Resources Quick Reference

In a Chef recipe, a resource is something that exists on a system along with its state. A service to be started, a package to be installed, or a user to be created are all examples of resources. Resource properties are shown here for illustration only. Read more at https://docs.chef.io/resources.html. Download Chef Server and more at https://downloads.chef.io.

package

Use to install, remove, and upgrade packages.

```
package 'httpd' do
  version '2.2.3'
  action :install
end
```

```
package 'httpd' # all default values are applied
```

template

Use to manage files from the '/templates' directory in a cookbook. Erubis templating processes the file contents after transferring it to a node.

```
template '/var/www/index.html' do
  owner 'root'
  group 'root'
  mode '0644'
  source 'index.html.erb'
  variables({ :key => value, :key => value })
  action :create
end
```

service

Use to manage services.

```
service 'ntpd' do
  action :start
end
```

file

Use to manage files directly on a node.

```
file '/etc/motd' do
  owner 'root'
  group 'root'
  mode '0644'
  content 'A string to write.'
  action :create
end
```

directory

Use to manage a directory.

```
directory '/data/alice' do
  owner 'alice'
  group 'root'
  mode '0750'
  recursive true
  action :create
end
```

execute

Use to execute a single, arbitrary command.

```
execute 'apt-get -y update'
```

```
execute 'apt-get -y update' do
  not_if { File.exist?('/var/tmp/apt-updated') }
end
```

```
execute 'apt-get-update' do
  command 'apt-get -y update'
  not_if { File.exist?('/var/tmp/apt-updated') }
end
```

user

Use to manage users and passwords.

```
user 'alice' do
  comment 'Alice is a user.'
  home '/home/alice'
  system false
  shell '/bin/sh'
  uid '501'
  action :create
end
```

group

Use to manage a local group.

```
group 'staff' do
  gid '501'
  members ['alice', 'bob']
  action :create
end
```

ruby_block

Use to execute arbitrary Ruby code during the chef-client run.

```
ruby_block 'hello' do
  block do
    puts 'Hello!'
  end
  action :run
end
```

log

Use to create log entries in the chef-client log file.

```
log 'A message to add to the log file.' do
  level :info
  action :write # default is STDOUT
end
```

remote_directory

Use to transfer a directory from a cookbook.

```
remote_directory '/path/to/directory' do
  owner 'root'
  group 'root'
  mode '0755'
  recursive true
  source 'directory_to_transfer'
  action :create
end
```

remote_file

Use to transfer files from a remote location.

```
remote_file '/data/alice/stuff.zip' do
  owner 'alice'
  group 'staff'
  mode '0644'
  checksum 'b57cbc304cbfe1cfce08...1a7e'
  source 'http://chef.io/stuff.zip'
  action :create
end
```

cookbook_file

Use to transfer files from the '/files' directory in a cookbook.

```
cookbook_file '/var/www/css/bootstrap.css' do
  owner 'root'
  group 'root'
  mode '0644'
  source 'bootstrap.css'
  action :create
end
```

script (bash, csh, perl, python, ruby)

Use to execute scripts with a specific interpreter.

```
script 'extract_module' do
  interpreter 'bash'
  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
```

Each interpreter has a specific resource. For example, **bash**:

```
bash 'extract_module' do
  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
```

git

Use to manage files that are sourced from a git repository.

```
git "#{Chef::Config[:file_cache_path]}/app_name" do
  repository node['app_name']['git_repository']
  revision node['app_name']['git_revision']
  action :sync
end
```

link

Use to create symbolic and hard links.

```
link '/path/to/target/file' do
  mode '0755'
  link_type :symbolic # or :hard
  to '/path/to/linked/file'
  action :create
end
```

Common properties/actions (use w/any resource)

Defer property evaluation until runtime:

```
property_name lazy { Ruby code }
```

Specify an interpreter:

```
guard_interpreter :bash # :python, :ruby, etc.
```

Continue chef-client run even if resource fails:

```
ignore_failure true
```

Prevent execution when true:

```
not_if { 'grep bob /etc/motd', :user => 'alice' }
```

Allow execution when true:

```
only_if { File.exist?('stuff.zip') }
```

Notify another resource when this one changes:

```
notifies :action, 'resource[name]'
```

Listen to another resource for changes:

```
subscribes :action, 'resource[name]'
```

Retry the resource:

```
retries 0 # number of times to retry
retry_delay 2 # retry delay, in seconds
```

Wait to be notified by another resource:

```
action :nothing # wait for notification
```

Example: package, template, service

Order matters! The package resource runs first, then the template resource, and then the service resource.

```
package 'haproxy' do
  action :install
end

template '/etc/haproxy/haproxy.cfg' do
  owner 'root'
  group 'root'
  mode '0644'
  source 'haproxy.cfg.erb'
  notifies :reload, 'service[haproxy]'
end

service 'haproxy' do
  supports :reload => true
  action [:start, :enable]
end
```

Read more:

* **https://docs.chef.io/resources.html**
* **https://docs.chef.io/resource_examples.html**

CHEF™