, much like how the **script** and **script**-based resources—**bash**, **csh**, **perl**, **python**, and **ruby**—are used. The **powershell\_script** is specific to the Microsoft Windows platform and the Windows PowerShell interpreter.

The **powershell\_script** resource creates and executes a temporary file (similar to how the **script** resource behaves), rather than running the command inline. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence.

**Write to an interpolated path**

powershell\_script 'write-to-interpolated-path' **do**

code <<-EOH

$stream = [System.IO.StreamWriter] "#{Chef::Config[:file\_cache\_path]}/powershell-test.txt"

$stream.WriteLine("In #{Chef::Config[:file\_cache\_path]}...word.")

$stream.close()

EOH

**end**

**Change the working directory**

powershell\_script 'cwd-then-write' **do**

cwd Chef::Config[:file\_cache\_path]

code <<-EOH

$stream = [System.IO.StreamWriter] "C:/powershell-test2.txt"

$pwd = pwd

$stream.WriteLine("This is the contents of: $pwd")

$dirs = dir

foreach ($dir in $dirs) {

$stream.WriteLine($dir.fullname)

}

$stream.close()

EOH

**end**

**Change the working directory in Microsoft Windows**

powershell\_script 'cwd-to-win-env-var' **do**

cwd '%TEMP%'

code <<-EOH

$stream = [System.IO.StreamWriter] "./temp-write-from-chef.txt"

$stream.WriteLine("chef on windows rox yo!")

$stream.close()

EOH

**end**

**Pass an environment variable to a script**

powershell\_script 'read-env-var' **do**

cwd Chef::Config[:file\_cache\_path]

environment ({'foo' => 'BAZ'})

code <<-EOH

$stream = [System.IO.StreamWriter] "./test-read-env-var.txt"

$stream.WriteLine("FOO is $env:foo")

$stream.close()

EOH

**end**

**Evaluate for true and/or false**

Use the convert\_boolean\_return attribute to raise an exception when certain conditions are met. For example, the following fragments will run successfully without error:

powershell\_script 'false' **do**

code '$false'

**end**

and:

powershell\_script 'true' **do**

code '$true'

**end**

whereas the following will raise an exception:

powershell\_script 'false' **do**

convert\_boolean\_return true

code '$false'

**end**

**Use the flags attribute**

powershell\_script 'Install IIS' **do**

code <<-EOH

Import-Module ServerManager

Add-WindowsFeature Web-Server

EOH

flags '-NoLogo, -NonInteractive, -NoProfile, -ExecutionPolicy Unrestricted, -InputFormat None, -File'

guard\_interpreter :powershell\_script

not\_if '(Get-WindowsFeature -Name Web-Server).Installed'

**end**

**Rename computer, join domain, reboot**

The following example shows how to rename a computer, join a domain, and then reboot the computer:

reboot 'Restart Computer' **do**

action :nothing

**end**

powershell\_script 'Rename and Join Domain' **do**

code <<-EOH

...your rename and domain join logic here...

EOH

not\_if <<-EOH

$ComputerSystem = gwmi win32\_computersystem

($ComputerSystem.Name -like '#{node['some\_attribute\_that\_has\_the\_new\_name']}') -and

$ComputerSystem.partofdomain)

EOH

notifies :reboot\_now, 'reboot[Restart Computer]', :immediately

**end**

where:

* The **powershell\_script** resource block renames a computer, and then joins a domain
* The **reboot** resource restarts the computer
* The not\_if guard prevents the Windows PowerShell script from running when the settings in the not\_ifguard match the desired state
* The notifies statement tells the **reboot** resource block to run if the **powershell\_script** block was executed during the chef-client run

**python¶**

Use the **python** resource to execute scripts using the Python interpreter. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence.

**Note**

The **python** script resource (which is based on the **script** resource) is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

No examples.

**reboot¶**

Use the **reboot** resource to reboot a node, a necessary step with some installations on certain platforms. This resource is supported for use on the Microsoft Windows, macOS, and Linux platforms. New in Chef Client 12.0.

**Reboot a node immediately**

reboot 'now' **do**

action :nothing

reason 'Cannot continue Chef run without a reboot.'

delay\_mins 2

**end**

execute 'foo' **do**

command '...'

notifies :reboot\_now, 'reboot[now]', :immediately

**end**

**Reboot a node at the end of a chef-client run**

reboot 'app\_requires\_reboot' **do**

action :request\_reboot

reason 'Need to reboot when the run completes successfully.'

delay\_mins 5

**end**

**Cancel a reboot**

reboot 'cancel\_reboot\_request' **do**

action :cancel

reason 'Cancel a previous end-of-run reboot request.'

**end**

**Rename computer, join domain, reboot**

The following example shows how to rename a computer, join a domain, and then reboot the computer:

reboot 'Restart Computer' **do**

action :nothing

**end**

powershell\_script 'Rename and Join Domain' **do**

code <<-EOH

...your rename and domain join logic here...

EOH

not\_if <<-EOH

$ComputerSystem = gwmi win32\_computersystem

($ComputerSystem.Name -like '#{node['some\_attribute\_that\_has\_the\_new\_name']}') -and

$ComputerSystem.partofdomain)

EOH

notifies :reboot\_now, 'reboot[Restart Computer]', :immediately

**end**

where:

* The **powershell\_script** resource block renames a computer, and then joins a domain
* The **reboot** resource restarts the computer
* The not\_if guard prevents the Windows PowerShell script from running when the settings in the not\_ifguard match the desired state
* The notifies statement tells the **reboot** resource block to run if the **powershell\_script** block was executed during the chef-client run

**registry\_key¶**

Use the **registry\_key** resource to create and delete registry keys in Microsoft Windows.

**Create a registry key**

Use a double-quoted string:

registry\_key "HKEY\_LOCAL\_MACHINE**\\**path-to-key**\\**Policies**\\**System" **do**

values [{

name: 'EnableLUA',

type: :dword,

data: 0

}]

action :create

**end**

or a single-quoted string:

registry\_key 'HKEY\_LOCAL\_MACHINE\path-to-key\Policies\System' **do**

values [{

name: 'EnableLUA',

type: :dword,

data: 0

}]

action :create

**end**

**Delete a registry key value**

Use a double-quoted string:

registry\_key "HKEY\_LOCAL\_MACHINE**\\**SOFTWARE**\\**path**\\**to**\\**key**\\**AU" **do**

values [{

name: 'NoAutoRebootWithLoggedOnUsers',

type: :dword,

data: ''

}]

action :delete

**end**

or a single-quoted string:

registry\_key 'HKEY\_LOCAL\_MACHINE\SOFTWARE\path\to\key\AU' **do**

values [{

name: 'NoAutoRebootWithLoggedOnUsers',

type: :dword,

data: ''

}]

action :delete

**end**

**Note**

If data: is not specified, you get an error: Missing data key in RegistryKey values hash

**Delete a registry key and its subkeys, recursively**

Use a double-quoted string:

registry\_key "HKCU**\\**SOFTWARE**\\**Policies**\\**path**\\**to**\\**key**\\**Themes" **do**

recursive true

action :delete\_key

**end**

or a single-quoted string:

registry\_key 'HKCU\SOFTWARE\Policies\path\to\key\Themes' **do**

recursive true

action :delete\_key

**end**

**Note**

Be careful when using the :delete\_key action with the recursive attribute. This will delete the registry key, all of its values and all of the names, types, and data associated with them. This cannot be undone by the chef-client.

**Use re-directed keys**

In 64-bit versions of Microsoft Windows, HKEY\_LOCAL\_MACHINE\SOFTWARE\Example is a re-directed key. In the following examples, because HKEY\_LOCAL\_MACHINE\SOFTWARE\Example is a 32-bit key, the output will be “Found 32-bit key” if they are run on a version of Microsoft Windows that is 64-bit:

registry\_key "HKEY\_LOCAL\_MACHINE**\\**SOFTWARE**\\**Example" **do**

architecture :i386

recursive true

action :create

**end**

or:

registry\_key "HKEY\_LOCAL\_MACHINE**\\**SOFTWARE**\\**Example" **do**

architecture :x86\_64

recursive true

action :delete\_key

**end**

or:

ruby\_block 'check 32-bit' **do**

block **do**

puts 'Found 32-bit key'

**end**

only\_if {

registry\_key\_exists?("HKEY\_LOCAL\_MACHINE\SOFTWARE**\\**Example",

:i386)

}

**end**

or:

ruby\_block 'check 64-bit' **do**

block **do**

puts 'Found 64-bit key'

**end**

only\_if {

registry\_key\_exists?("HKEY\_LOCAL\_MACHINE**\\**SOFTWARE**\\**Example",

:x86\_64)

}

**end**

**Set proxy settings to be the same as those used by the chef-client**

Use a double-quoted string:

proxy = URI.parse(Chef::Config[:http\_proxy])

registry\_key "HKCU\Software\Microsoft\path**\t**o\key\Internet Settings" **do**

values [{name: 'ProxyEnable', type: :reg\_dword, data: 1},

{name: 'ProxyServer', data: "**#{**proxy.host**}**:**#{**proxy.port**}**"},

{name: 'ProxyOverride', type: :reg\_string, data: <local>},

]

action :create

**end**

or a single-quoted string:

proxy = URI.parse(Chef::Config[:http\_proxy])

registry\_key 'HKCU\Software\Microsoft\path\to\key\Internet Settings' **do**

values [{name: 'ProxyEnable', type: :reg\_dword, data: 1},

{name: 'ProxyServer', data: "**#{**proxy.host**}**:**#{**proxy.port**}**"},

{name: 'ProxyOverride', type: :reg\_string, data: <local>},

]

action :create

**end**

**Set the name of a registry key to “(Default)”**

Use a double-quoted string:

registry\_key 'Set (Default) value' **do**

key "HKLM**\\**Software**\\**Test**\\**Key**\\**Path"

values [

{name: '', type: :string, data: 'test'},

]

action :create

**end**

or a single-quoted string:

registry\_key 'Set (Default) value' **do**

key 'HKLM\Software\Test\Key\Path'

values [

{name: '', type: :string, data: 'test'},

]

action :create

**end**

where name: '' contains an empty string, which will set the name of the registry key to (Default).

**remote\_directory¶**

Use the **remote\_directory** resource to incrementally transfer a directory from a cookbook to a node. The directory that is copied from the cookbook should be located under COOKBOOK\_NAME/files/default/REMOTE\_DIRECTORY. The **remote\_directory** resource will obey file specificity.

**Recursively transfer a directory from a remote location**

*# create up to 10 backups of the files*

*# set the files owner different from the directory*

remote\_directory '/tmp/remote\_something' **do**

source 'something'

files\_backup 10

files\_owner 'root'

files\_group 'root'

files\_mode '0644'

owner 'nobody'

group 'nobody'

mode '0755'

**end**

**Use with the chef\_handler lightweight resource**

The following example shows how to use the **remote\_directory** resource and the **chef\_handler** resource to reboot a handler named WindowsRebootHandler:

*# the following code sample comes from the*

*# ``reboot\_handler`` recipe in the ``windows`` cookbook:*

*# https://github.com/chef-cookbooks/windows*

remote\_directory node['chef\_handler']['handler\_path'] **do**

source 'handlers'

recursive true

action :create

**end**

chef\_handler 'WindowsRebootHandler' **do**

source "**#{**node['chef\_handler']['handler\_path']**}**/windows\_reboot\_handler.rb"

arguments node['windows']['allow\_pending\_reboots']

supports :report => true, :exception => false

action :enable

**end**

**remote\_file¶**

Use the **remote\_file** resource to transfer a file from a remote location using file specificity. This resource is similar to the **file** resource.

**Transfer a file from a URL**

remote\_file '/tmp/testfile' **do**

source 'http://www.example.com/tempfiles/testfile'

mode '0755'

checksum '3a7dac00b1' *# A SHA256 (or portion thereof) of the file.*

**end**

**Install a file from a remote location using bash**

The following is an example of how to install the foo123 module for Nginx. This module adds shell-style functionality to an Nginx configuration file and does the following:

* Declares three variables
* Gets the Nginx file from a remote location
* Installs the file using Bash to the path specified by the src\_filepath variable

*# the following code sample is similar to the ``upload\_progress\_module``*

*# recipe in the ``nginx`` cookbook:*

*# https://github.com/chef-cookbooks/nginx*

src\_filename = "foo123-nginx-module-v**#{**

node['nginx']['foo123']['version']

**}**.tar.gz"

src\_filepath = "**#{**Chef::Config['file\_cache\_path']**}**/**#{**src\_filename**}**"

extract\_path = "**#{**

Chef::Config['file\_cache\_path']

**}**/nginx\_foo123\_module/**#{**

node['nginx']['foo123']['checksum']

**}**"

remote\_file 'src\_filepath' **do**

source node['nginx']['foo123']['url']

checksum node['nginx']['foo123']['checksum']

owner 'root'

group 'root'

mode '0755'

**end**

bash 'extract\_module' **do**

cwd ::File.dirname(src\_filepath)

code <<-EOH

mkdir -p #{extract\_path}

tar xzf #{src\_filename} -C #{extract\_path}

mv #{extract\_path}/\*/\* #{extract\_path}/

EOH

not\_if { ::File.exist?(extract\_path) }

**end**

**Store certain settings**

The following recipe shows how an attributes file can be used to store certain settings. An attributes file is located in the attributes/ directory in the same cookbook as the recipe which calls the attributes file. In this example, the attributes file specifies certain settings for Python that are then used across all nodes against which this recipe will run.

Python packages have versions, installation directories, URLs, and checksum files. An attributes file that exists to support this type of recipe would include settings like the following:

default['python']['version'] = '2.7.1'

**if** python['install\_method'] == 'package'

default['python']['prefix\_dir'] = '/usr'

**else**

default['python']['prefix\_dir'] = '/usr/local'

**end**

default['python']['url'] = 'http://www.python.org/ftp/python'

default['python']['checksum'] = '80e387...85fd61'

and then the methods in the recipe may refer to these values. A recipe that is used to install Python will need to do the following:

* Identify each package to be installed (implied in this example, not shown)
* Define variables for the package version and the install\_path
* Get the package from a remote location, but only if the package does not already exist on the target system
* Use the **bash** resource to install the package on the node, but only when the package is not already installed

*# the following code sample comes from the ``oc-nginx`` cookbook on |github|: https://github.com/cookbooks/oc-nginx*

version = node['python']['version']

install\_path = "**#{**node['python']['prefix\_dir']**}**/lib/python**#{**version.split(/(^\d+\.\d+)/)[1]**}**"

remote\_file "**#{**Chef::Config[:file\_cache\_path]**}**/Python-**#{**version**}**.tar.bz2" **do**

source "**#{**node['python']['url']**}**/**#{**version**}**/Python-**#{**version**}**.tar.bz2"

checksum node['python']['checksum']

mode '0755'

not\_if { ::File.exist?(install\_path) }

**end**

bash 'build-and-install-python' **do**

cwd Chef::Config[:file\_cache\_path]

code <<-EOF

tar -jxvf Python-#{version}.tar.bz2

(cd Python-#{version} && ./configure #{configure\_options})

(cd Python-#{version} && make && make install)

EOF

not\_if { ::File.exist?(install\_path) }

**end**

**Use the platform\_family? method**

The following is an example of using the platform\_family? method in the Recipe DSL to create a variable that can be used with other resources in the same recipe. In this example, platform\_family? is being used to ensure that a specific binary is used for a specific platform before using the **remote\_file** resource to download a file from a remote location, and then using the **execute** resource to install that file by running a command.

**if** platform\_family?('rhel')

pip\_binary = '/usr/bin/pip'

**else**

pip\_binary = '/usr/local/bin/pip'

**end**

remote\_file "**#{**Chef::Config[:file\_cache\_path]**}**/distribute\_setup.py" **do**

source 'http://python-distribute.org/distribute\_setup.py'

mode '0755'

not\_if { File.exist?(pip\_binary) }

**end**

execute 'install-pip' **do**

cwd Chef::Config[:file\_cache\_path]

command <<-EOF

# command for installing Python goes here

EOF

not\_if { File.exist?(pip\_binary) }

**end**

where a command for installing Python might look something like:

*#{node['python']['binary']} distribute\_setup.py*

*#{::File.dirname(pip\_binary)}/easy\_install pip*

**Specify local Windows file path as a valid URI**

When specifying a local Microsoft Windows file path as a valid file URI, an additional forward slash (/) is required. For example:

remote\_file 'file:///c:/path/to/file' **do**

... *# other attributes*

**end**

**route¶**

Use the **route** resource to manage the system routing table in a Linux environment.

**Add a host route**

route '10.0.1.10/32' **do**

gateway '10.0.0.20'

device 'eth1'

**end**

**Delete a network route**

route '10.1.1.0/24' **do**

gateway '10.0.0.20'

action :delete

**end**

**rpm\_package¶**

Use the **rpm\_package** resource to manage packages for the RPM Package Manager platform.

**Install a package**

rpm\_package 'name of package' **do**

action :install

**end**

**ruby¶**

Use the **ruby** resource to execute scripts using the Ruby interpreter. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence.

**Note**

The **ruby** script resource (which is based on the **script** resource) is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

No examples.

**ruby\_block¶**

Use the **ruby\_block** resource to execute Ruby code during a chef-client run. Ruby code in the ruby\_blockresource is evaluated with other resources during convergence, whereas Ruby code outside of a ruby\_blockresource is evaluated before other resources, as the recipe is compiled.

**Re-read configuration data**

ruby\_block 'reload\_client\_config' **do**

block **do**

Chef::Config.from\_file('/etc/chef/client.rb')

**end**

action :run

**end**

**Install repositories from a file, trigger a command, and force the internal cache to reload**

The following example shows how to install new Yum repositories from a file, where the installation of the repository triggers a creation of the Yum cache that forces the internal cache for the chef-client to reload:

execute 'create-yum-cache' **do**

command 'yum -q makecache'

action :nothing

**end**

ruby\_block 'reload-internal-yum-cache' **do**

block **do**

Chef::Provider::Package::Yum::YumCache.instance.reload

**end**

action :nothing

**end**

cookbook\_file '/etc/yum.repos.d/custom.repo' **do**

source 'custom'

mode '0755'

notifies :run, 'execute[create-yum-cache]', :immediately

notifies :create, 'ruby\_block[reload-internal-yum-cache]', :immediately

**end**

**Use an if statement with the platform recipe DSL method**

The following example shows how an if statement can be used with the platform? method in the Recipe DSL to run code specific to Microsoft Windows. The code is defined using the **ruby\_block** resource:

*# the following code sample comes from the ``client`` recipe*

*# in the following cookbook: https://github.com/chef-cookbooks/mysql*

**if** platform?('windows')

ruby\_block 'copy libmysql.dll into ruby path' **do**

block **do**

require 'fileutils'

FileUtils.cp "**#{**node['mysql']['client']['lib\_dir']**}\\**libmysql.dll",

node['mysql']['client']['ruby\_dir']

**end**

not\_if { File.exist?("**#{**node['mysql']['client']['ruby\_dir']**}\\**libmysql.dll") }

**end**

**end**

**Stash a file in a data bag**

The following example shows how to use the **ruby\_block** resource to stash a BitTorrent file in a data bag so that it can be distributed to nodes in the organization.

*# the following code sample comes from the ``seed`` recipe*

*# in the following cookbook: https://github.com/mattray/bittorrent-cookbook*

ruby\_block 'share the torrent file' **do**

block **do**

f = File.open(node['bittorrent']['torrent'],'rb')

*#read the .torrent file and base64 encode it*

enc = Base64.encode64(f.read)

data = {

'id'=>bittorrent\_item\_id(node['bittorrent']['file']),

'seed'=>node.ipaddress,

'torrent'=>enc

}

item = Chef::DataBagItem.new

item.data\_bag('bittorrent')

item.raw\_data = data

item.save

**end**

action :nothing

subscribes :create, "bittorrent\_torrent[**#{**node['bittorrent']['torrent']**}**]", :immediately

**end**

**Update the /etc/hosts file**

The following example shows how the **ruby\_block** resource can be used to update the /etc/hosts file:

*# the following code sample comes from the ``ec2`` recipe*

*# in the following cookbook: https://github.com/chef-cookbooks/dynect*

ruby\_block 'edit etc hosts' **do**

block **do**

rc = Chef::Util::FileEdit.new('/etc/hosts')

rc.search\_file\_replace\_line(/^127\.0\.0\.1 localhost$/,

'127.0.0.1 #{new\_fqdn} #{new\_hostname} localhost')

rc.write\_file

**end**

**end**

**Set environment variables**

The following example shows how to use variables within a Ruby block to set environment variables using rbenv.

node.override[:rbenv][:root] = rbenv\_root

node.override[:ruby\_build][:bin\_path] = rbenv\_binary\_path

ruby\_block 'initialize' **do**

block **do**

ENV['RBENV\_ROOT'] = node[:rbenv][:root]

ENV['PATH'] = "**#{**node[:rbenv][:root]**}**/bin:**#{**node[:ruby\_build][:bin\_path]**}**:**#{**ENV['PATH']**}**"

**end**

**end**

**Set JAVA\_HOME**

The following example shows how to use a variable within a Ruby block to set the java\_home environment variable:

ruby\_block 'set-env-java-home' **do**

block **do**

ENV['JAVA\_HOME'] = java\_home

**end**

**end**

**Reload the configuration**

The following example shows how to reload the configuration of a chef-client using the **remote\_file** resource to:

* using an if statement to check whether the plugins on a node are the latest versions
* identify the location from which Ohai plugins are stored
* using the notifies property and a **ruby\_block** resource to trigger an update (if required) and to then reload the client.rb file.

directory 'node[:ohai][:plugin\_path]' **do**

owner 'chef'

recursive true

**end**

ruby\_block 'reload\_config' **do**

block **do**

Chef::Config.from\_file('/etc/chef/client.rb')

**end**

action :nothing

**end**

**if** node[:ohai].key?(:plugins)

node[:ohai][:plugins].each **do** |plugin|

remote\_file node[:ohai][:plugin\_path] +"/**#{**plugin**}**" **do**

source plugin

owner 'chef'

notifies :run, 'ruby\_block[reload\_config]', :immediately

**end**

**end**

**end**

**script¶**

Use the **script** resource to execute scripts using a specified interpreter, such as Bash, csh, Perl, Python, or Ruby. This resource may also use any of the actions and properties that are available to the **execute** resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not\_if and only\_if to guard this resource for idempotence.

**Note**

The **script** resource is different from the **ruby\_block** resource because Ruby code that is run with this resource is created as a temporary file and executed like other script resources, rather than run inline.

**Use a named provider to run a script**

bash 'install\_something' **do**

user 'root'

cwd '/tmp'

code <<-EOH

wget http://www.example.com/tarball.tar.gz

tar -zxf tarball.tar.gz

cd tarball

./configure

make

make install

EOH

**end**

**Run a script**

script 'install\_something' **do**

interpreter 'bash'

user 'root'

cwd '/tmp'

code <<-EOH

wget http://www.example.com/tarball.tar.gz

tar -zxf tarball.tar.gz

cd tarball

./configure

make

make install

EOH

**end**

or something like:

bash 'openvpn-server-key' **do**

environment('KEY\_CN' => 'server')

code <<-EOF

openssl req -batch -days #{node['openvpn']['key']['expire']} \

-nodes -new -newkey rsa:#{key\_size} -keyout #{key\_dir}/server.key \

-out #{key\_dir}/server.csr -extensions server \

-config #{key\_dir}/openssl.cnf

EOF

not\_if { File.exist?('#{key\_dir}/server.crt') }

**end**

where code contains the OpenSSL command to be run. The not\_if property tells the chef-client not to run the command if the file already exists.

**Install a file from a remote location using bash**

The following is an example of how to install the foo123 module for Nginx. This module adds shell-style functionality to an Nginx configuration file and does the following:

* Declares three variables
* Gets the Nginx file from a remote location
* Installs the file using Bash to the path specified by the src\_filepath variable

*# the following code sample is similar to the ``upload\_progress\_module``*

*# recipe in the ``nginx`` cookbook:*

*# https://github.com/chef-cookbooks/nginx*

src\_filename = "foo123-nginx-module-v**#{**

node['nginx']['foo123']['version']

**}**.tar.gz"

src\_filepath = "**#{**Chef::Config['file\_cache\_path']**}**/**#{**src\_filename**}**"

extract\_path = "**#{**

Chef::Config['file\_cache\_path']

**}**/nginx\_foo123\_module/**#{**

node['nginx']['foo123']['checksum']

**}**"

remote\_file 'src\_filepath' **do**

source node['nginx']['foo123']['url']

checksum node['nginx']['foo123']['checksum']

owner 'root'

group 'root'

mode '0755'

**end**

bash 'extract\_module' **do**

cwd ::File.dirname(src\_filepath)

code <<-EOH

mkdir -p #{extract\_path}

tar xzf #{src\_filename} -C #{extract\_path}

mv #{extract\_path}/\*/\* #{extract\_path}/

EOH

not\_if { ::File.exist?(extract\_path) }

**end**

**Install an application from git using bash**

The following example shows how Bash can be used to install a plug-in for rbenv named ruby-build, which is located in git version source control. First, the application is synchronized, and then Bash changes its working directory to the location in which ruby-build is located, and then runs a command.

git "**#{**Chef::Config[:file\_cache\_path]**}**/ruby-build" **do**

repository 'git://github.com/sstephenson/ruby-build.git'

reference 'master'

action :sync

**end**

bash 'install\_ruby\_build' **do**

cwd '#{Chef::Config[:file\_cache\_path]}/ruby-build'

user 'rbenv'

group 'rbenv'

code <<-EOH

./install.sh

EOH

environment 'PREFIX' => '/usr/local'

**end**

To read more about ruby-build, see here: https://github.com/sstephenson/ruby-build.

**Store certain settings**

The following recipe shows how an attributes file can be used to store certain settings. An attributes file is located in the attributes/ directory in the same cookbook as the recipe which calls the attributes file. In this example, the attributes file specifies certain settings for Python that are then used across all nodes against which this recipe will run.

Python packages have versions, installation directories, URLs, and checksum files. An attributes file that exists to support this type of recipe would include settings like the following:

default['python']['version'] = '2.7.1'

**if** python['install\_method'] == 'package'

default['python']['prefix\_dir'] = '/usr'

**else**

default['python']['prefix\_dir'] = '/usr/local'

**end**

default['python']['url'] = 'http://www.python.org/ftp/python'

default['python']['checksum'] = '80e387...85fd61'

and then the methods in the recipe may refer to these values. A recipe that is used to install Python will need to do the following:

* Identify each package to be installed (implied in this example, not shown)
* Define variables for the package version and the install\_path
* Get the package from a remote location, but only if the package does not already exist on the target system
* Use the **bash** resource to install the package on the node, but only when the package is not already installed

*# the following code sample comes from the ``oc-nginx`` cookbook on |github|: https://github.com/cookbooks/oc-nginx*

version = node['python']['version']

install\_path = "**#{**node['python']['prefix\_dir']**}**/lib/python**#{**version.split(/(^\d+\.\d+)/)[1]**}**"

remote\_file "**#{**Chef::Config[:file\_cache\_path]**}**/Python-**#{**version**}**.tar.bz2" **do**

source "**#{**node['python']['url']**}**/**#{**version**}**/Python-**#{**version**}**.tar.bz2"

checksum node['python']['checksum']

mode '0755'

not\_if { ::File.exist?(install\_path) }

**end**

bash 'build-and-install-python' **do**

cwd Chef::Config[:file\_cache\_path]

code <<-EOF

tar -jxvf Python-#{version}.tar.bz2

(cd Python-#{version} && ./configure #{configure\_options})

(cd Python-#{version} && make && make install)

EOF

not\_if { ::File.exist?(install\_path) }

**end**

**service¶**

Use the **service** resource to manage a service.

**Start a service**

service 'example\_service' **do**

action :start

**end**

**Start a service, enable it**

service 'example\_service' **do**

supports :status => true, :restart => true, :reload => true

action [ :enable, :start ]

**end**

**Use a pattern**

service 'samba' **do**

pattern 'smbd'

action [:enable, :start]

**end**

**Use the :nothing common action**

service 'memcached' **do**

action :nothing

**end**

**Use the retries common attribute**

service 'apache' **do**

action [ :enable, :start ]

retries 3

**end**

**Use the retries and provider common attributes**

service 'some\_service' **do**

provider Chef::Provider::Service::Upstart

action [ :enable, :start ]

retries 3

**end**

**Manage a service, depending on the node platform**

service 'example\_service' **do**

**case** node['platform']

**when** 'centos','redhat','fedora'

service\_name 'redhat\_name'

**else**

service\_name 'other\_name'

**end**

supports :restart => true

action [ :enable, :start ]

**end**

**Change a service provider, depending on the node platform**

service 'example\_service' **do**

**case** node['platform']

**when** 'ubuntu'

**if** node['platform\_version'].to\_f >= 9.10

provider Chef::Provider::Service::Upstart

**end**

**end**

action [:enable, :start]

**end**

**Reload a service using a template**

To reload a service that is based on a template, use the **template** and **service** resources together in the same recipe, similar to the following:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

**end**

service 'apache' **do**

action :enable

subscribes :reload, 'template[/tmp/somefile]', :immediately

**end**

where the subscribes notification is used to reload the service whenever the template is modified.

**Enable a service after a restart or reload**

service 'apache' **do**

supports :restart => true, :reload => true

action :enable

**end**

**Set an IP address using variables and a template**

The following example shows how the **template** resource can be used in a recipe to combine settings stored in an attributes file, variables within a recipe, and a template to set the IP addresses that are used by the Nginx service. The attributes file contains the following:

default['nginx']['dir'] = '/etc/nginx'

The recipe then does the following to:

* Declare two variables at the beginning of the recipe, one for the remote IP address and the other for the authorized IP address
* Use the **service** resource to restart and reload the Nginx service
* Load a template named authorized\_ip.erb from the /templates directory that is used to set the IP address values based on the variables specified in the recipe

node.default['nginx']['remote\_ip\_var'] = 'remote\_addr'

node.default['nginx']['authorized\_ips'] = ['127.0.0.1/32']

service 'nginx' **do**

supports :status => true, :restart => true, :reload => true

**end**

template 'authorized\_ip' **do**

path "**#{**node['nginx']['dir']**}**/authorized\_ip"

source 'modules/authorized\_ip.erb'

owner 'root'

group 'root'

mode '0755'

variables(

:remote\_ip\_var => node['nginx']['remote\_ip\_var'],

:authorized\_ips => node['nginx']['authorized\_ips']

)

notifies :reload, 'service[nginx]', :immediately

**end**

where the variables property tells the template to use the variables set at the beginning of the recipe and the source property is used to call a template file located in the cookbook’s /templates directory. The template file looks similar to:

geo $<%= @remote\_ip\_var %> $authorized\_ip {

default no;

<% @authorized\_ips.each do |ip| %>

<%= "**#{**ip**}** yes;" %>

<% end %>

}

**Use a cron timer to manage a service**

The following example shows how to install the crond application using two resources and a variable:

*# the following code sample comes from the ``cron`` cookbook:*

*# https://github.com/chef-cookbooks/cron*

cron\_package = **case** node['platform']

**when** 'redhat', 'centos', 'scientific', 'fedora', 'amazon'

node['platform\_version'].to\_f >= 6.0 ? 'cronie' : 'vixie-cron'

**else**

'cron'

**end**

package cron\_package **do**

action :install

**end**

service 'crond' **do**

**case** node['platform']

**when** 'redhat', 'centos', 'scientific', 'fedora', 'amazon'

service\_name 'crond'

**when** 'debian', 'ubuntu', 'suse'

service\_name 'cron'

**end**

action [:start, :enable]

**end**

where

* cron\_package is a variable that is used to identify which platforms apply to which install packages
* the **package** resource uses the cron\_package variable to determine how to install the crond application on various nodes (with various platforms)
* the **service** resource enables the crond application on nodes that have Red Hat, CentOS, Red Hat Enterprise Linux, Fedora, or Amazon Web Services (AWS), and the cron service on nodes that run Debian, Ubuntu, or openSUSE

**Restart a service, and then notify a different service**

The following example shows how start a service named example\_service and immediately notify the Nginx service to restart.

service 'example\_service' **do**

action :start

notifies :restart, 'service[nginx]', :immediately

**end**

**Stop a service, do stuff, and then restart it**

The following example shows how to use the **execute**, **service**, and **mount** resources together to ensure that a node running on Amazon EC2 is running MySQL. This example does the following:

* Checks to see if the Amazon EC2 node has MySQL
* If the node has MySQL, stops MySQL
* Installs MySQL
* Mounts the node
* Restarts MySQL

*# the following code sample comes from the ``server\_ec2``*

*# recipe in the following cookbook:*

*# https://github.com/chef-cookbooks/mysql*

**if** (node.attribute?('ec2') && ! FileTest.directory?(node['mysql']['ec2\_path']))

service 'mysql' **do**

action :stop

**end**

execute 'install-mysql' **do**

command "mv **#{**node['mysql']['data\_dir']**}** **#{**node['mysql']['ec2\_path']**}**"

not\_if **do** FileTest.directory?(node['mysql']['ec2\_path']) **end**

**end**

[node['mysql']['ec2\_path'], node['mysql']['data\_dir']].each **do** |dir|

directory dir **do**

owner 'mysql'

group 'mysql'

**end**

**end**

mount node['mysql']['data\_dir'] **do**

device node['mysql']['ec2\_path']

fstype 'none'

options 'bind,rw'

action [:mount, :enable]

**end**

service 'mysql' **do**

action :start

**end**

**end**

where

* the two **service** resources are used to stop, and then restart the MySQL service
* the **execute** resource is used to install MySQL
* the **mount** resource is used to mount the node and enable MySQL

**Control a service using the execute resource**

**Warning**

This is an example of something that should NOT be done. Use the **service** resource to control a service, not the **execute**resource.

Do something like this:

service 'tomcat' **do**

action :start

**end**

and NOT something like this:

execute 'start-tomcat' **do**

command '/etc/init.d/tomcat6 start'

action :run

**end**

There is no reason to use the **execute** resource to control a service because the **service** resource exposes the start\_command property directly, which gives a recipe full control over the command issued in a much cleaner, more direct manner.

**Enable a service on AIX using the mkitab command**

The **service** resource does not support using the :enable and :disable actions with resources that are managed using System Resource Controller (SRC). This is because System Resource Controller (SRC) does not have a standard mechanism for enabling and disabling services on system boot.

One approach for enabling or disabling services that are managed by System Resource Controller (SRC) is to use the **execute** resource to invoke mkitab, and then use that command to enable or disable the service.

The following example shows how to install a service:

execute "install **#{**node['chef\_client']['svc\_name']**}** in SRC" **do**

command "mkssys -s **#{**node['chef\_client']['svc\_name']**}**

-p **#{**node['chef\_client']['bin']**}**

-u root

-S

-n 15

-f 9

-o **#{**node['chef\_client']['log\_dir']**}**/client.log

-e **#{**node['chef\_client']['log\_dir']**}**/client.log -a '

-i **#{**node['chef\_client']['interval']**}**

-s **#{**node['chef\_client']['splay']**}**'"

not\_if "lssrc -s **#{**node['chef\_client']['svc\_name']**}**"

action :run

**end**

and then enable it using the mkitab command:

execute "enable **#{**node['chef\_client']['svc\_name']**}**" **do**

command "mkitab '**#{**node['chef\_client']['svc\_name']**}**:2:once:/usr/bin/startsrc

-s **#{**node['chef\_client']['svc\_name']**}** > /dev/console 2>&1'"

not\_if "lsitab **#{**node['chef\_client']['svc\_name']**}**"

**end**

**smartos\_package¶**

Use the **smartos\_package** resource to manage packages for the SmartOS platform.

**Install a package**

smartos\_package 'name of package' **do**

action :install

**end**

**solaris\_package¶**

The **solaris\_package** resource is used to manage packages for the Solaris platform.

**Install a package**

solaris\_package 'name of package' **do**

source '/packages\_directory'

action :install

**end**

**subversion¶**

Use the **subversion** resource to manage source control resources that exist in a Subversion repository.

**Get the latest version of an application**

subversion 'CouchDB Edge' **do**

repository 'http://svn.apache.org/repos/asf/couchdb/trunk'

revision 'HEAD'

destination '/opt/mysources/couch'

action :sync

**end**

**template¶**

Use the **template** resource to manage the contents of a file using an Embedded Ruby (ERB) template by transferring files from a sub-directory of COOKBOOK\_NAME/templates/ to a specified path located on a host that is running the chef-client. This resource includes actions and properties from the **file** resource. Template files managed by the **template** resource follow the same file specificity rules as the **remote\_file** and **file** resources.

**Configure a file from a template**

template '/tmp/config.conf' **do**

source 'config.conf.erb'

**end**

**Configure a file from a local template**

template '/tmp/config.conf' **do**

local true

source '/tmp/config.conf.erb'

**end**

**Configure a file using a variable map**

template '/tmp/config.conf' **do**

source 'config.conf.erb'

variables(

:config\_var => node['configs']['config\_var']

)

**end**

**Use the not\_if condition**

The following example shows how to use the not\_if condition to create a file based on a template and using the presence of an attribute value on the node to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if { node['some\_value'] }

**end**

The following example shows how to use the not\_if condition to create a file based on a template and then Ruby code to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if **do**

File.exist?('/etc/passwd')

**end**

**end**

The following example shows how to use the not\_if condition to create a file based on a template and using a Ruby block (with curly braces) to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if { File.exist?('/etc/passwd') }

**end**

The following example shows how to use the not\_if condition to create a file based on a template and using a string to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

not\_if 'test -f /etc/passwd'

**end**

**Use the only\_if condition**

The following example shows how to use the only\_if condition to create a file based on a template and using the presence of an attribute on the node to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

only\_if { node['some\_value'] }

**end**

The following example shows how to use the only\_if condition to create a file based on a template, and then use Ruby to specify a condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

only\_if **do** ! File.exist?('/etc/passwd') **end**

**end**

The following example shows how to use the only\_if condition to create a file based on a template and using a string to specify the condition:

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

only\_if 'test -f /etc/passwd'

**end**

**Use a whitespace array (%w)**

The following example shows how to use a Ruby whitespace array to define a list of configuration tools, and then use that list of tools within the **template** resource to ensure that all of these configuration tools are using the same RSA key:

%w{openssl.cnf pkitool vars Rakefile}.each **do** |f|

template "/etc/openvpn/easy-rsa/**#{**f**}**" **do**

source "**#{**f**}**.erb"

owner 'root'

group 'root'

mode '0755'

**end**

**end**

**Use a relative path**

template "**#{**ENV['HOME']**}**/chef-getting-started.txt" **do**

source 'chef-getting-started.txt.erb'

mode '0755'

**end**

**Delay notifications**

template '/etc/nagios3/configures-nagios.conf' **do**

*# other parameters*

notifies :run, 'execute[test-nagios-config]', :delayed

**end**

**Notify immediately**

By default, notifications are :delayed, that is they are queued up as they are triggered, and then executed at the very end of a chef-client run. To run an action immediately, use :immediately:

template '/etc/nagios3/configures-nagios.conf' **do**

*# other parameters*

notifies :run, 'execute[test-nagios-config]', :immediately

**end**

and then the chef-client would immediately run the following:

execute 'test-nagios-config' **do**

command 'nagios3 --verify-config'

action :nothing

**end**

**Notify multiple resources**

template '/etc/chef/server.rb' **do**

source 'server.rb.erb'

owner 'root'

group 'root'

mode '0755'

notifies :restart, 'service[chef-solr]', :delayed

notifies :restart, 'service[chef-solr-indexer]', :delayed

notifies :restart, 'service[chef-server]', :delayed

**end**

**Reload a service**

template '/tmp/somefile' **do**

mode '0755'

source 'somefile.erb'

notifies :reload, 'service[apache]', :immediately

**end**

**Restart a service when a template is modified**

template '/etc/www/configures-apache.conf' **do**

notifies :restart, 'service[apache]', :immediately

**end**

**Send notifications to multiple resources**

To send notifications to multiple resources, just use multiple attributes. Multiple attributes will get sent to the notified resources in the order specified.

template '/etc/netatalk/netatalk.conf' **do**

notifies :restart, 'service[afpd]', :immediately

notifies :restart, 'service[cnid]', :immediately

**end**

service 'afpd'

service 'cnid'

**Execute a command using a template**

The following example shows how to set up IPv4 packet forwarding using the **execute** resource to run a command named forward\_ipv4 that uses a template defined by the **template** resource:

execute 'forward\_ipv4' **do**

command 'echo > /proc/.../ipv4/ip\_forward'

action :nothing

**end**

template '/etc/file\_name.conf' **do**

source 'routing/file\_name.conf.erb'

notifies :run, 'execute[forward\_ipv4]', :delayed

**end**

where the command property for the **execute** resource contains the command that is to be run and the sourceproperty for the **template** resource specifies which template to use. The notifies property for the **template**specifies that the execute[forward\_ipv4] (which is defined by the **execute** resource) should be queued up and run at the end of the chef-client run.

**Set an IP address using variables and a template**

The following example shows how the **template** resource can be used in a recipe to combine settings stored in an attributes file, variables within a recipe, and a template to set the IP addresses that are used by the Nginx service. The attributes file contains the following:

default['nginx']['dir'] = '/etc/nginx'

The recipe then does the following to:

* Declare two variables at the beginning of the recipe, one for the remote IP address and the other for the authorized IP address
* Use the **service** resource to restart and reload the Nginx service
* Load a template named authorized\_ip.erb from the /templates directory that is used to set the IP address values based on the variables specified in the recipe

node.default['nginx']['remote\_ip\_var'] = 'remote\_addr'

node.default['nginx']['authorized\_ips'] = ['127.0.0.1/32']

service 'nginx' **do**

supports :status => true, :restart => true, :reload => true

**end**

template 'authorized\_ip' **do**

path "**#{**node['nginx']['dir']**}**/authorized\_ip"

source 'modules/authorized\_ip.erb'

owner 'root'

group 'root'

mode '0755'

variables(

:remote\_ip\_var => node['nginx']['remote\_ip\_var'],

:authorized\_ips => node['nginx']['authorized\_ips']

)

notifies :reload, 'service[nginx]', :immediately

**end**

where the variables property tells the template to use the variables set at the beginning of the recipe and the source property is used to call a template file located in the cookbook’s /templates directory. The template file looks similar to:

geo $<%= @remote\_ip\_var %> $authorized\_ip {

default no;

<% @authorized\_ips.each do |ip| %>

<%= "**#{**ip**}** yes;" %>

<% end %>

}

**Add a rule to an IP table**

The following example shows how to add a rule named test\_rule to an IP table using the **execute** resource to run a command using a template that is defined by the **template** resource:

execute 'test\_rule' **do**

command 'command\_to\_run

--option value

...

--option value

--source #{node[:name\_of\_node][:ipsec][:local][:subnet]}

-j test\_rule'

action :nothing

**end**

template '/etc/file\_name.local' **do**

source 'routing/file\_name.local.erb'

notifies :run, 'execute[test\_rule]', :delayed

**end**

where the command property for the **execute** resource contains the command that is to be run and the sourceproperty for the **template** resource specifies which template to use. The notifies property for the **template**specifies that the execute[test\_rule] (which is defined by the **execute** resource) should be queued up and run at the end of the chef-client run.

**Apply proxy settings consistently across a Chef organization**

The following example shows how a template can be used to apply consistent proxy settings for all nodes of the same type:

template "**#{**node[:matching\_node][:dir]**}**/sites-available/site\_proxy.conf" **do**

source 'site\_proxy.matching\_node.conf.erb'

owner 'root'

group 'root'

mode '0755'

variables(

:ssl\_certificate => "**#{**node[:matching\_node][:dir]**}**/shared/certificates/site\_proxy.crt",

:ssl\_key => "**#{**node[:matching\_node][:dir]**}**/shared/certificates/site\_proxy.key",

:listen\_port => node[:site][:matching\_node\_proxy][:listen\_port],

:server\_name => node[:site][:matching\_node\_proxy][:server\_name],

:fqdn => node[:fqdn],

:server\_options => node[:site][:matching\_node][:server][:options],

:proxy\_options => node[:site][:matching\_node][:proxy][:options]

)

**end**

where matching\_node represents a type of node (like Nginx) and site\_proxy represents the type of proxy being used for that type of node (like Nexus).

**Get template settings from a local file**

The **template** resource can be used to render a template based on settings contained in a local file on disk or to get the settings from a template in a cookbook. Most of the time, the settings are retrieved from a template in a cookbook. The following example shows how the **template** resource can be used to retrieve these settings from a local file.

The following example is based on a few assumptions:

* The environment is a Ruby on Rails application that needs render a file named database.yml
* Information about the application—the user, their password, the server—is stored in a data bag on the Chef server
* The application is already deployed to the system and that only requirement in this example is to render the database.yml file

The application source tree looks something like:

myapp/

-> config/

-> database.yml.erb

**Note**

There should not be a file named database.yml (without the .erb), as the database.yml file is what will be rendered using the **template** resource.

The deployment of the app will end up in /srv, so the full path to this template would be something like /srv/myapp/current/config/database.yml.erb.

The content of the template itself may look like this:

<%= @rails\_env %>:

adapter: <%= @adapter %>

host: <%= @host %>

database: <%= @database %>

username: <%= @username %>

password: <%= @password %>

encoding: 'utf8'

reconnect: true

The recipe will be similar to the following:

results = search(:node, "role:myapp\_database\_master AND chef\_environment:**#{**node.chef\_environment**}**")

db\_master = results[0]

template '/srv/myapp/shared/database.yml' **do**

source '/srv/myapp/current/config/database.yml.erb'

local true

variables(

:rails\_env => node.chef\_environment,

:adapter => db\_master['myapp']['db\_adapter'],

:host => db\_master['fqdn'],

:database => "myapp\_**#{**node.chef\_environment**}**",

:username => "myapp",

:password => "SUPERSECRET",

)

**end**

where:

* the search method in the Recipe DSL is used to find the first node that is the database master (of which there should only be one)
* the :adapter variable property may also require an attribute to have been set on a role, which then determines the correct adapter

The template will render similar to the following:

production:

adapter: mysql

host: domU-12-31-39-14-F1-C3.compute-1.internal

database: myapp\_production

username: myapp

password: SUPERSECRET

encoding: utf8

reconnect: true

This example showed how to use the **template** resource to render a template based on settings contained in a local file. Some other issues that should be considered when using this type of approach include:

* Should the database.yml file be in a .gitignore file?
* How do developers run the application locally?
* Does this work with chef-solo?

**Pass values from recipe to template**

The following example shows how pass a value to a template using the variables property in the **template**resource. The template file is similar to:

[tcpout]

defaultGroup = splunk\_indexers\_<%= node['splunk']['receiver\_port'] %>

disabled=false

[tcpout:splunk\_indexers\_<%= node['splunk']['receiver\_port'] %>]

server=<% @splunk\_servers.map **do** |s| -%><%= s['ipaddress'] %>:<%= s['splunk']['receiver\_port'] %> <% end.join(', ') -%>

<% @outputs\_conf.each\_pair do |name, value| -%>

<%= name %> = <%= value %>

<% end -%>

The recipe then uses the variables attribute to find the values for splunk\_servers and outputs\_conf, before passing them into the template:

template "**#{**splunk\_dir**}**/etc/system/local/outputs.conf" **do**

source 'outputs.conf.erb'

mode '0755'

variables :splunk\_servers => splunk\_servers, :outputs\_conf => node['splunk']['outputs\_conf']

notifies :restart, 'service[splunk]'

**end**

This example can be found in the client.rb recipe and the outputs.conf.erb template files that are located in the chef-splunk cookbook that is maintained by Chef.

**user¶**

Use the **user** resource to add users, update existing users, remove users, and to lock/unlock user passwords.

**Note**

System attributes are collected by Ohai at the start of every chef-client run. By design, the actions available to the **user**resource are processed **after** the start of the chef-client run. This means that system attributes added or modified by the **user**resource during the chef-client run must be reloaded before they can be available to the chef-client. These system attributes can be reloaded in two ways: by picking up the values at the start of the (next) chef-client run or by using the ohai resource to reload the system attributes during the current chef-client run.

**Create a user named “random”**

user 'random' **do**

manage\_home true

comment 'User Random'

uid '1234'

gid '1234'

home '/home/random'

shell '/bin/bash'

password '$1$JJsvHslV$szsCjVEroftprNn4JHtDi'

**end**

**Create a system user**

user 'systemguy' **do**

comment 'system guy'

system true

shell '/bin/false'

**end**

**Create a system user with a variable**

The following example shows how to create a system user. In this instance, the home value is calculated and stored in a variable called user\_home which sets the user’s home attribute.

user\_home = "/home/**#{**node['cookbook\_name']['user']**}**"

user node['cookbook\_name']['user'] **do**

gid node['cookbook\_name']['group']

shell '/bin/bash'

home user\_home

system true

action :create

**end**

**Use SALTED-SHA512 passwords**

macOS 10.7 calculates the password shadow hash using SALTED-SHA512. The length of the shadow hash value is 68 bytes, the salt value is the first 4 bytes, with the remaining 64 being the shadow hash itself. The following code will calculate password shadow hashes for macOS 10.7:

password = 'my\_awesome\_password'

salt = OpenSSL::Random.random\_bytes(4)

encoded\_password = OpenSSL::Digest::SHA512.hexdigest(salt + password)

shadow\_hash = salt.unpack('H\*').first + encoded\_password

Use the calculated password shadow hash with the **user** resource:

user 'my\_awesome\_user' **do**

password 'c9b3bd....d843' *# Length: 136*

**end**

New in Chef Client 12.0.

**Use SALTED-SHA512-PBKDF2 passwords**

macOS 10.8 (and higher) calculates the password shadow hash using SALTED-SHA512-PBKDF2. The length of the shadow hash value is 128 bytes, the salt value is 32 bytes, and an integer specifies the number of iterations. The following code will calculate password shadow hashes for macOS 10.8 (and higher):

password = 'my\_awesome\_password'

salt = OpenSSL::Random.random\_bytes(32)

iterations = 25000 *# Any value above 20k should be fine.*

shadow\_hash = OpenSSL::PKCS5::pbkdf2\_hmac(

password,

salt,

iterations,

128,

OpenSSL::Digest::SHA512.new

).unpack('H\*').first

salt\_value = salt.unpack('H\*').first

Use the calculated password shadow hash with the **user** resource:

user 'my\_awesome\_user' **do**

password 'cbd1a....fc843' *# Length: 256*

salt 'bd1a....fc83' *# Length: 64*

iterations 25000

**end**

New in Chef Client 12.0.

**windows\_package¶**

Use the **windows\_package** resource to manage Microsoft Installer Package (MSI) packages for the Microsoft Windows platform.

**Install a package**

windows\_package '7zip' **do**

action :install

source 'C:\7z920.msi'

**end**

**Specify a URL for the source attribute**

windows\_package '7zip' **do**

source 'http://www.7-zip.org/a/7z938-x64.msi'

**end**

**Specify path and checksum**

windows\_package '7zip' **do**

source 'http://www.7-zip.org/a/7z938-x64.msi'

checksum '7c8e873991c82ad9cfc123415254ea6101e9a645e12977dcd518979e50fdedf3'

**end**

**Modify remote\_file resource attributes**

The **windows\_package** resource may specify a package at a remote location using the remote\_file\_attributes property. This uses the **remote\_file** resource to download the contents at the specified URL and passes in a Hash that modifes the properties of the remote\_file resource.

For example:

windows\_package '7zip' **do**

source 'http://www.7-zip.org/a/7z938-x64.msi'

remote\_file\_attributes ({

:path => 'C:\\7zip.msi',

:checksum => '7c8e873991c82ad9cfc123415254ea6101e9a645e12977dcd518979e50fdedf3'

})

**end**

**Download a nsis (Nullsoft) package resource**

windows\_package 'Mercurial 3.6.1 (64-bit)' **do**

source 'http://mercurial.selenic.com/release/windows/Mercurial-3.6.1-x64.exe'

checksum 'febd29578cb6736163d232708b834a2ddd119aa40abc536b2c313fc5e1b5831d'

**end**

**Download a custom package**

windows\_package 'Microsoft Visual C++ 2005 Redistributable' **do**

source 'https://download.microsoft.com/download/6/B/B/6BB661D6-A8AE-4819-B79F-236472F6070C/vcredist\_x86.exe'

installer\_type :custom

options '/Q'

**end**

**windows\_service¶**

Use the **windows\_service** resource to create, delete, and manage a service on the Microsoft Windows platform.

**Start a service manually**

windows\_service 'BITS' **do**

action :configure\_startup

startup\_type :manual

**end**

**yum\_package¶**

Use the **yum\_package** resource to install, upgrade, and remove packages with Yum for the Red Hat and CentOS platforms. The **yum\_package** resource is able to resolve provides data for packages much like Yum can do when it is run from the command line. This allows a variety of options for installing packages, like minimum versions, virtual provides, and library names.

**Install an exact version**

yum\_package 'netpbm = 10.35.58-8.el5'

**Install a minimum version**

yum\_package 'netpbm >= 10.35.58-8.el5'

**Install a minimum version using the default action**

yum\_package 'netpbm'

**To install a package**

yum\_package 'netpbm' **do**

action :install

**end**

**To install a partial minimum version**

yum\_package 'netpbm >= 10'

**To install a specific architecture**

yum\_package 'netpbm' **do**

arch 'i386'

**end**

or:

yum\_package 'netpbm.x86\_64'

**To install a specific version-release**

yum\_package 'netpbm' **do**

version '10.35.58-8.el5'

**end**

**To install a specific version (even when older than the current)**

yum\_package 'tzdata' **do**

version '2011b-1.el5'

allow\_downgrade true

**end**

**Handle cookbook\_file and yum\_package resources in the same recipe**

When a **cookbook\_file** resource and a **package** resource are both called from within the same recipe, use the flush\_cache attribute to dump the in-memory Yum cache, and then use the repository immediately to ensure that the correct package is installed:

cookbook\_file '/etc/yum.repos.d/custom.repo' **do**

source 'custom'

mode '0755'

**end**

package 'only-in-custom-repo' **do**

action :install

flush\_cache [ :before ]

**end**