

Creating a Custom Resource



Objectives

After completing this module, you should be able to:

- Create a custom resource file
- Define a custom resource action
- Extract Chef resources into a custom resource action implementation
- Create custom resource properties

After completing this module you should be able to: create a custom resource file; define a custom resource action; and extract Chef resources into a custom resource action implementation.

EXERCISE



Review the Cookbook

We need to make sure everything works before we get started.

Objective:

- ☐ Checkout different branch of cookbook
- ☐ Review and execute the integration tests
- ☐ Review and execute the unit tests
- ☐ Review the default recipe

Before we begin creating this custom resource it is important to review the cookbook. We will start looking at the integration tests defined.

Move in the apache cookbook



```
> cd ~/apache
```

Add the Changed Files to Staging

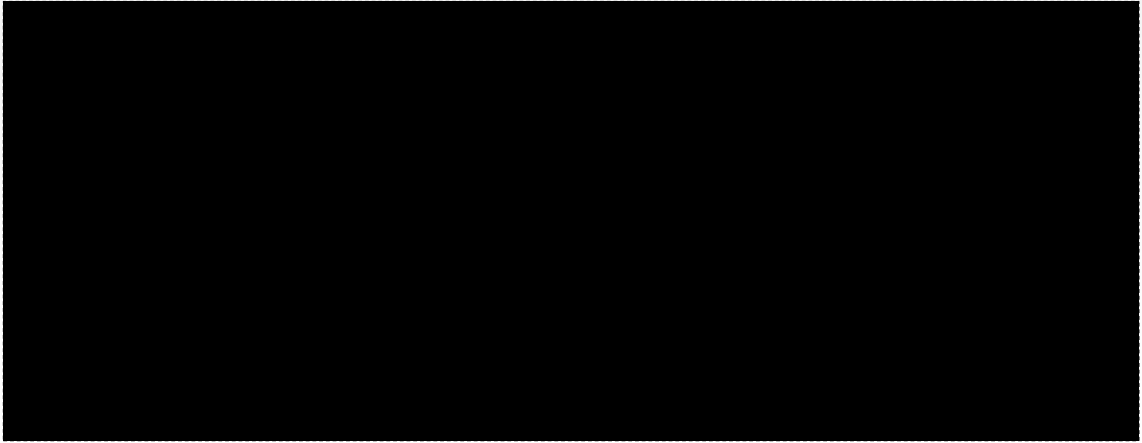


```
> git add .
```

Commit the Staged Changes



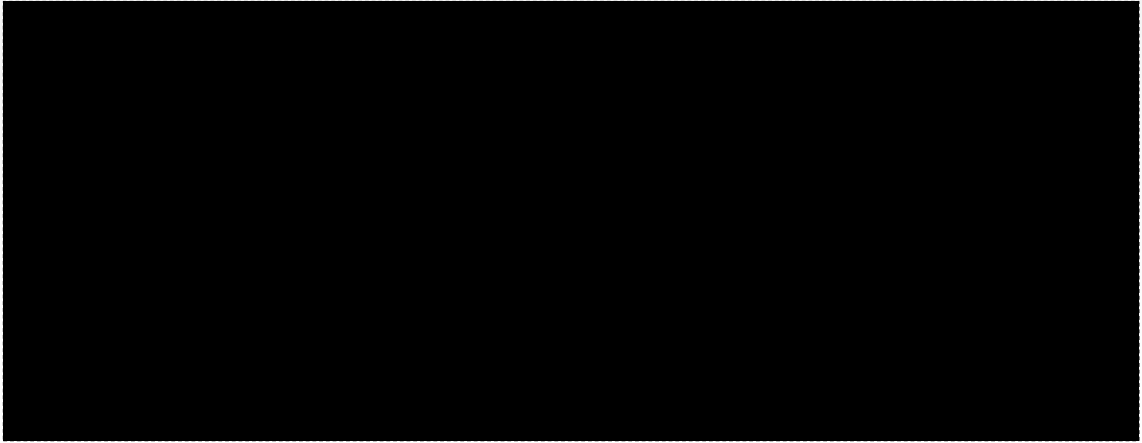
```
> git commit -m "Saving TDD work"
```



Change Git Branches



```
> git checkout extending-cookbook
```



EXERCISE



Review the Cookbook

We need to make sure everything works before we get started.

Objective:

- ✓ Checkout different branch of cookbook
- ☐ Review and execute the integration tests
- ☐ Review and execute the unit tests
- ☐ Review the default recipe

Before we begin creating this custom resource it is important to review the cookbook. We will start looking at the integration tests defined.

Reviewing the Existing Integration Tests



```
~/apache/test/smoke/default/default_test.rb
```

```
describe command('curl http://localhost') do
  its(:stdout) { should match(/Welcome home/) }
end

describe command('curl http://localhost:8080') do
  its(:stdout) { should match(/Welcome admins/) }
end
```

There are two tests define that assert that the a default website is available on port 80 and a second website available on port 8080. Each of these websites cater to the different possible roles one could have with the website. The standard user visits the sit on port 80 where admins visit the site on port 8080.

Executing the Existing Integration Tests



```
> kitchen verify
```

```
Target:  ssh://vagrant@127.0.0.1:2222
```

```
✓ Command curl http://localhost stdout should match /Welcome  
home/
```

```
✓ Command curl http://localhost:8080 stdout should match  
/Welcome admins/
```

```
Summary: 2 successful, 0 failures, 0 skipped
```

```
Finished verifying <default-centos-67> (0m0.74s).
```

```
-----> Kitchen is finished. (0m7.37s)
```

Before refactoring the cookbook it is important that you verify that the cookbook is in a known good state. To do that you would want to use Test Kitchen to execute the two test are defined.

Each example should pass without failure.

EXERCISE



Review the Cookbook

We need to make sure everything works before we get started.

Objective:

- ✓ Checkout different branch of cookbook
- ✓ Review and execute the integration tests
- ☐ Review and execute the unit tests
- ☐ Review the default recipe

Let's examine the unit tests that are defined within the cookbook.

Reviewing the Existing Unit Tests



~/apache/spec/unit/recipes/default_spec.rb

```
require 'spec_helper'

describe 'apache::default' do
  context 'When all attributes are default, on an unspecified platform' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end
  end

  # ... CONTINUES ON THE NEXT SLIDE ...
end
```

There is a single specification file defined for the default recipe. The first expectation defined is the generic one that assures us that the chef run should converge without raising an error.

Reviewing the Existing Unit Tests



~/apache/spec/unit/recipes/default_spec.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...

it 'installs the necessary package' do
  expect(chef_run).to install_package('httpd')
end

it 'starts the necessary service' do
  expect(chef_run).to start_service('httpd')
end

it 'enables the necessary service' do
  expect(chef_run).to enable_service('httpd')
end

# ... CONTINUES ON THE NEXT SLIDE ...
```

The next few expectations ensure that the necessary packages are installed and the services are started and enabled.

Reviewing the Existing Unit Tests



~/apache/spec/unit/recipes/default_spec.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...

describe 'for the default site' do
  it 'writes out a new home page' do
    expect(chef_run).to render_file('/var/www/html/index.html').with_content('<h1>Welcome
home!</h1>')
  end
end

# ... CONTINUES ON THE NEXT SLIDE ...
```

The next expectation ensures that the default site has an html page that is written out and contains a small amount of content that we assume should be present within that file to ensure our guests are welcome to the site.

Reviewing the Existing Unit Tests



~/apache/spec/unit/recipes/default_spec.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...  
describe 'for the admin site' do  
  it 'creates the directory' do  
    expect(chef_run).to create_directory('/srv/httpd/admins/html')  
  end  
  
  it 'creates the configuration' do  
    expect(chef_run).to render_file('/etc/httpd/conf.d/admins.conf')  
  end  
  
  it 'creates a new home page' do  
    expect(chef_run).to render_file('/srv/httpd/admins/html/index.html').with('<h1...h1>')  
  end  
# ... CONTINUES ON THE NEXT SLIDE ...
```

For the admin site we ensure that the site directory is created, a configuration file is written, and that the home page displays a welcoming message to the admins visiting the site.

Reviewing the Existing Unit Tests



~/apache/spec/unit/recipes/default_spec.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...  
    end # describe admin site  
  end # context  
end # describe 'apache::default'
```

This is the end of the file showing the remaining 'end' keywords necessary to properly close the blocks that were opened (with the do keyword) above. Comments follow each one to show their matching 'do' in the file above.

Executing the Existing Unit Tests



```
> chef exec rspec
```

```
.....
```

```
Finished in 0.92866 seconds (files took 2.76 seconds to load)  
8 examples, 0 failures
```

After reviewing the expectations it is important to execute them to ensure that all of them pass.

EXERCISE



Review the Cookbook

We need to make sure everything works before we get started.

Objective:

- ✓ Checkout different branch of cookbook
- ✓ Review and execute the integration tests
- ✓ Review and execute the unit tests
- ☐ Review the default recipe

Finally it is time to review the default recipe.

Reviewing the Default Recipe



~/apache/recipes/default.rb

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2017 The Authors, All Rights Reserved.
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

# ... CONTINUES ON THE NEXT SLIDE ...
```

First we see that the recipe installs the necessary packages to install the web server. An html page is written out for the default site to contain the appropriate welcome message.

Reviewing the Default Recipe



~/apache/recipes/default.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...
directory '/srv/httpd/admins/html' do
  recursive true
  mode '0755'
end

template '/etc/httpd/conf.d/admins.conf' do
  source 'conf.erb'
  mode '0644'
  variables(document_root: '/srv/httpd/admins/html', port: 8080)
  notifies :restart, 'service[httpd]'
end

file '/srv/httpd/admins/html/index.html' do
  content '<h1>Welcome admins!</h1>'
end
# ... CONTINUES ON THE NEXT SLIDE ...
```

The next three resources setup the admin site. First creating the directory for the admin site to store the html it will display. A configuration file is written to ensure the webserver will find the new site that we have defined. Last an index html file is added to the admin site with a welcoming message.

Reviewing the Default Recipe



~/apache/recipes/default.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...
```

```
service 'httpd' do
  action [:enable, :start]
end
```

For the webserver to work correctly with the default site and the admin site the service needs to be started. We also enable the service to ensure the web server will start again if the instance this is being executed on happens to reboot.

EXERCISE



Review the Cookbook

We need to make sure everything works before we get started.

Objective:

- ✓ Checkout different branch of cookbook
- ✓ Review and execute the integration tests
- ✓ Review and execute the unit tests
- ✓ Review the default recipe

Reviewing the integration tests, unit tests, and recipe gives us a good understanding of what this cookbook accomplishes.

EXERCISE



Creating a Custom Resource

This will make our recipe much cleaner.

Objective:

- ☐ Create a custom resource that create the admin site
- ☐ Allow the custom resource to have configurable properties

With a working cookbook it is time to refactor it to use custom resources. A custom resource will help make the recipe we define express our intentions more clearly and allow us to hide some of the implementation details that make it harder for us to at-a-glance understand what a recipe is accomplishing. It will also assist us if we wanted to support multiple different sites for other roles that have yet been defined.

Generating a Custom Resource



```
> chef generate lwrp vhost
```

```
Compiling Cookbooks...
```

```
Recipe: code_generator::lwrp
```

```
  * directory[/home/chef/apache/resources] action create
    - create new directory /home/chef/apache/resources
  * template[/home/chef/apache/resources/vhost.rb] action create
    - create new file ~/httpd/resources/vhost.rb
    - update content in file /home/chef/apache/resources/vhost.rb from none to e3b0c4
      (diff output suppressed by config)
  * directory[/home/chef/apache/providers] action create
```

The chef command-line tool allows you to generate some initial directories and resource file. While we are developing a Custom Resource the former name for them was called Light Weight Resource Provider or LWRP. The chef command still uses the acronym lwrp as the generate sub-command.

We call these multiple different sites, available on different ports, a virtual host. This is often abbreviated as 'vhost'. Create a custom resource with the name 'vhost'.

Removing an un-needed Directory



```
> rm -rf providers
```

A LWRP (light-weight resource provider) requires two directories. A resources directory and a provider directory. The custom resource implementation requires only the resources directory.

The providers directory is not needed so it should be removed.

Defining the Create Action

 `~/apache/resources/vhost.rb`

```
action :create do  
  
end
```

Within the resources directory a file named 'vhost' should exist. Within it we are simply going to define an action with the name `:create`. This create action is where we will define the resources necessary to create a new vhost.

Implementing the Create Action



~/apache/resources/vhost.rb

```
action :create do
  directory '/srv/httpd/admins/html' do
    recursive true
    mode '0755'
  end

  template '/etc/httpd/conf.d/admins.conf' do
    source 'conf.erb'
    mode '0644'
    variables(document_root: '/srv/httpd/admins/html', port: 8080)
    notifies :restart, 'service[httpd]'
  end

  file '/srv/apache/admins/html/index.html' do
    content '<h1>Welcome admins!</h1>'
  end
end
```

To create a new virtual host we need to generate a directory, add a configuration file, and define an html file. This is similar to the exact same resources that we defined for the admin site in the default recipe.

Our first implementation for our custom resource will create the exact same admin site exactly as it is done in the default recipe. These values are hard-coded to the admin site which we will address after getting our implementation working.

Refactoring the Default Recipe



~/apache/recipes/default.rb

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2017 The Authors, All Rights Reserved.
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

directory '/srv/httpd/admins/html' do
  recursive true
  mode '0755'
end

template '/etc/httpd/conf.d/admins.conf' do
```

Now that those three resources are defined within the custom resource we want to use it within our recipe. We can now remove the use of these three resources within the default recipe.

Remove the directory resource, the template resource, and the file resource that generate the admin site.

Refactoring the Default Recipe



~/apache/recipes/default.rb

```
template '/etc/httpd/conf.d/admins.conf' do
  source 'conf.erb'
  mode '0644'
  variables(
    document_root: '/srv/httpd/admins/html',
    port: 8080
  )
  notifies :restart, 'service[httpd]'
end

file '/srv/httpd/admins/html/index.html' do
  content '<h1>Welcome admins!</h1>'
end

service 'httpd' do
  action [:enable, :start]
end
```

Remove the directory resource, the template resource, and the file resource that generate the admin site.

Adding the New Custom Resource



~/apache/recipes/default.rb

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

apache_vhost 'admins' do
  action :create
end

service 'httpd' do
  action [:enable, :start]
end
```

Now we can insert the custom resource that is create for us. The full name of the custom resource comes from the name of the cookbook joined with an underscore to the name of the ruby file defined within the resources directory.

In this instance the cookbook's name is 'apache' and the ruby file is named 'vhost' so the default name for the resource is 'apache_vhost'. We inform the resource that we want to generate the site for admins, though the name of the resource is not used in any way in our definition. We explicitly state that the resource will use the create action.

Executing the Unit Tests



```
> chef exec rspec
```

```
Finished in 0.9673 seconds (files took 2.72 seconds to load)
```

```
8 examples, 3 failures
```

```
Failed examples:
```

```
rspec ./spec/unit/recipes/default_spec.rb:39 # apache::default When all attributes are default, on an unspecified platform for the admin site creates the directory
```

```
rspec ./spec/unit/recipes/default_spec.rb:43 # apache::default When all attributes are default, on an unspecified platform for the admin site creates the configuration
```

```
rspec ./spec/unit/recipes/default_spec.rb:47 # apache::default When all attributes are default, on an unspecified platform for the admin site creates a new home page
```

With the custom resource defined now within the default recipe it is time to run our unit tests to ensure that we have not broken our implementation.

When executing the tests you will see three failures. These three failures will instruct you that it does not see the following resources created: the directory for the admin site; the configuration file built from the template; and the html file.

This does not seem right. The resources defined within the custom resource do just that.

CONCEPT

Resource Collection

> rspec

●

other resources ...

●

template '...'

●

directory '...'

●

file '...'

●

other resources ...


This resource is missing from the resource collection.

This resource is missing from the resource collection.

This resource is missing from the resource collection.

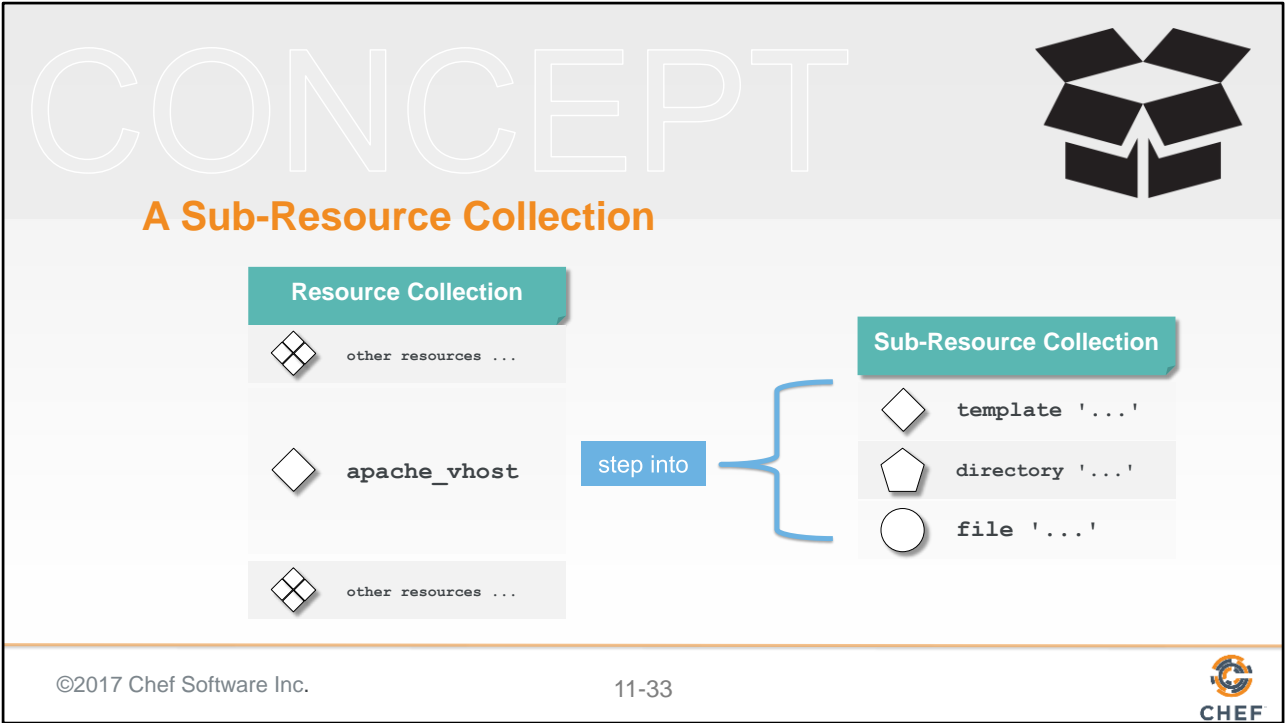
©2017 Chef Software Inc.

11-32



Remember the ChefSpec expectations are validating the contents of the state of the resource collection.

When we created this custom resource we moved the three resources within the recipe into the action we defined. This changed the state of the resource collection and caused the failures we see when we execute the test suite.



©2017 Chef Software Inc.

11-33



This custom resource created a resource collection within our resource collection; a sub-resource collection. ChefSpec by default does not step into this sub-resource collection. We can however enable that behavior if we modify our test setup to explicitly state we are interested in evaluating the contents of this sub-resource collection.

We will discuss more about the implications of having a sub-resource collection in the follow-up module.

Updating the Unit Tests to Verify Custom Resource



~/apache/spec/unit/recipes/default_spec.rb

```
require 'spec_helper'

describe 'apache::default' do
  context 'When all attributes are default, on an unspecified platform' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(step_into: ['apache_vhost'])
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    # ... CONTINUES ON THE NEXT SLIDE ...
  end
end
```

The unit tests fail because the resources defined within the custom resource are now no longer placed onto the resource collection. This is because the custom resource is placed on the resource collection and the resources internally within it are placed on a secondary resource collection that the custom resource owns.

To ask our unit tests to verify the resources defined within our custom resource we need to explicitly ask the ChefSpec runner to step into the resource and examine the resources it uses to accomplish it's work.

Executing the Unit Tests



```
> chef exec rspec
```

```
.....
```

```
Finished in 0.98788 seconds (files took 2.95 seconds to load)
```

```
8 examples, 0 failures
```

Running the unit tests again should show all the expectations have been met.

Converging the Test Instance



```
> kitchen converge
```

```
(up to date)
(up to date)
  (up to date)
(up to date)
* service[apache] action start (up to date)
```

```
Running handlers:
```

```
Running handlers complete
```

```
Chef Client finished, 0/8 resources updated in 05 seconds
```

```
Finished converging <default-centos-67> (0m8.81s).
```

```
-----> Kitchen is finished. (0m9.80s)
```

It is also important to execute the integration tests defined. First converging the test instance to ensure the recipe is defined correctly and converges successfully.

Verifying the Test Instance



```
> kitchen verify
```

```
-----> Starting Kitchen (v1.11.1)
-----> Setting up <default-centos-67>...
        Finished setting up <default-centos-67> (0m0.00s).
-----> Verifying <default-centos-67>...
        Use `/home/chef/apache/test/smoke/default` for testing

Target:  ssh://vagrant@127.0.0.1:2222
```

- ✓ Command curl http://localhost stdout should match /Welcome home/
- ✓ Command curl http://localhost:8080 stdout should match /Welcome admins/

```
Summary: 2 successful, 0 failures, 0 skipped
        Finished verifying <default-centos-67> (0m0.70s).
-----> Kitchen is finished. (0m1.82s)
```

And finally we verify that the state of the system is still hosting our two sites.

Run a Complete Test on the Test Instance



```
> kitchen test
```

```
Target:  ssh://vagrant@127.0.0.1:2222
```

- ✓ Command curl http://localhost stdout should match /Welcome home/
- ✓ Command curl http://localhost:8080 stdout should match /Welcome admins/

```
Summary: 2 successful, 0 failures, 0 skipped
  Finished verifying <default-centos-67> (0m0.70s) .
----> Destroying <default-centos-67>...
  ==> default: Forcing shutdown of VM...
  ==> default: Destroying VM and associated drives...
  Vagrant instance <default-centos-67> destroyed.
  Finished destroying <default-centos-67> (0m4.09s) .
  Finished testing <default-centos-67> (2m21.40s) .
----> Kitchen is finished. (2m22.49s)
```

We made changes to the recipe and then used kitchen to converge this recipe against the test instance. This ensured that our recipe will successfully converge against a system that has already been configured and not raise any errors.

However, we still need to ensure that the recipe will converge successfully on a brand new instance so it is important to ask Test Kitchen to destroy the instance, converge a new instance, and verify the results. This can be done with the 'kitchen test' command.

EXERCISE



Creating a Custom Resource

This will make our recipe much cleaner.

Objective:

- ✓ Create a custom resource that create the admin site
- ☐ Allow the custom resource to have configurable properties

The initial implementation of the custom resource has been created and we have verified that it works by running our two test suites.

Now it is time to address the problem with the implementation having hard-coded values specific to the admin site. We want to make it more generic so that it can deploy a different, custom site for us if needed.

This can be done through properties that you defined on the custom resource.



CONCEPT

Resource Properties

A property is defined with the following syntax.

```
~/apache/resources/vhost.rb
```

```
property :site_name, String
```

1

2

property method

1
name (symbol)

2
type

Optional Parameters

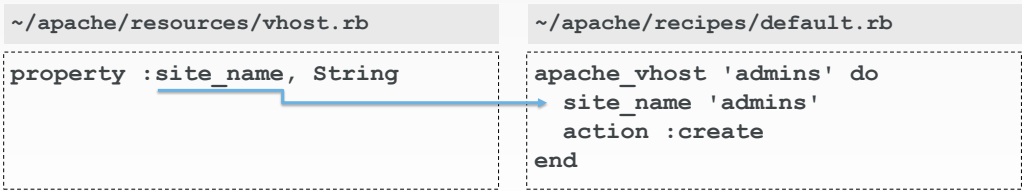
Properties are defined in the same file as you define the resource actions. Generally these are defined at the top of the file to make them immediately visible. A property is defined by specifying a method named `property` with two required parameters and a third set of optional parameters. The name of the property is defined as a Ruby Symbol. The type is a Ruby class name. This type enforces what kind of values are supported by this property; typically it is a `String` for text and a `Fixnum` for numbers. The optional parameters are defined as a Hash. We will explore defining a property with these parameters in the next module.

CONCEPT



Resource Properties

Properties are defined in the resource definition and then available within definition of the resource within the recipe.



The property that you define within the custom resource definition becomes part of how you can describe the resource within the recipe.

Defining a Property to Manage the site_name



~/apache/resources/vhost.rb

```
property :site_name, String

action :create do

  directory "/srv/httpd/#{site_name}/html" do
    recursive true
    mode '0755'
  end

  template "/etc/httpd/conf.d/#{site_name}.conf" do
    source 'conf.erb'
    mode '0644'
    variables(document_root: "/srv/httpd/#{site_name}/html", port: 8080)
    notifies :restart, 'service[httpd]'
  end

  # ... CONTINUES ON THE NEXT SLIDE ...
```

Let's start by defining a property named 'site_name' that will contain the name of the site we want to create. The name of the site will be used to create the directory for our index page, the configuration file details, and the message we send out to the visitor.

The 'site_name' is going to be a text so we specify the type as String.

Updating the Action to use the Property



```
~/apache/resource/vhost.rb
```

```
action :create do
  directory "/srv/httpd/#{site_name}/html" do
    recursive true
    mode '0755'
  end

  template "/etc/httpd/conf.d/#{site_name}.conf" do
    source 'conf.erb'
    mode '0644'
    variables(document_root: "/srv/httpd/#{site_name}/html", port: 8080)
    notifies :restart, "service[httpd]"
  end

  file "/srv/httpd/#{site_name}/html/index.html" do
    content "<h1>Welcome #{site_name}</h1>"
  end
end
```

Within the action implementation we want to remove the mention of 'admin' and replace it with the value found within the 'site_name' custom property. A resource property creates a method with the same name as the property.

Now we need to replace the 'admin' text with the result of the property. This requires us to update a number of our resources to use String interpolation to express the directory created, the configuration path, and then default html page.

Updating the Resource to use the Property



~/apache/recipes/default.rb

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

apache_vhost 'admins' do
  site_name 'admins'
  action :create
end

service 'httpd' do
  action [:enable, :start]
end
```

This property is not automatically defined and does not contain a default value so we must add this property to the custom resource implementation with the default recipe. In this case we are adding 'site_name' and specifying the value is 'admins'.

This means our implementation should be exactly the same as before but the details are now configurable through this property instead of being hard-coded to 'admin'.

Executing the Unit Tests



```
> chef exec rspec
```

```
.....
```

```
Finished in 1.22 seconds (files took 7.58 seconds to load)
```

```
8 examples, 0 failures
```

The unit tests should pass when executed.

Converging and Verifying the Test Instance



```
> kitchen converge && kitchen verify
```

```
-----> Verifying <default-centos-67>...  
       Use `/home/chef/apache/test/smoke/default` for testing
```

```
Target:  ssh://vagrant@127.0.0.1:2222
```

- ✓ Command curl http://localhost stdout should match /Welcome home/
- ✓ Command curl http://localhost:8080 stdout should match /Welcome admins/

```
Summary: 2 successful, 0 failures, 0 skipped  
       Finished verifying <default-centos-67> (0m0.77s).  
-----> Kitchen is finished. (2m42.37s)
```

The integration tests should pass when executed.

EXERCISE



Creating a Custom Resource

That's much better.

Objective:

- ✓ Create a custom resource that create the admin site
- ✓ Allow the custom resource to have configurable properties

We now have a custom resource implementation that has helped express our intentions more clearly in the default recipe.

LAB



apache_vhost - site_port Property

- ❑ Create a custom resource property named **site_port** that is a *Fixnum*
- ❑ Within the apache_vhost's create action replace the hard-coded value 8080 with the **site_port** property
- ❑ Within the default recipe set the apache_vhost resource named 'admins' to have a site_port 8080

The custom resource still needs a little more work to make it configurable. The create action is still hard-coded to specify the port 8080 for all sites that are created.

During this exercise you will define a new property within the custom resource that allows a port to be specified for the site. Replace any hard-coded port values within the resource action implementation and then add the new property to the implementation of the custom resource in the default recipe.

Instructor Note: Allow 10 minutes to complete this exercise

Defining a Property to Manage the site_port



~/apache/resource/vhost.rb

```
property :site_name, String
property :site_port, Fixnum

action :create do
  directory "/srv/httpd/#{site_name}/html" do
    recursive true
    mode '0755'
  end

  template "/etc/httpd/conf.d/#{site_name}.conf" do
    source 'conf.erb'
    mode '0644'
    variables(document_root: "/srv/httpd/#{site_name}/html", port: site_port)
    notifies :restart, 'service[httpd]'
  end
end
# ... REMAINDER OF CUSTOM RESOURCE ...
```

The property is defined near the top of the resource file. A port is generally a whole number so we want that reflected in the type.

A Fixnum can contain negative integers and floating point numbers so this type does not perfectly represent the domain of acceptable values. Later we may explore ways to ensure better restrictions on the values provided to properties.

Within the action implementation the 8080 value should be replaced with the value found in 'site_port'.

Updating the Resource to use the Property



~/apache/recipes/default.rb

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

apache_vhost 'admins' do
  site_name 'admins'
  site_port 8080
  action :create
end

service 'httpd' do
  action [:enable, :start]
end
```

In the default recipe, within the 'apache_vhost' resource, we must define a value for this site_port. Similar to before we simply define the value that was previously hard-coded here as a property.

Executing the Unit Tests



```
> chef exec rspec
```

```
.....
```

```
Finished in 1.22 seconds (files took 7.58 seconds to load)
```

```
8 examples, 0 failures
```

The unit tests should pass when executed.

Converging and Verifying the Test Instance



```
> kitchen converge && kitchen verify
```

```
-----> Verifying <default-centos-67>...  
       Use `/home/chef/apache/test/smoke/default` for testing
```

```
Target:  ssh://vagrant@127.0.0.1:2222
```

- ✓ Command curl http://localhost stdout should match /Welcome home/
- ✓ Command curl http://localhost:8080 stdout should match /Welcome admins/

```
Summary: 2 successful, 0 failures, 0 skipped  
       Finished verifying <default-centos-67> (0m0.77s).  
-----> Kitchen is finished. (2m42.37s)
```

The integration tests should pass when executed.

LAB



apache_vhost - site_port Property

- ✓ Create a custom resource property named **site_port** that is a *Fixnum*
- ✓ Within the apache_vhost's create action replace the hard-coded value 8080 with the **site_port** property
- ✓ Within the default recipe set the apache_vhost resource named 'admins' to have a site_port 8080

With the site_port property developed the 'apache_vhost' custom resource is now capable of being used to create more sites if needed for different roles on different ports for our web server.

PROBLEM



Remove the Welcome Site

When apache installs itself it defines a default site on port 80. Up until this point we have relied on this site. We now want to remove that initial welcome site but we want to do that we a new action we will define on the custom resource we are defining.

By default Apache creates a welcome configuration file within the same directory we are creating our new virtual hosts. We want to delete this configuration file but we want to create a resource that will also cleanup any html files that our resource might create as well. This will allow us to create and remove sites as we want.

LAB



apache_vhost Remove Action

- ❑ Define the 'remove' action for apache_vhost that defines the policy:

A directory named `/srv/httpd/#{site_name}/html` is deleted

A file named `/etc/httpd/conf.d/#{site_name}.conf` is deleted

- ❑ Update the default recipe's policy to include a new resource:

An `apache_vhost` resource named 'welcome' is removed

This next lab exercise challenges you to create the remove action for the custom resource, use that remove action to remove the default site that ships with the webserver, and deploy a new site instead which welcomes users.

Instructor Note: Allow 15 minutes to complete this exercise

Defining the Resource's Remove Action



~/apache/resources/vhost.rb

```
# ... CREATE ACTION ...  
  
action :remove do  
  directory "/srv/httpd/#{site_name}/html" do  
    action :delete  
  end  
  
  file "/etc/httpd/conf.d/#{site_name}.conf" do  
    action :delete  
  end  
end
```

The remove action asks that you remove the directory that may or may not exist at the location dependent on the `site_name` provided as a property. We also want it to remove the configuration file from the webserver's default configuration directory.

Adding the Resource with Remove Action to the Recipe

 ~/apache/recipes/default.rb

```
package 'httpd'

apache_vhost 'welcome' do
  site_name 'welcome'
  action :remove
end

apache_vhost 'admins' do
  site_port 8080
  site_name 'admins'
  action :create
end

service 'httpd' do
  action [:enable, :start]
end
```

When apache initial sets itself up it deploys the first, default site, with a welcome configuration file that we want to remove. While the 'welcome' directory does not exist the configuration file does and so we want that removed from the system.

This will ensure the default site that is deployed on port 80 is no longer deployed.

LAB



apache_vhost Remove Action

- ✓ Define the 'remove' action for apache_vhost that defines the policy:

A directory named `"/srv/httpd/#{site_name}/html"` is deleted

A file named `"/etc/httpd/conf.d/#{site_name}.conf"` is deleted

- ✓ Update the default recipe's policy to include a new resource:

An `apache_vhost` resource named 'welcome' is removed

The resource now has two actions and we have removed the initial welcome site.

LAB



apache_vhost Remove Action

- ☐ Update the default recipe's policy:

Remove the file resource named '/var/www/html/index.html'

Add an apache_vhost resource named 'users' is created with the site_port 80

- ☐ Update the ChefSpec tests to stop expecting the file resource and start expecting the new resources found within the apache_vhost resource named 'users'
- ☐ Update the InSpec tests to expect the default site to "Welcome users!"

This next lab exercise challenges you to create the remove action for the custom resource, use that remove action to remove the default site that ships with the webserver, and deploy a new site instead which welcomes users.

Instructor Note: Allow 15 minutes to complete this exercise

Removing the Resource from the Recipe

 ~/apache/recipes/default.rb

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

apache_vhost 'admins' do
  site_port 8080
  site_name 'admins'
  action :create
end

service 'httpd' do
  action [:enable, :start]
end
```

The file resource within the default recipe modifies a generic files that apache deploys. Manipulating this resource is no longer important so we want to remove this resource.

Adding the Resource to create the users site



~/apache/recipes/default.rb

```
apache_vhost 'welcome' do
  site_name 'welcome'
  action :remove
end

apache_vhost 'users' do
  site_port 80
  site_name 'users'
  action :create
end

apache_vhost 'admins' do
  notifies :restart, 'service[httpd]'
end
```

Now we want to add in our users site with our custom resource. We define a `site_port` and `site_name` to ensure we receive the correct message on the correct port.

Removing the Un-needed Unit Test Expectation



~/apache/spec/unit/recipes/default_spec.rb

```
# ... EXAMPLES DEFINED ABOVE ...  
  
describe 'for the default site' do  
  it 'writes out a new home page' do  
    expect(chef_run).not_to  
render_file('/var/www/html/index.html').with_content('<h1>Welcome home!</h1>')  
    end  
  end  
end  
  
# ... EXAMPLES DEFINED BELOW ...
```

This changes our default expectations that the site will say welcome home so we want to remove that content from our unit test.

Adding Expectations for the users Site



~/apache/spec/unit/recipes/default_spec.rb

```
# ... EXAMPLES DEFINED ABOVE ...

describe 'for the users site' do
  it 'creates the directory' do
    expect(chef_run).to create_directory('/srv/httpd/users/html')
  end

  it 'creates the configuration' do
    expect(chef_run).to render_file('/etc/httpd/conf.d/users.conf')
  end

  it 'creates a new home page' do
    expect(chef_run).to
    render_file('/srv/httpd/users/html/index.html').with_content('<h1>Welcome users!</h1>')
  end
end

# ... EXAMPLES DEFINED ABOVE ...
```

And add a new series of expectations that are very similar to the admins site. We want to ensure that the following are created: a directory to store the html; a configuration file for users; and a new html file that contains a message for the users.

Executing the Unit Test Suite



```
> chef exec rspec
```

```
.....
```

```
Finished in 1.22 seconds (files took 2.56 seconds to load)  
10 examples, 0 failures
```

Executing the unit tests we should see all these brand new expectations passing.

Updating the Expectation for the users Site



```
~/apache/test/smoke/default/default_test.rb
```

```
describe command('curl http://localhost') do
  its(:stdout) { should match(/Welcome users/) }
end

describe command('curl http://localhost:8080') do
  its(:stdout) { should match(/Welcome admins/) }
end
```

Finally we want to change the integration test to verify the message on port 80 to welcome users and not to welcome visitors home.

Converging and Verifying the Test Instance



```
> kitchen converge && kitchen verify
```

```
-----> Verifying <default-centos-67>...  
        Use `/home/chef/apache/test/smoke/default` for testing  
  
Target:  ssh://vagrant@127.0.0.1:2222  
  
✓ Command curl http://localhost stdout should match /Welcome home/  
✓ Command curl http://localhost:8080 stdout should match /Welcome admins/  
  
Summary: 2 successful, 0 failures, 0 skipped  
        Finished verifying <default-centos-67> (0m0.77s).  
-----> Kitchen is finished. (2m42.37s)
```

Executing the integration tests should result in all examples passing successfully.

LAB



apache_vhost Remove Action

- ✓ Update the default recipe's policy:
 - Remove the file resource named '/var/www/html/index.html'
 - Add an apache_vhost resource named 'users' is created with the site_port 80
- ✓ Update the ChefSpec tests to stop expecting the file resource and start expecting the new resources found within the apache_vhost resource named 'users'
- ✓ Update the InSpec tests to expect the default site to "Welcome users!"

The initial file resource has now been removed and we are creating a new apache virtual host on port 80 for our users' site. We have also updated all the expectations to correctly verify the state of the run list. Finally we also updated the tests that were executed on the virtual machine

Congratulations! The custom resource now is able to create sites and remove them. There are still more things to learn about custom resources that we will explore in the next module.

.

DISCUSSION



Discussion


What are the benefits of using a custom resource to manage the virtual hosts? What are the drawbacks of using a custom resource?

What does the resource collection look like when using a custom resource?

Let's finish this module with a discussion. Answer these questions. Remember that the answer "I don't know! That's why I'm here!" is a great answer.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

DISCUSSION




Q&A

What questions can we answer for you?

©2017 Chef Software Inc.

11-69



What questions can we answer for you?

