CHEF™

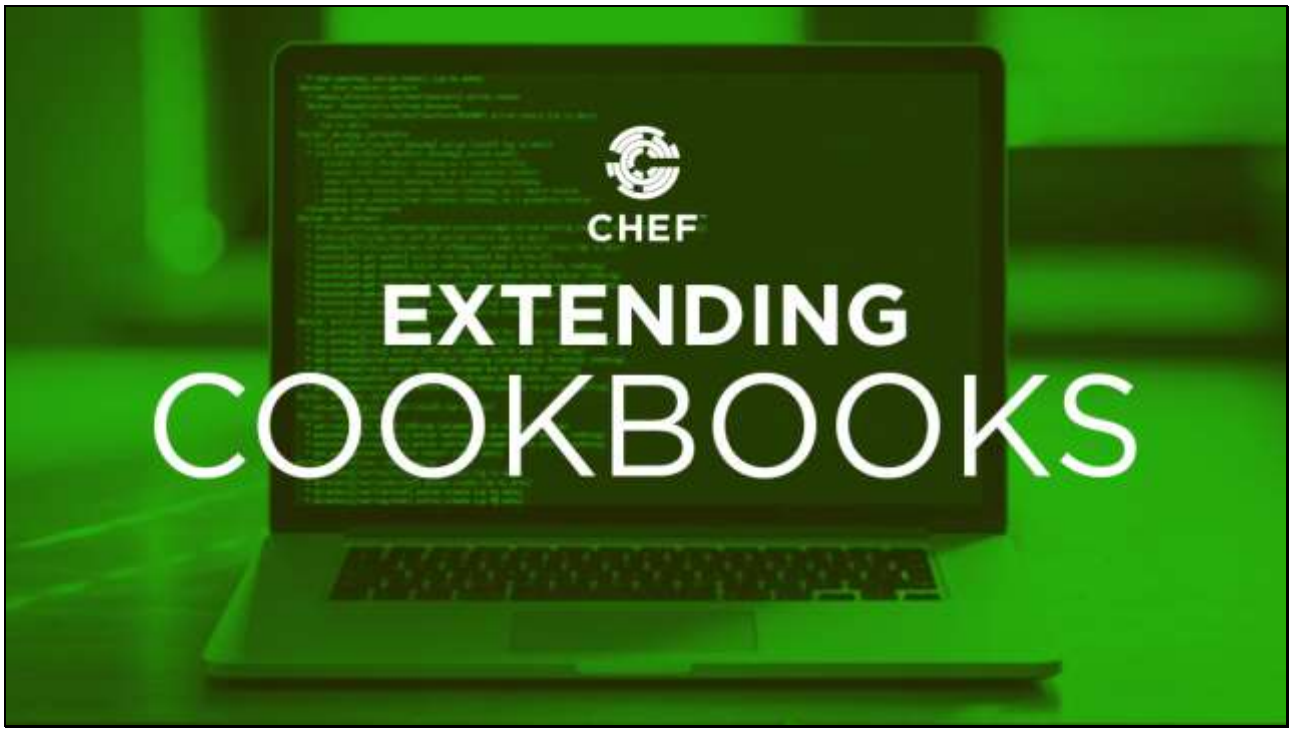
Chef Training Services

Extending Cookbooks

Guide

Course Rev 1.0.0

1: Introduction



Welcome to Extending Cookbooks.

Slide 2

Introduce Yourselves

Name

Current job role

Previous job roles / Background

Experience with Chef

Favorite Text Editor

Before we start let me introduce myself. Then I would like it if everyone had a chance to introduce themselves.

Instructor Note: Often times with larger, in-person groups I prefer to have the individuals perform this introduction one-on-one. Having people leave their desks and greet as many people as they can during the time allotted. I often feel this works better as it removes the pressure from the single individual to introduce themselves in a way that is presenting themselves and not actually greeting people. When Online, I create a pre-defined order, announce that order, and then invite a person to speak, thank them when they are done.

Slide 3

Expectations

You will leave this class with the ability to extend the components of Cookbooks.

You bring with you your own domain expertise and problems. Chef is a framework for solving those problems. Our job is to teach you how to express solutions to your problems with Chef.

The goal of this training is to teach you techniques that will help you extend the functionality of your cookbooks. We also want to share the thought process on why and how to best employ these techniques.

Chef is built on top of Ruby. This means you have the power of a programming language at your disposal and we will have to keep a tight focus on the challenges and exercises presented in this content. During and throughout the content we will have discussion where we may have additional time to talk about many different topics but in this interest of time and popular opinion we may need to leave those discussions.

During the introductions you learned about the other individuals here in the course with you. They may have shared similar problems and domains. During the time that we are here respectfully reach out them so that you can continue the conversation, grow each others' knowledge, and become better professionals.

Slide 4

Expectations

Ask Me Anything: It is important that we answer your questions and set you on the path to be able to find more answers.

Break It: If everything works the first time go back and make some changes. Explore! Discovering the boundaries will help you when you continue on your journey.

All throughout this training I strongly encourage you to ask questions whenever you do not understand a topic, an acronym, concept, or software. By asking a question you better your learning and often times better the learning of those with you in this training. Asking questions is a sign of curiosity that we want to encourage and foster while we are here together.

This curiosity can also be employed by exploring the boundaries of the tools you are using and the language you are writing. The exercises and the labs we will perform will often lead you through examples that work from the beginning to the end. When you develop solutions it is rare that something works from the start all the way to the end. Errors and issues come up from typos or the incorrect usage of a command of the programming language. When you fall off the path it can often be hard to find your way back. Here, if you find yourself always on the correct path explore what happens when you step off of it, what you see, the error messages you are presented with, the new results you might find.

Slide 5

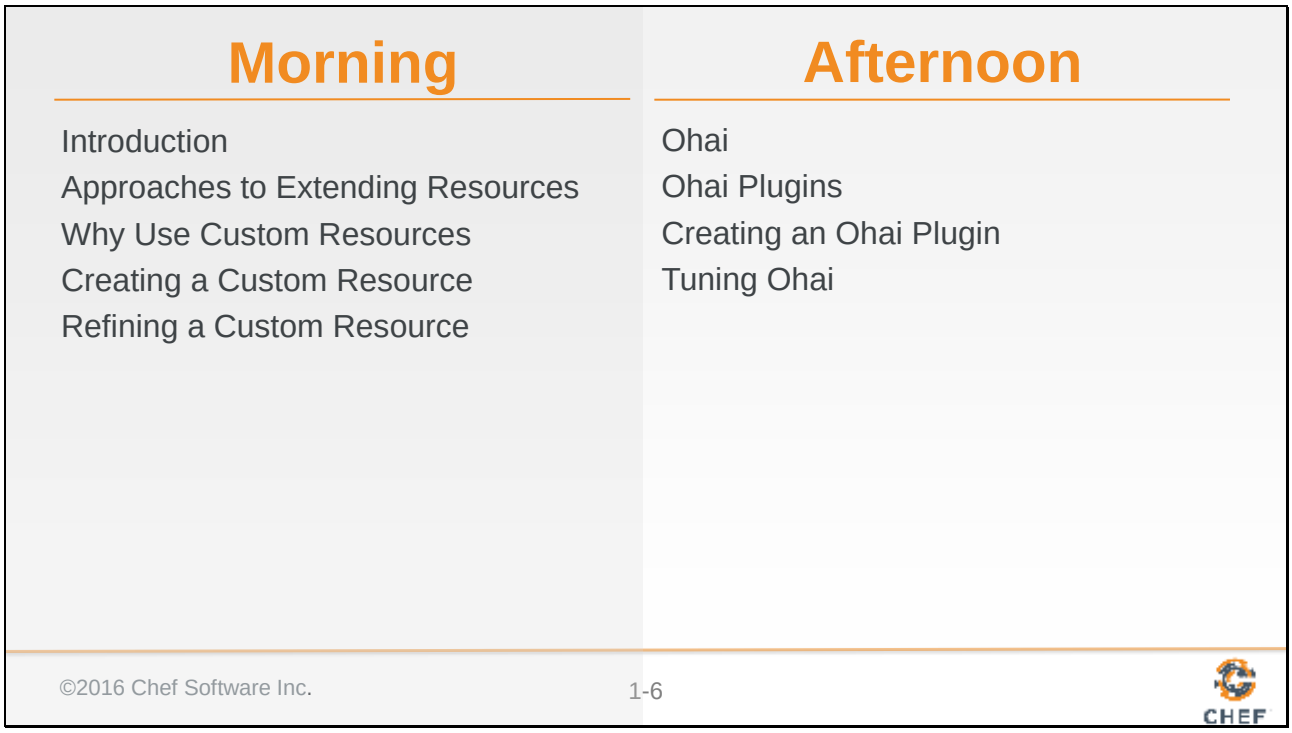
Group Exercises, Labs, and Discussion

This course is designed to be hands on. You will run lots of commands, write lots of code, and express your understanding.

- **Group Exercises:** All participants and the instructor will work through the content together. The instructor will often lead the way and explain things as we proceed.
- **Lab:** You will be asked to perform the task on your own or in groups.
- **Discussion:** As a group we will talk about the concepts introduced and the work that we have completed.

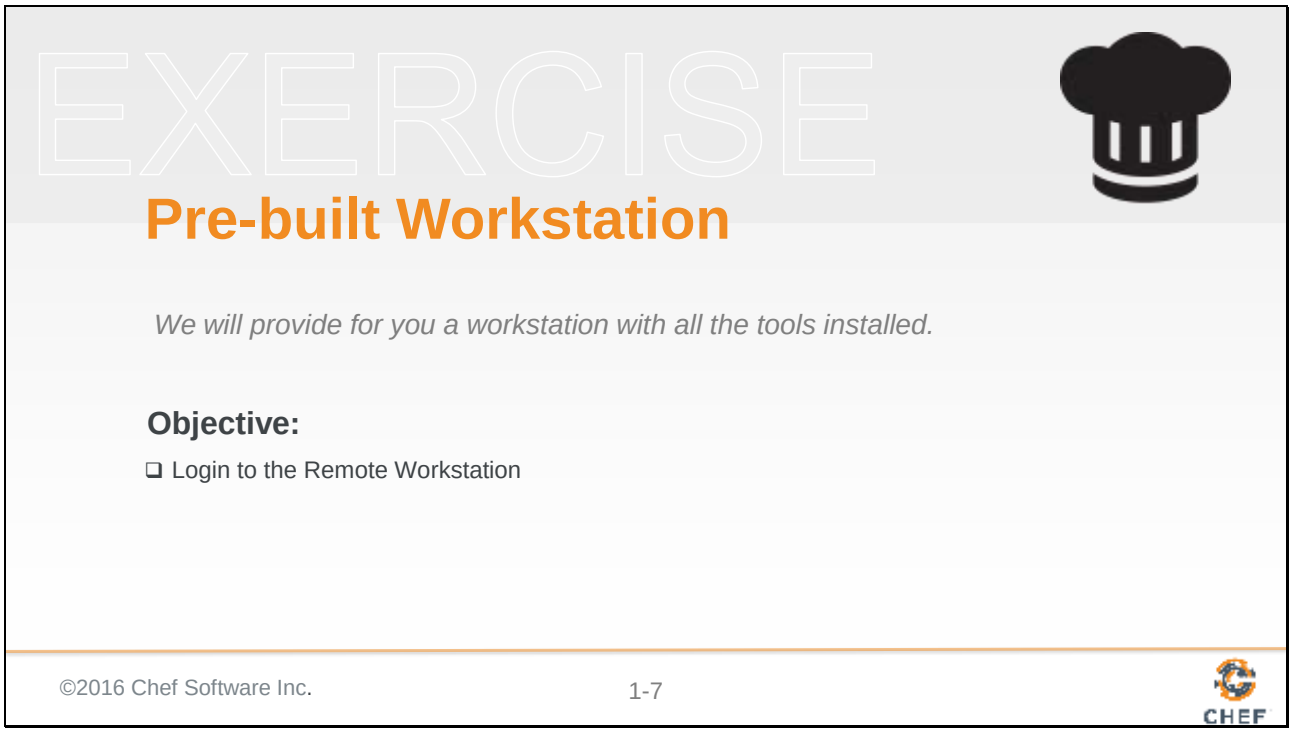
The content of this training has been designed in a way to emphasize this hands-on approach to the content. Together, we will perform exercises together that accomplish an understood objective. After that is done you will often emphasize an activity by performing a lab. The lab is designed to challenge your understanding and retention of the previously accomplished exercises. You can work through this labs on your own or in groups. After completing the labs we will all come together again to review the exercise. Finally, we will end each section with a discussion about the topics that we introduced. These discussions will often ask you to share your opinions, recent experiences, or previous experiences within this domain.

Slide 6



This is the outline of the events for this training. Please take a moment to review this list to ensure that the topics listed here meet your expectations. Take a moment to note which topics are of most interest to you. Also note which topics are not present here on this list. We will discuss your thoughts at the end of the section.

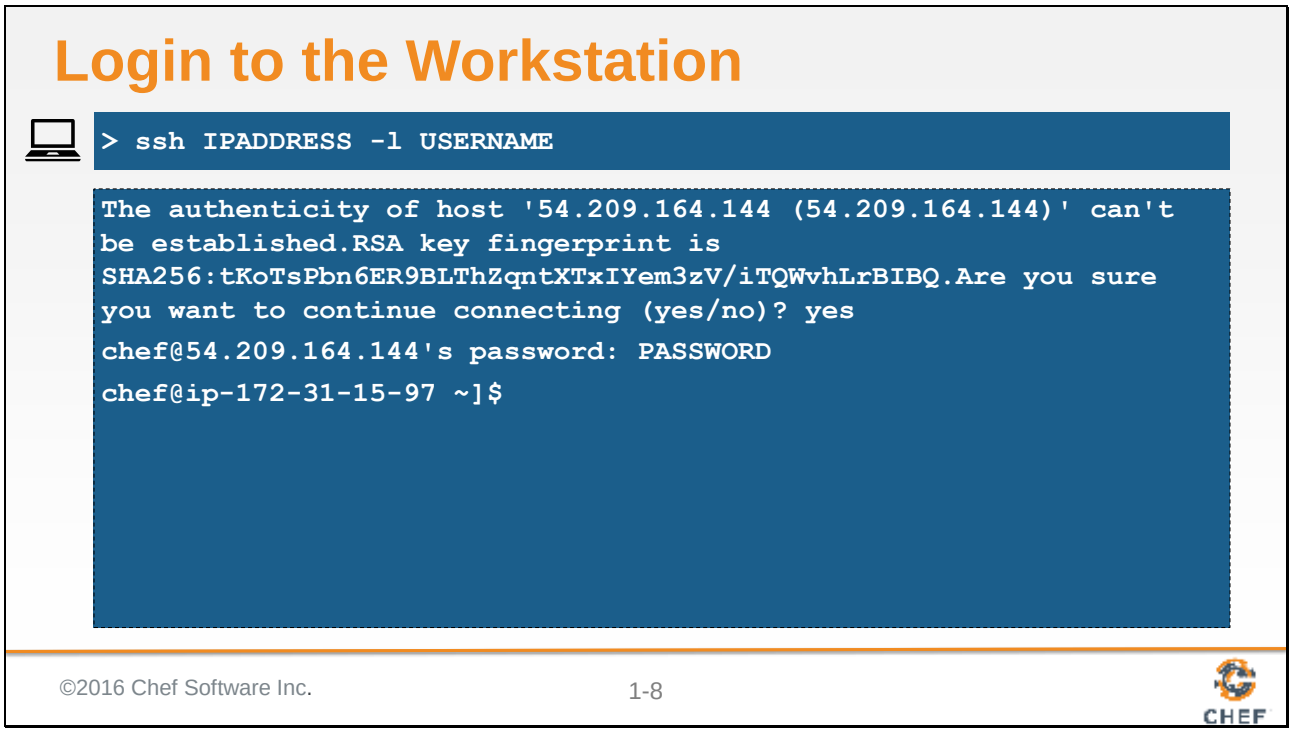
Slide 7



As I mentioned there is a lot work planned for the day. To ensure we focus on the concepts we introduce and not on troubleshooting systems we are providing you a workstation with the necessary tools installed to get started right away.

Instructor Note: At the end of the training it is often a good idea to offer your services to help individuals install necessary software or troubleshoot their systems.

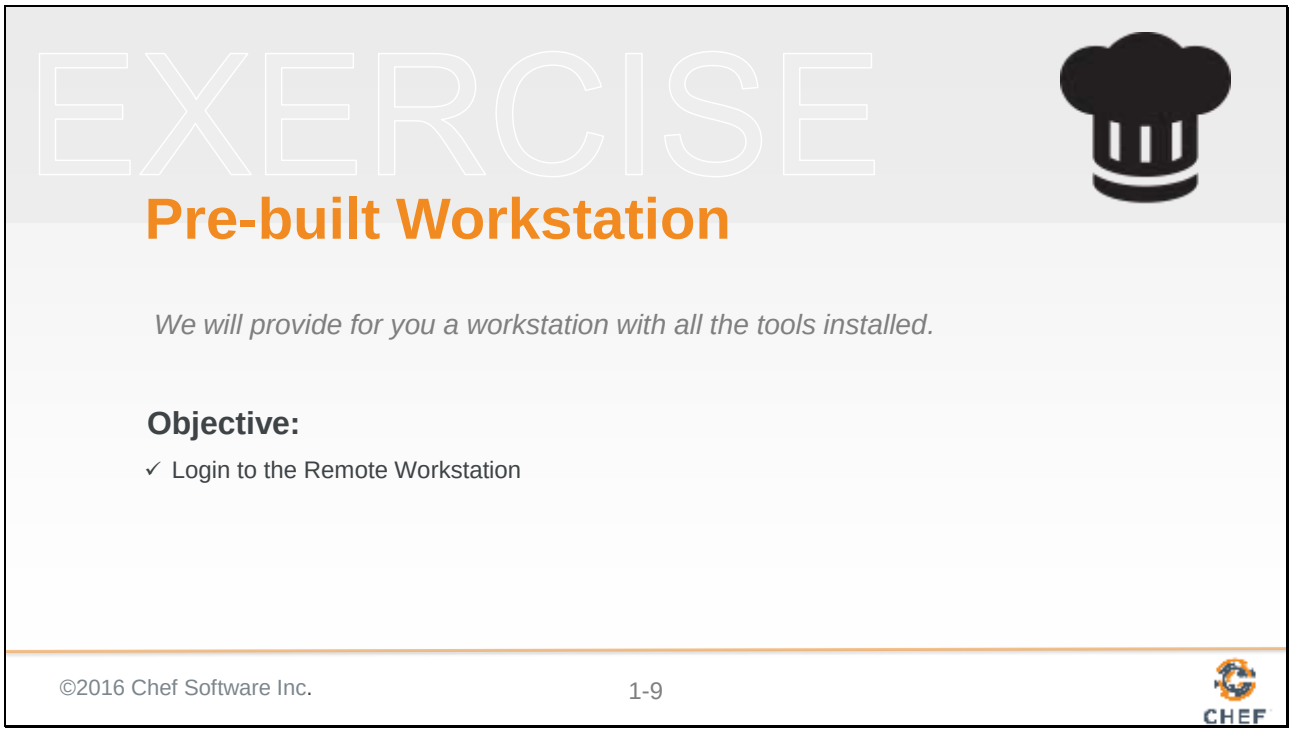
Slide 8



I will provide you with the address, username and password of the workstation. With that information you will need to use the SSH tool that you have installed to connect that workstation.

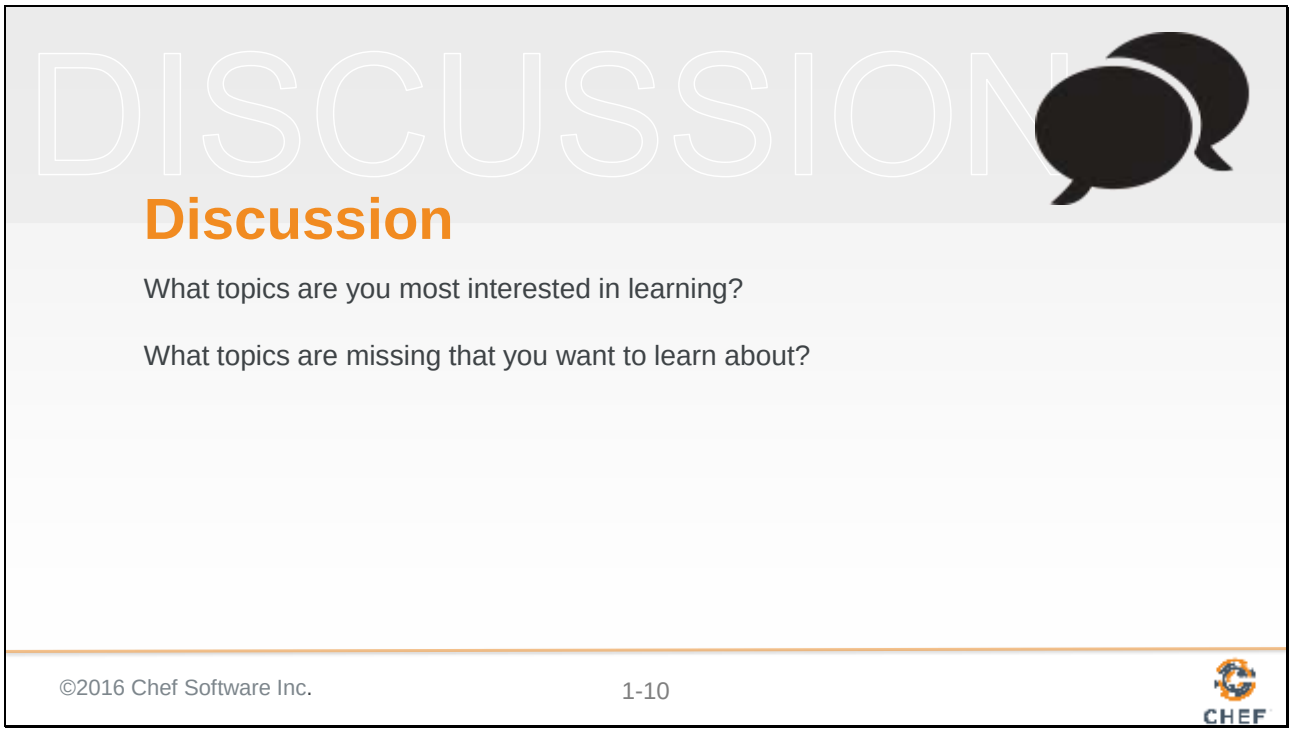
This demonstrates how you might connect to the remote machine using your terminal or command-prompt if you have access to the application ssh. This may be different based on your operating system.

Slide 9



Now that you are connected to that workstation we have taken care of all the necessary work to get started with the training.

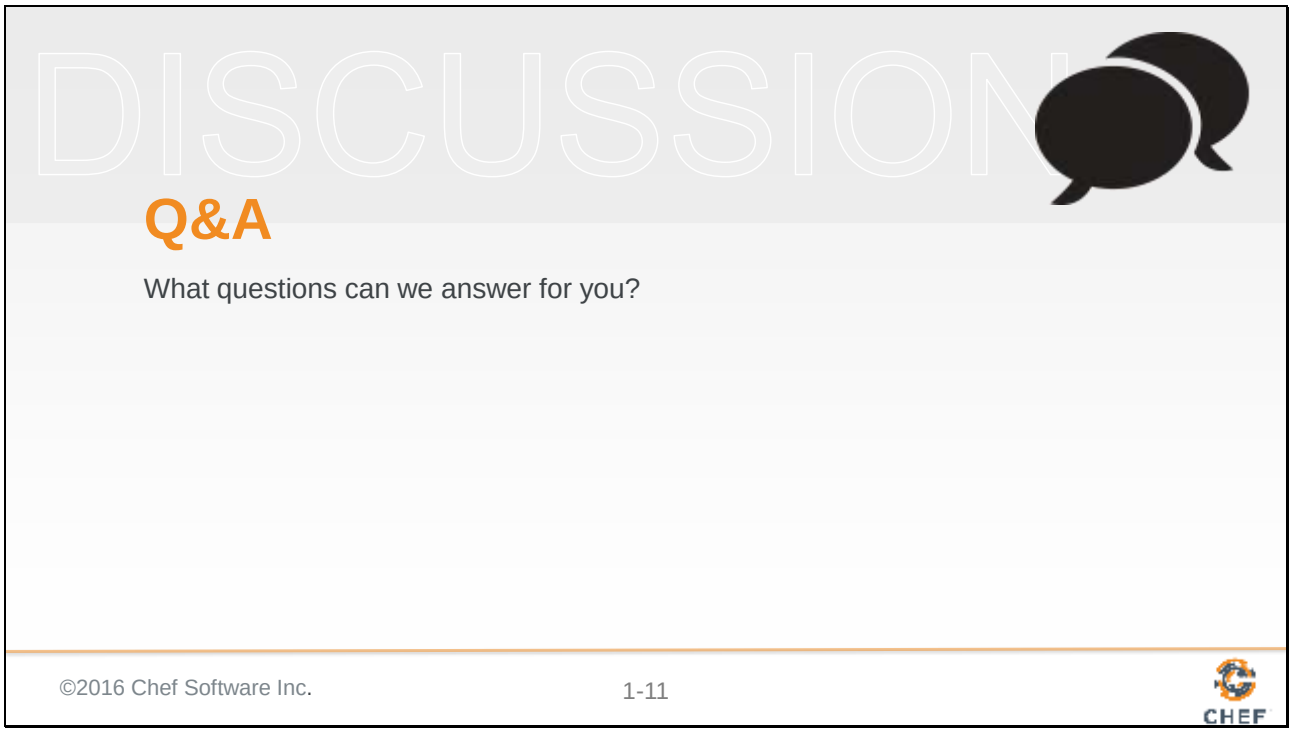
Slide 10



Let us end with a discussion about the following topics.

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.

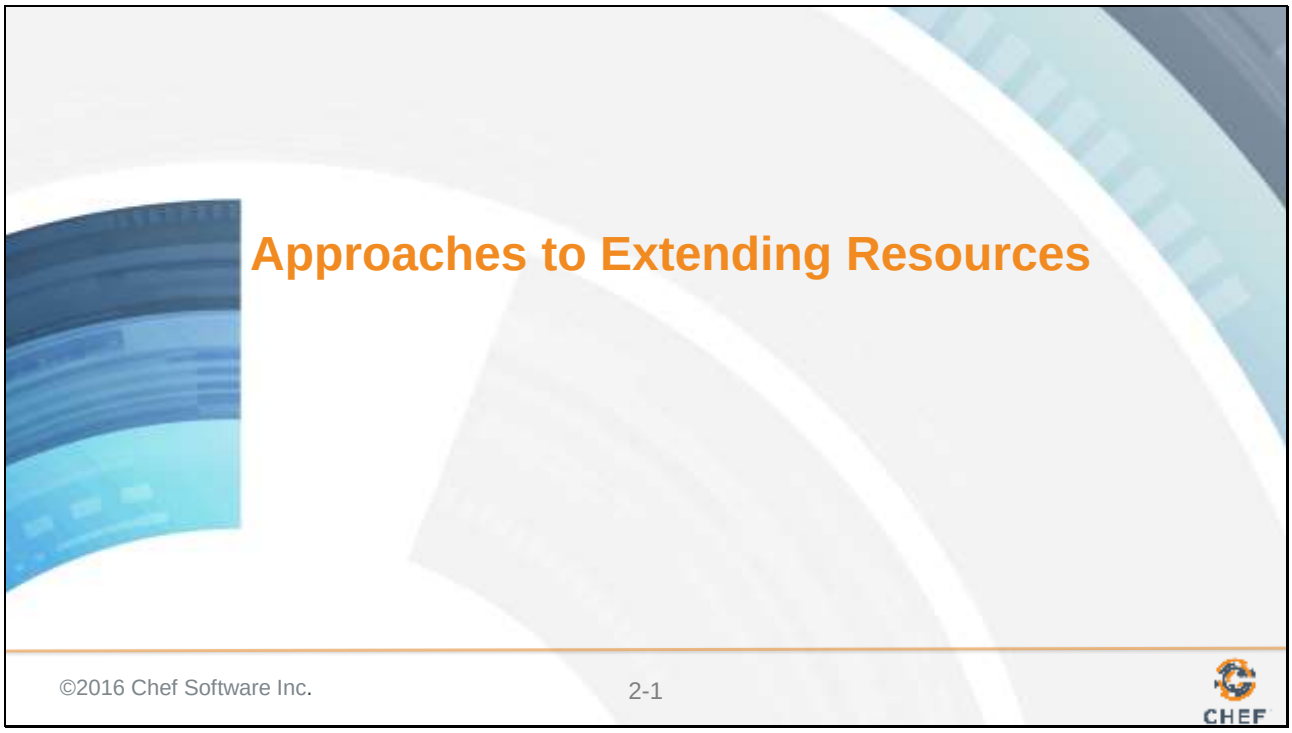
Slide 11



Before we continue let us stop for a moment answer any questions that anyone might have at this time.

Slide 12

2: Approaches to Extending Resources



You express the state of your infrastructure with resources, defined in recipes, encapsulated in cookbooks. Chef provides a core set of resources (dependent on your version of Chef and your platform). These core resources allow you to express the desired state of your infrastructure in a majority of situations. They can also be combined together to express the desired state where these individual resources fall short.

Early on when working with Chef these core resources and their ability to be combined will handle a majority of the configuration management issues that you face. After awhile you will come across more specific resource needs that have not yet been created or perhaps help describe a common set of resources you continue to use together.

When a necessary resource does not exist or when you want to express a group of resources a single resource, Chef provides a few ways to accomplish this.

Slide 2

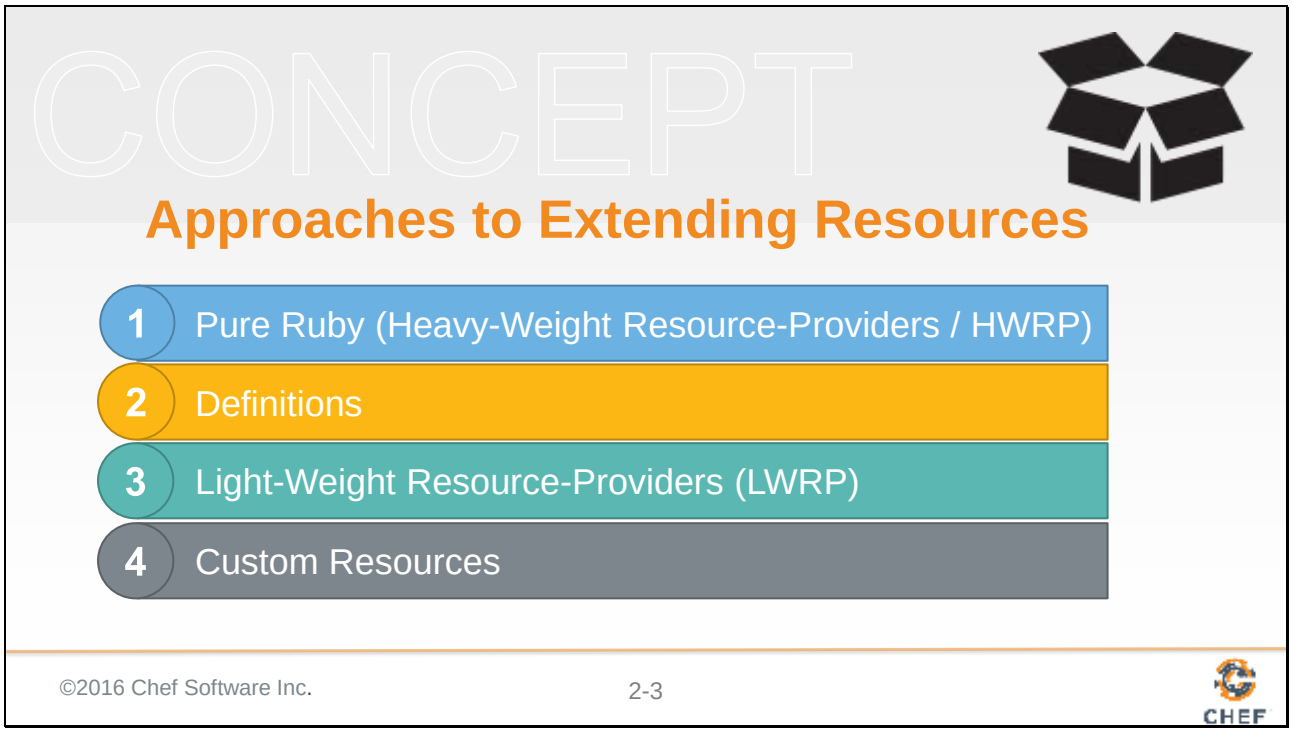
Objectives

After completing this module, you should be able to:

- Describe the difference between:
  - Custom Resources
  - Definitions
  - Heavy-Weight Resource-Providers
  - Light-Weight Resource-Providers

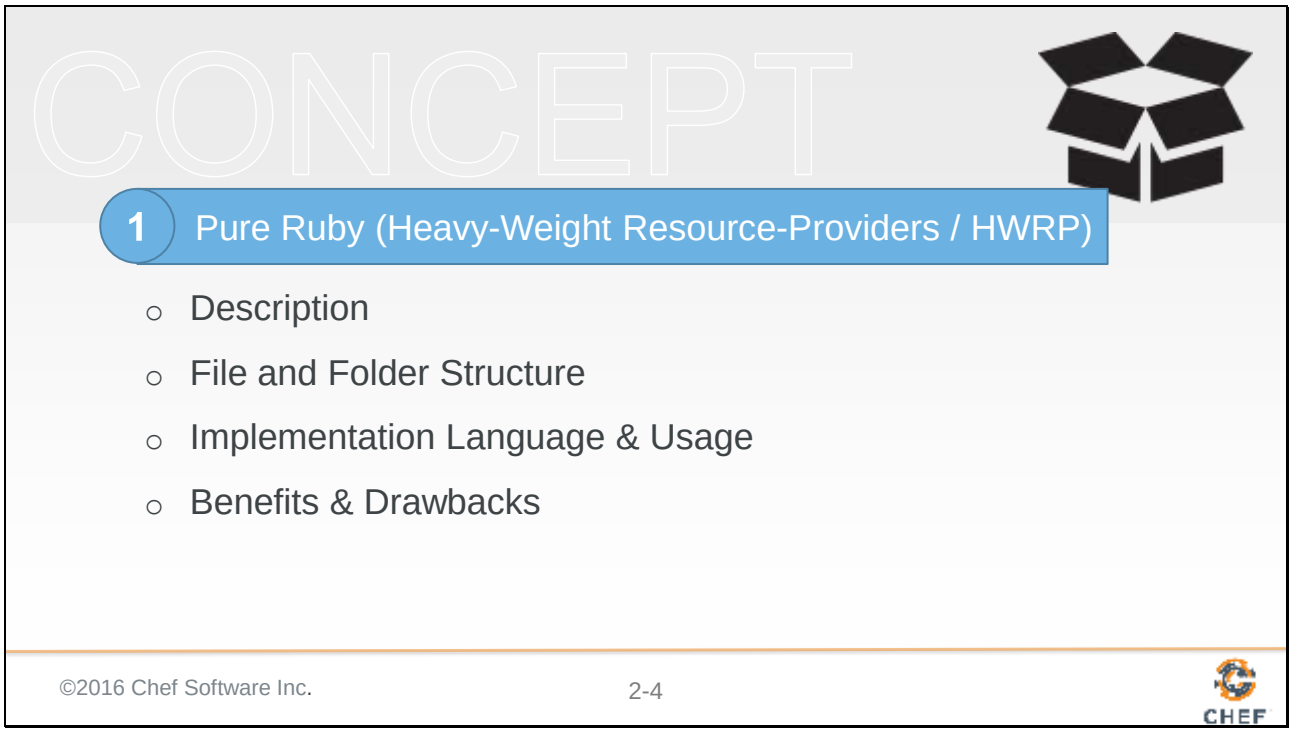
After completing this module you will be able to describe the differences between Custom Resources, Definitions, Heavy-Weight Resource Providers and Light-Weight Resource Providers.

Slide 3



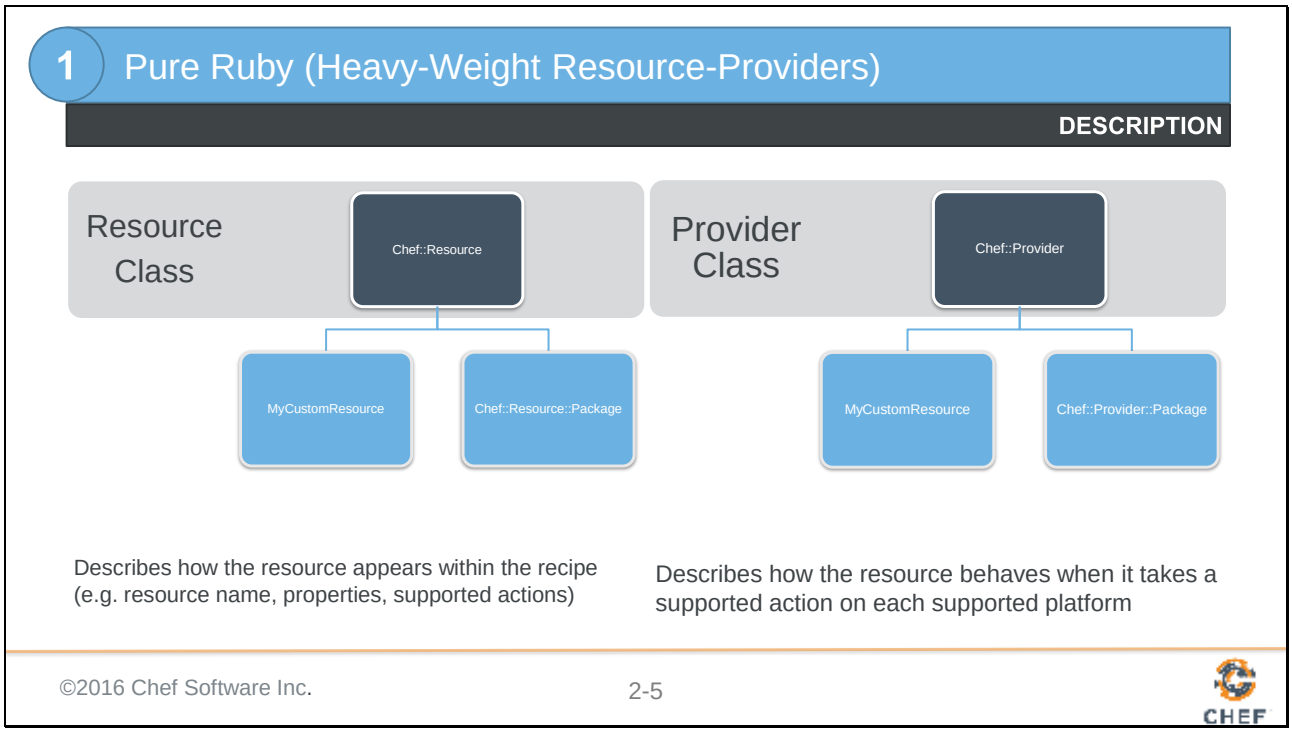
Having reached the limit of the core set of resources presents a new set of challenges before you. Fortunately these challenges are not insurmountable because of some of the design choices Chef has made to make it possible to extend its functionality. Chef is a maturing product that continues to evolve to bring joy to its users. While we are going to focus on Custom Resources it is important that have a basic understanding of these other implementations.

Slide 4



I will provide a description of each, explain the files and folder structure, take a quick look at how each is implemented, and then talk about any requirements or limitations when pursing this implementation choice.

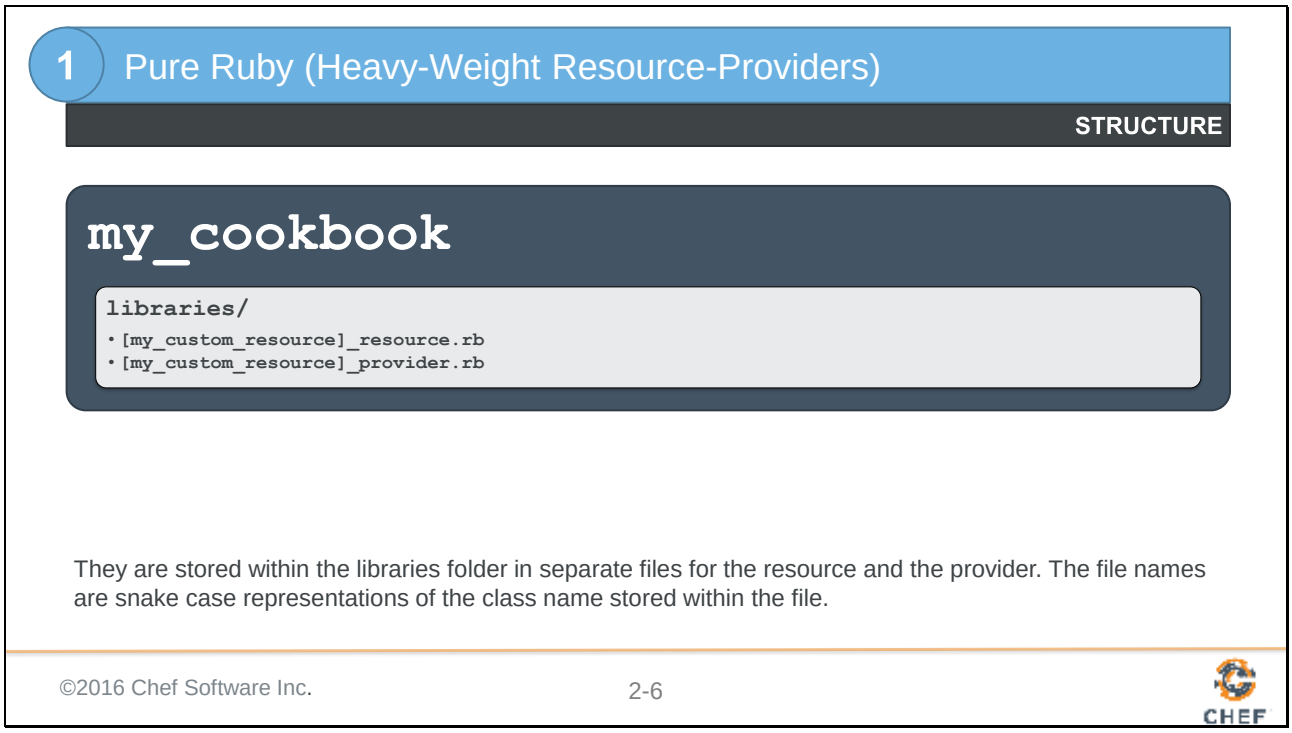
Slide 5



Chef's core resources are written in Ruby. The first approach to creating your own resources is to create your own with Ruby classes. These pure Ruby implementations of Resources is often referred to as Heavy-Weight Resource-Provider, or HWRP. Each resource defined in Chef is defined in two classes which sub-class the core Chef Resource and Chef Provider class. Sub-classing is an object-oriented programming term that means to inherit characteristics (e.g. methods and variables) from the parent class. Within the subclass you are required to override specific methods for the class to behave as a resource within the system.

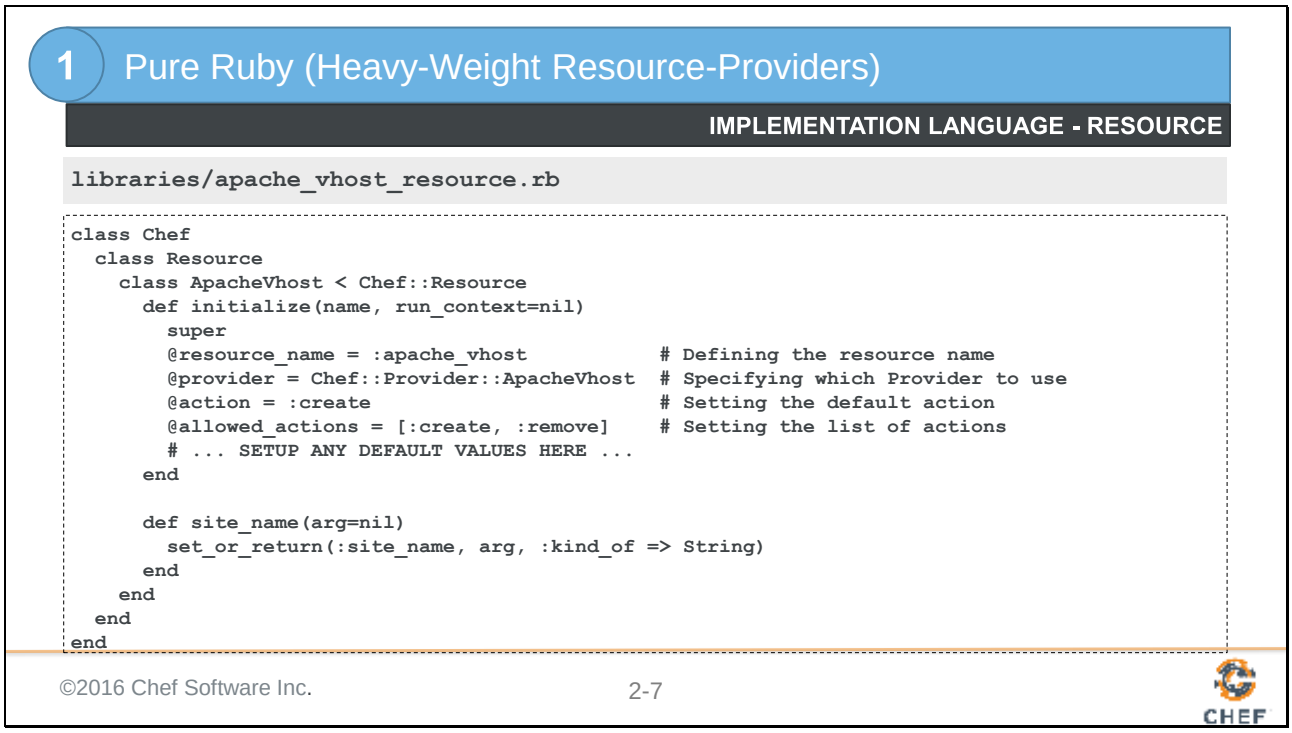
The Chef::Resource class describes how the resource appears within the recipe; the interface. The Chef::Provider class describes how the resource will act when it takes one of the supported action on each supported platform.

Slide 6



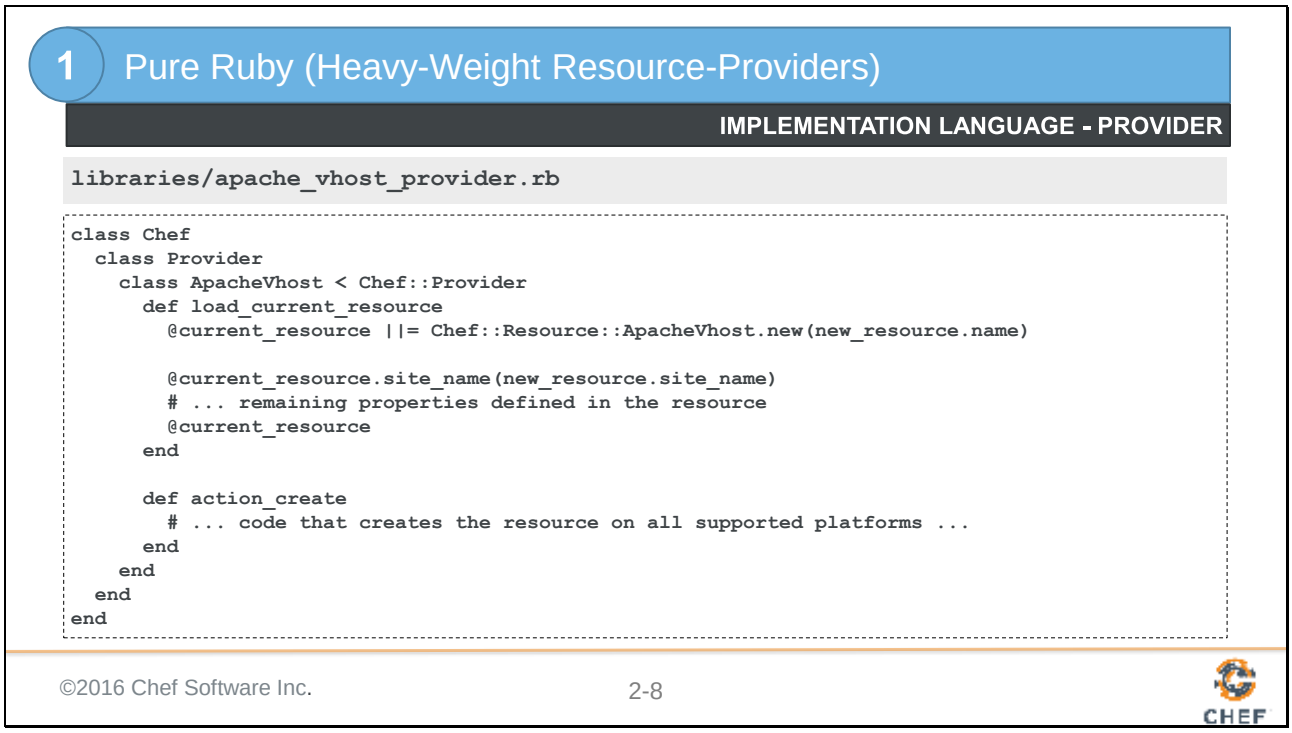
An HWRP, as pure Ruby, is stored in within the 'libraries' directory. Each class, one for the resource and the provider, are stored in separate files. The name of the file matches the class name except it has been snake-cased. Snake-casing lower cases the class name and places underscores between letters where capital letters used to exist. This is a common Ruby practice and one enforced by Rubocop. All the Ruby files within that directory are evaluated after the cookbook is synchronized and loaded.

Slide 7



When defining the resource for a Heavy-Weight Resource-Provider you sub-class the Chef Resource class. The initialize method is overridden to specify new default values and allows us to configure the class as necessary when the resource is created in memory. Each potential attribute is defined as a method which uses a helper to setup the default values, value types it supports, etc.

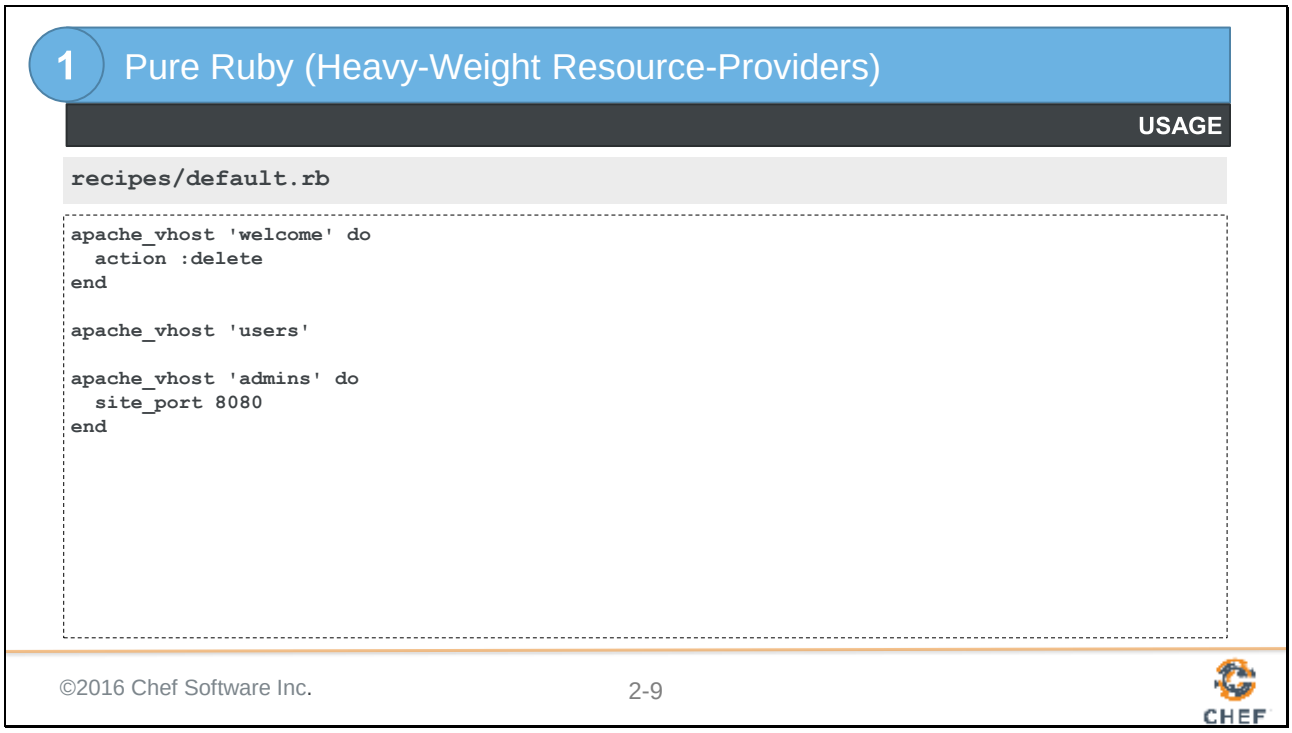
Slide 8



When defining the provider for a Heavy-Weight Resource-Provider you sub-class the Chef Provider class. The initialize method does not have to be overridden. The load_current_resource method must be overridden and is where the configuration from the resource is created and configured for use in each of the supported actions. The actions here are defined as methods with the prefix 'action_' and within them you would define the code necessary to perform the operations for this resource.

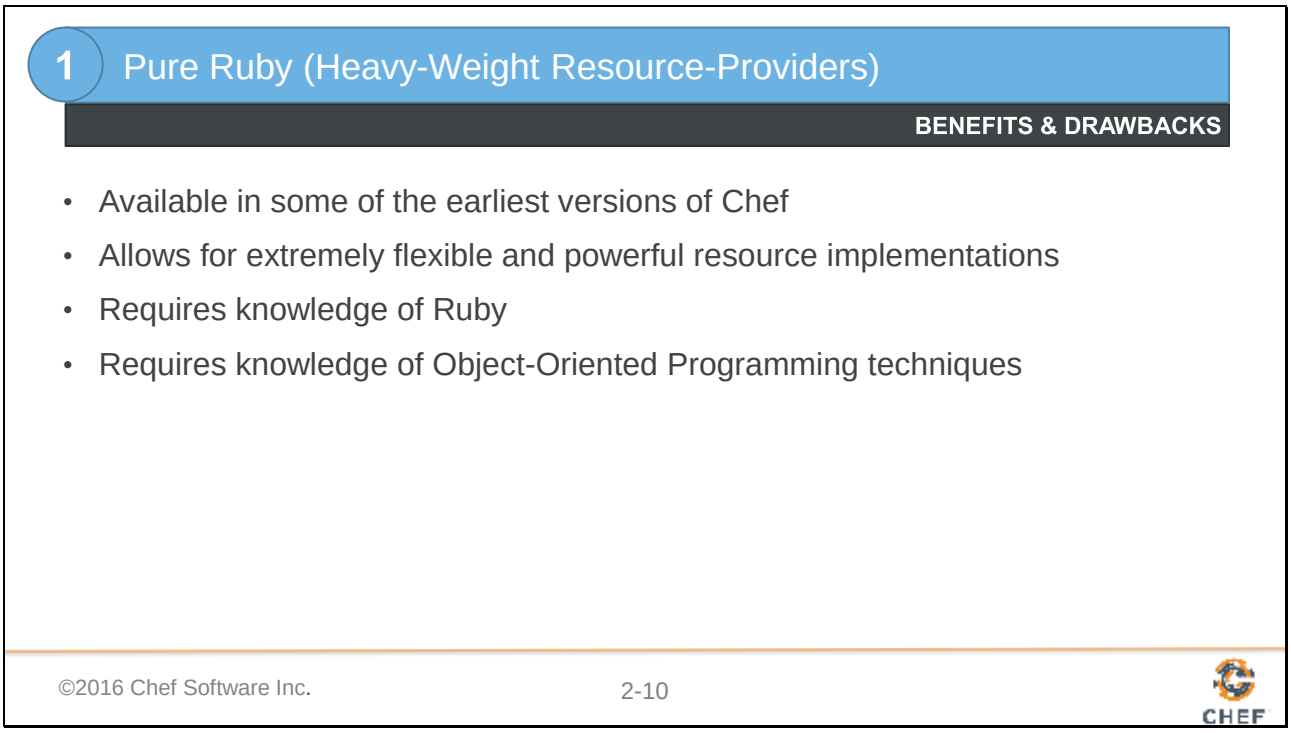
Chef provides additional helpers to allow you to shell out to perform operations on the system. You also have the entire Ruby language and any gems that might be packaged with the Chef DK (or you have added to Chef DK) at your disposal.

Slide 9



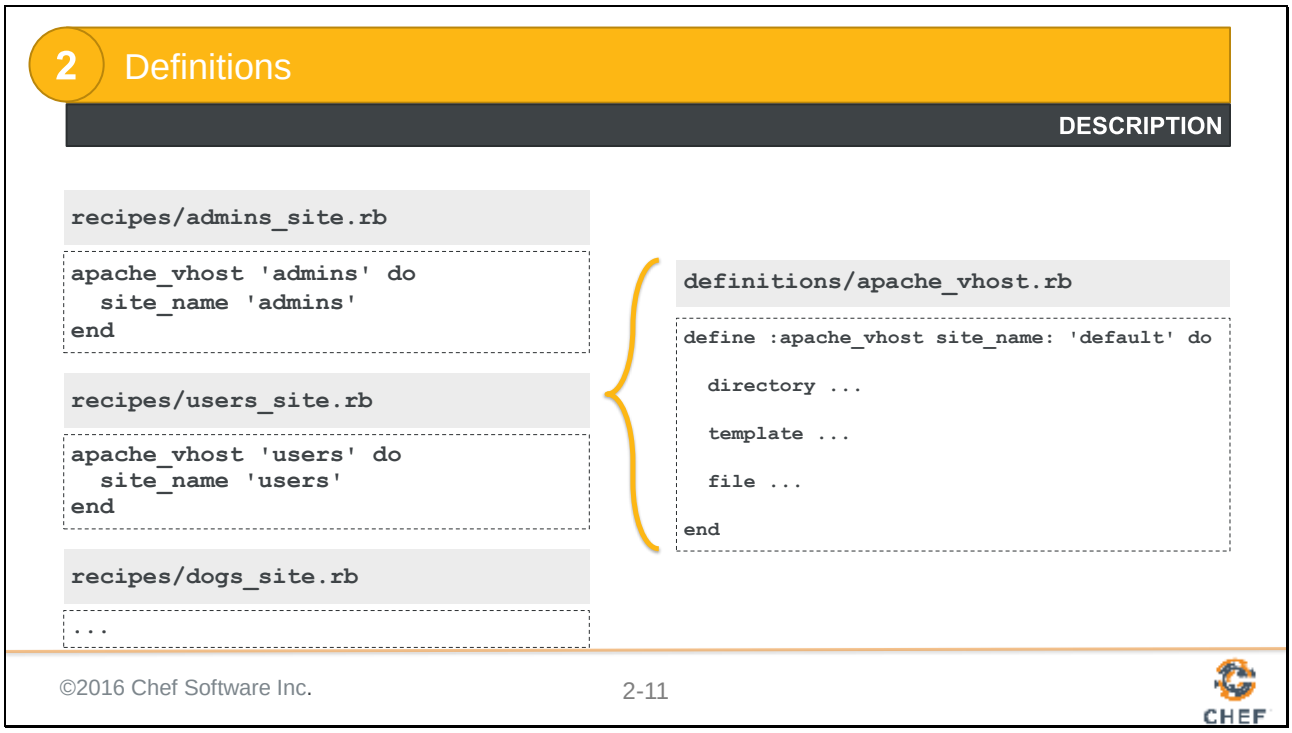
The resource would now be available within any recipe defined in this cookbook or any cookbook that adds this cookbook as a dependency. Here in this example recipe the resources delete and creates apache sites. All three of the resources rely on the site name attribute being tied to the name provided to the resource. The first deletes the welcome site. The next two both rely on the default action of create. The second resource assumes the default site port value.

Slide 10



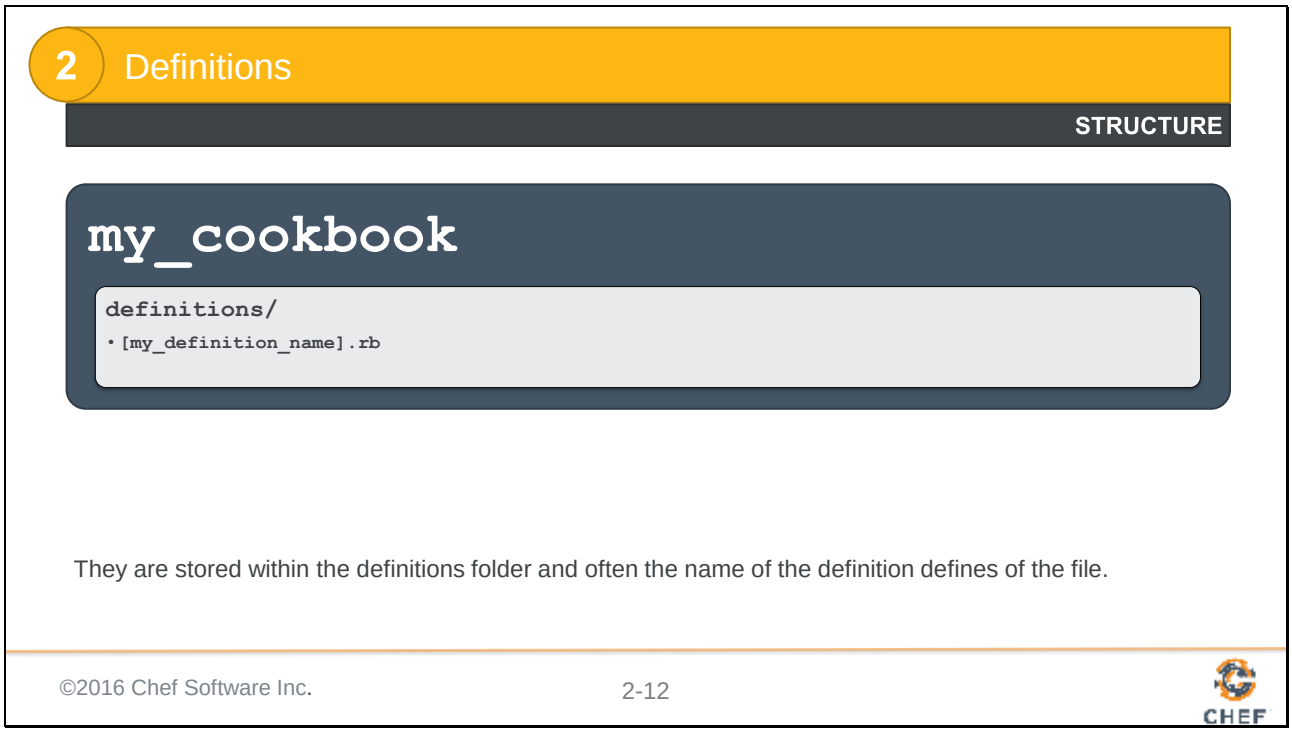
HWRP are incredibly useful when you need the full power of Ruby to implement your own resource. However, they come at the cost of understanding a number of Object-Oriented Programming techniques and the Ruby language. When exploring community cookbooks you may find examples of these resources in use.

Slide 11

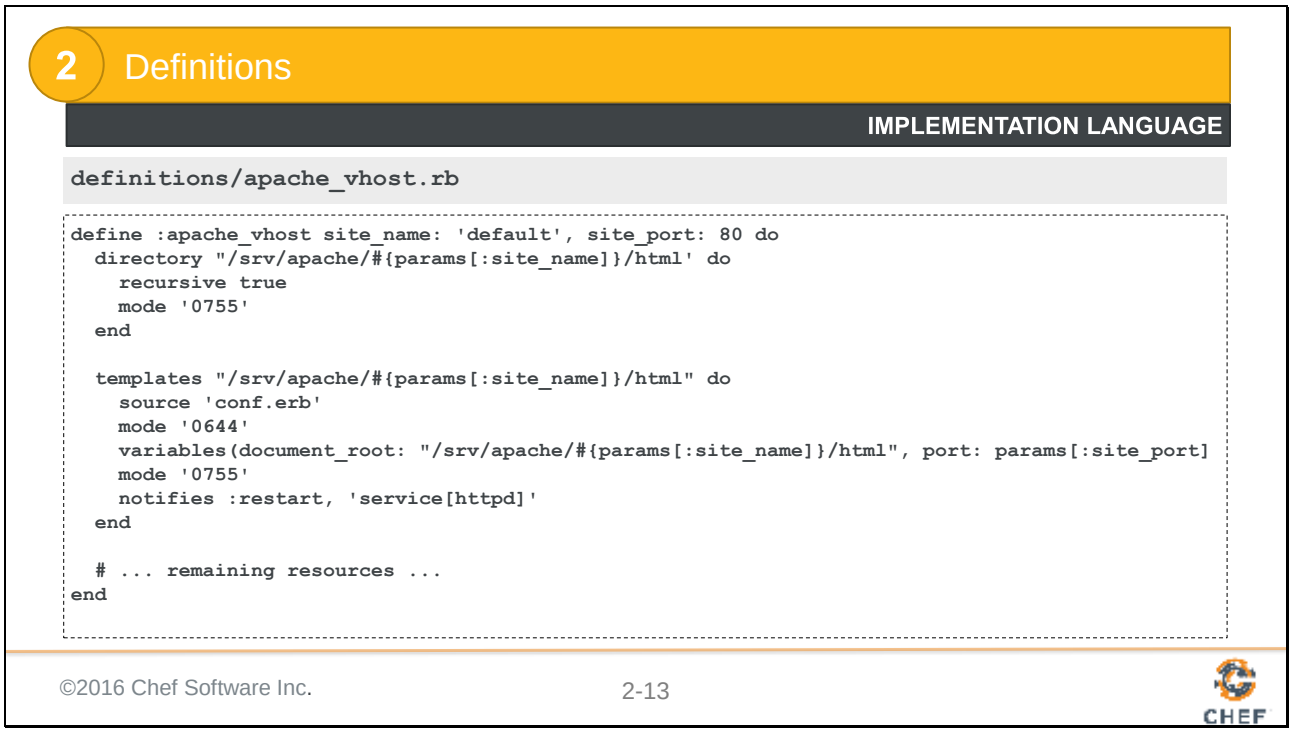


Definitions behaves like a compile time macro that is reusable across recipes. Macros all you to write a small amount of code that expands out into the contents of the definition wherever it is found within the recipes. With a definition you give it a name, provide parameters, and specify a block of code.

Slide 12

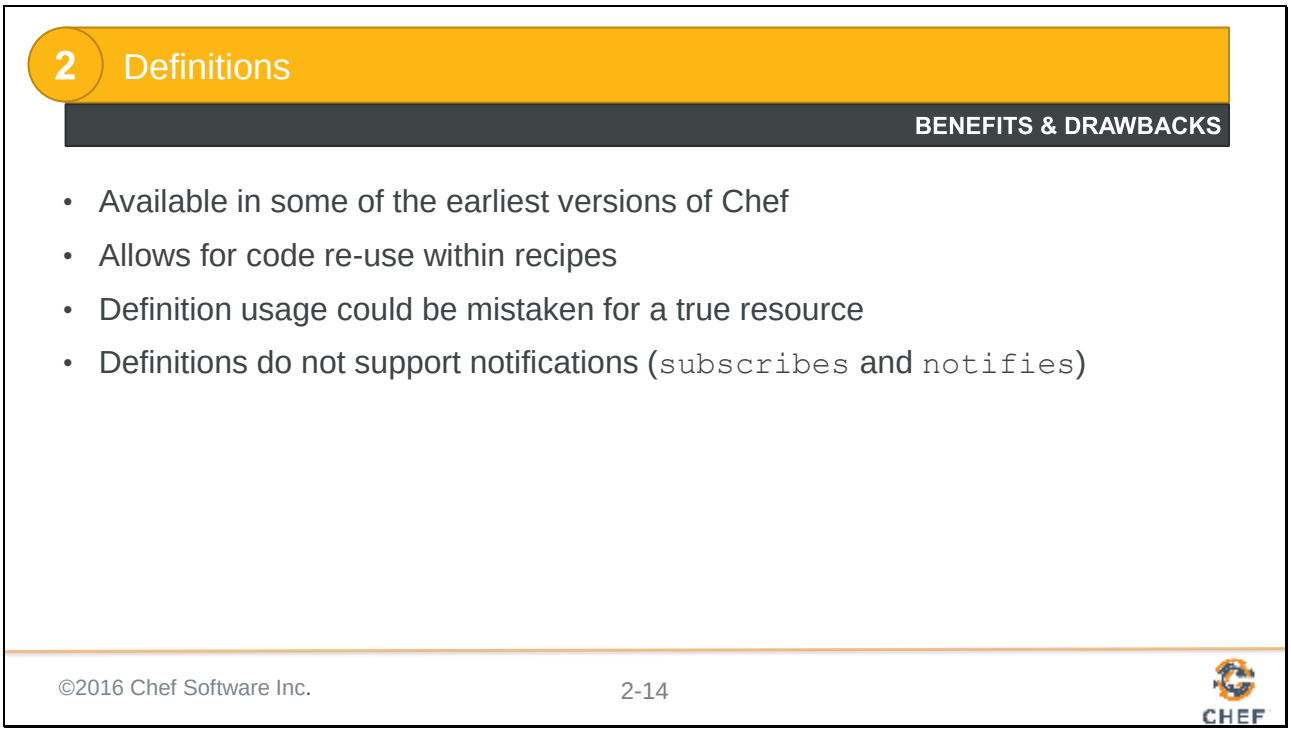


The code that defines the definition is stored within a definitions directory in a Ruby file that is processed with the definition Domain Specific Language.



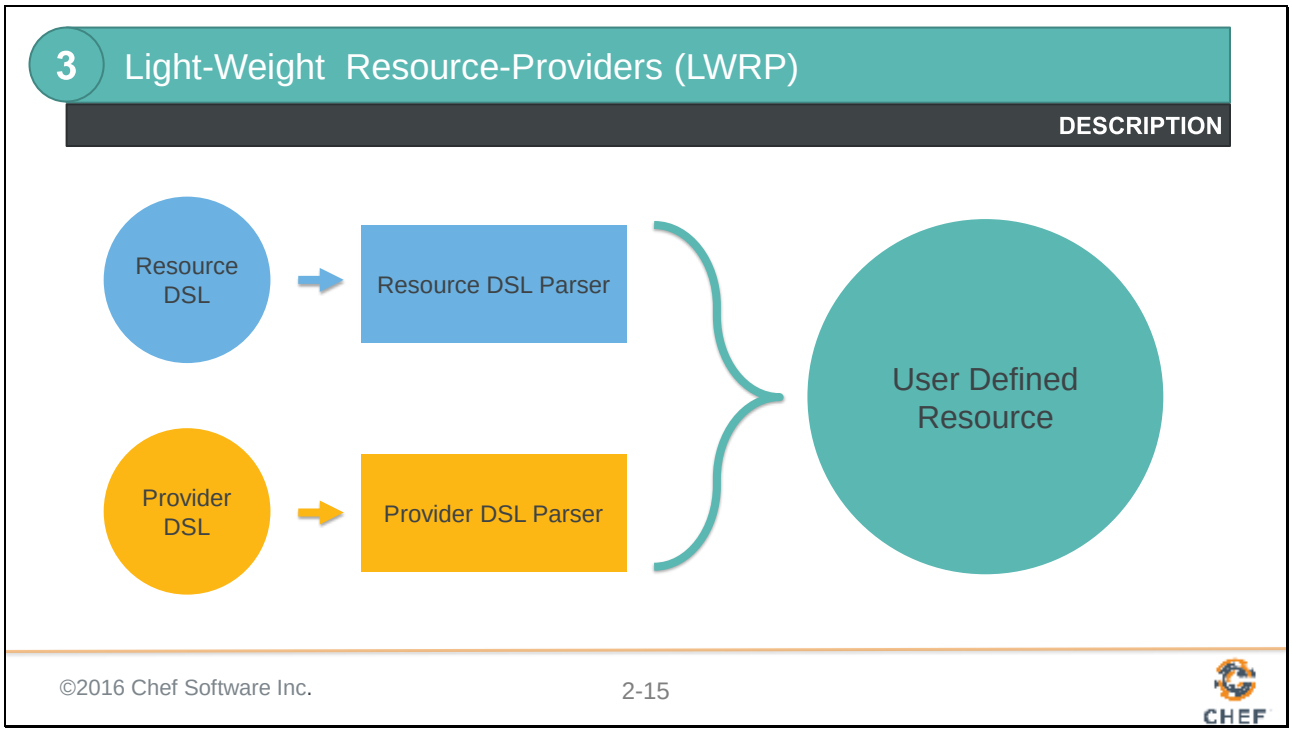
When creating a definition you specify a name and a hash of any parameters you wish to provide. Within the definition the parameters are retrievable from a hash named params. The use of the definition within a recipe looks similar to a resource but that is not the case. Definitions cannot notify other resources, subscribe to notifications from other resources, (i.e. `notifies` and `subscribes`) and cannot employ guards (i.e. `only_if` and `not_if`).

Slide 14



Definitions shipped in some of the earliest versions of Chef and are still supported today. However, as of Chef 12.5 it is strongly recommended that you choose a solution built with custom resources.

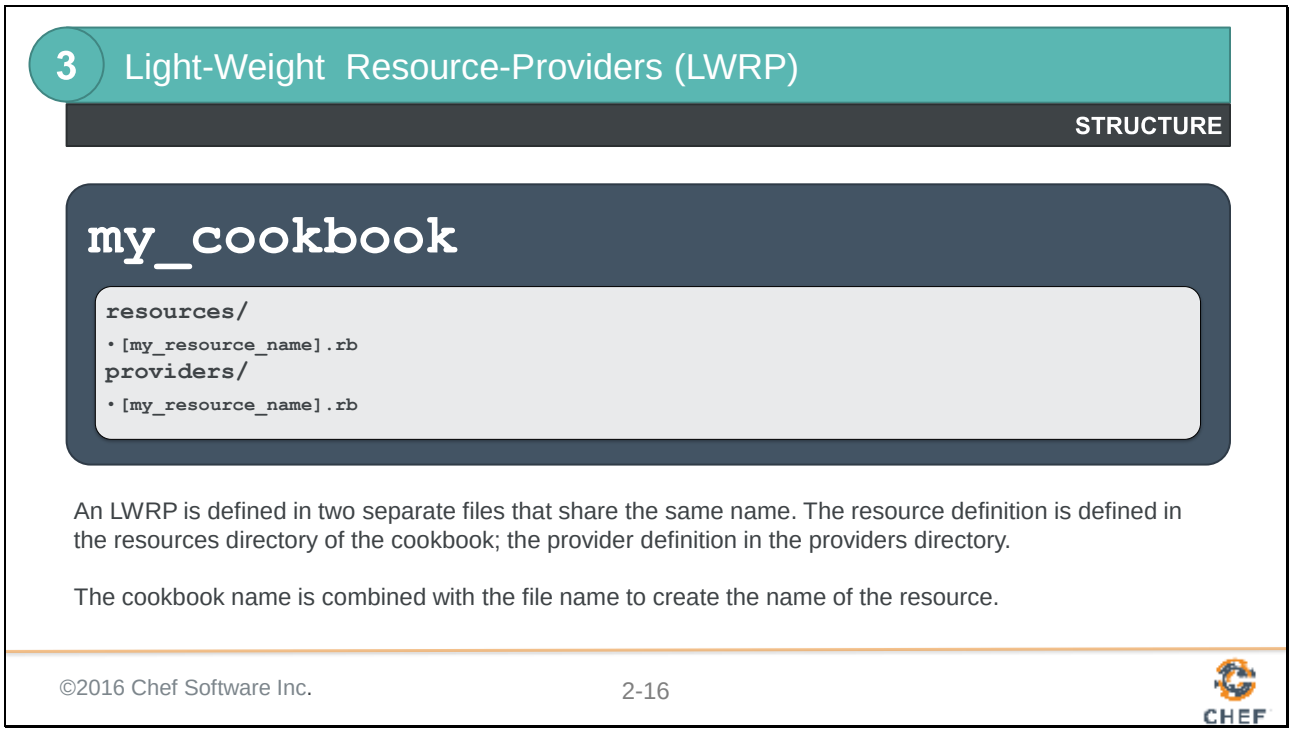
Slide 15



Light-Weight Resource-Provider, or LWRP, are Chef resources defined in two Domain Specific Languages (DSL) that allow you to create resources without having to understand the complexity presented by HWRP.

An LWRP is as much a resource as the core resources defined in Chef. The resource and the provider is parsed and converted into Ruby objects.

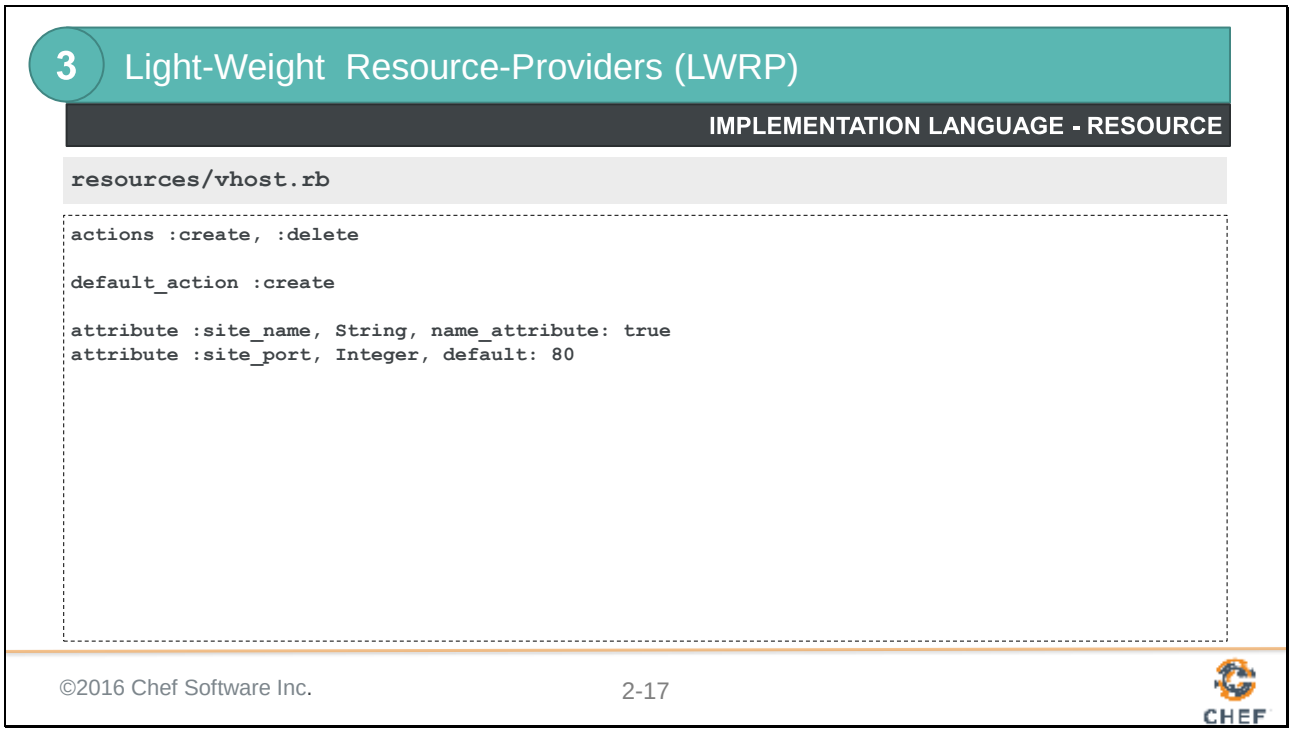
Slide 16



A single LWRP definition is defined in two separate files. The file is named exactly the same but one file resides in the 'resources' directory; the other in the 'providers' directory. Both of these files are parsed after the cookbook is synchronized and loaded. Each file's DSL is then converted into Ruby class at runtime.

Within the file in the 'resources' directory you define the interface for the custom resource. There, within a resource DSL, you can specify a name of the resource, the list of available actions, the default action, and the properties that may be set for the resource. Within the file in the 'providers' directory you define the implementation for the custom resource. There, within a provider DSL, you specify what happens when an action is chosen.

Slide 17

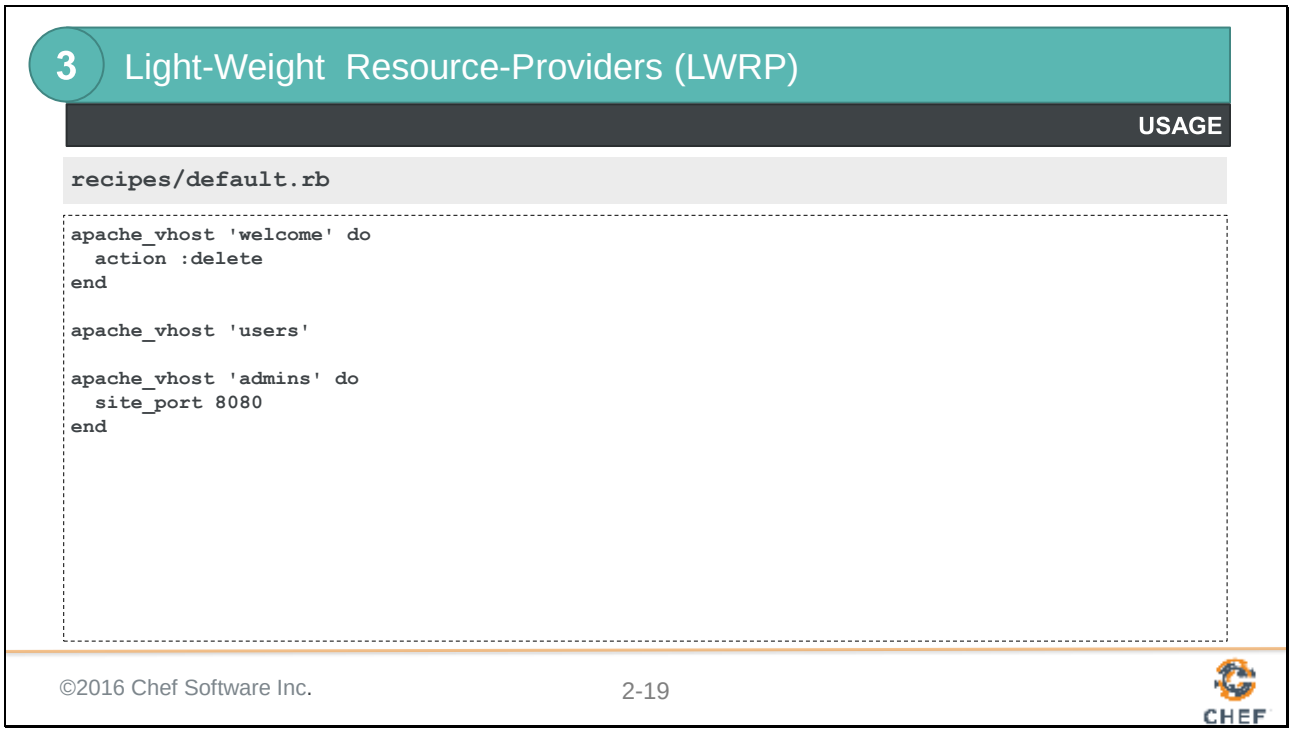


Within the resources file you specify the available actions, the default action, and the supported attributes that can be used when specifying the resource.

Slide 18

Within the provider definition you specify action blocks for each of the actions defined in the resource file. Within the action you specify resources as if you are defining a small recipe. The attributes defined for the resource are available within the action through a local variable or method named 'new_resource'.

Slide 19



The name of the cookbook is combined with the name of the resource/provider file name with an underscore to create the user defined resource. This was explicitly defined in the HWRP but is automatically generated.

Otherwise this is the same results as the one defined by the HWRP.

Slide 20

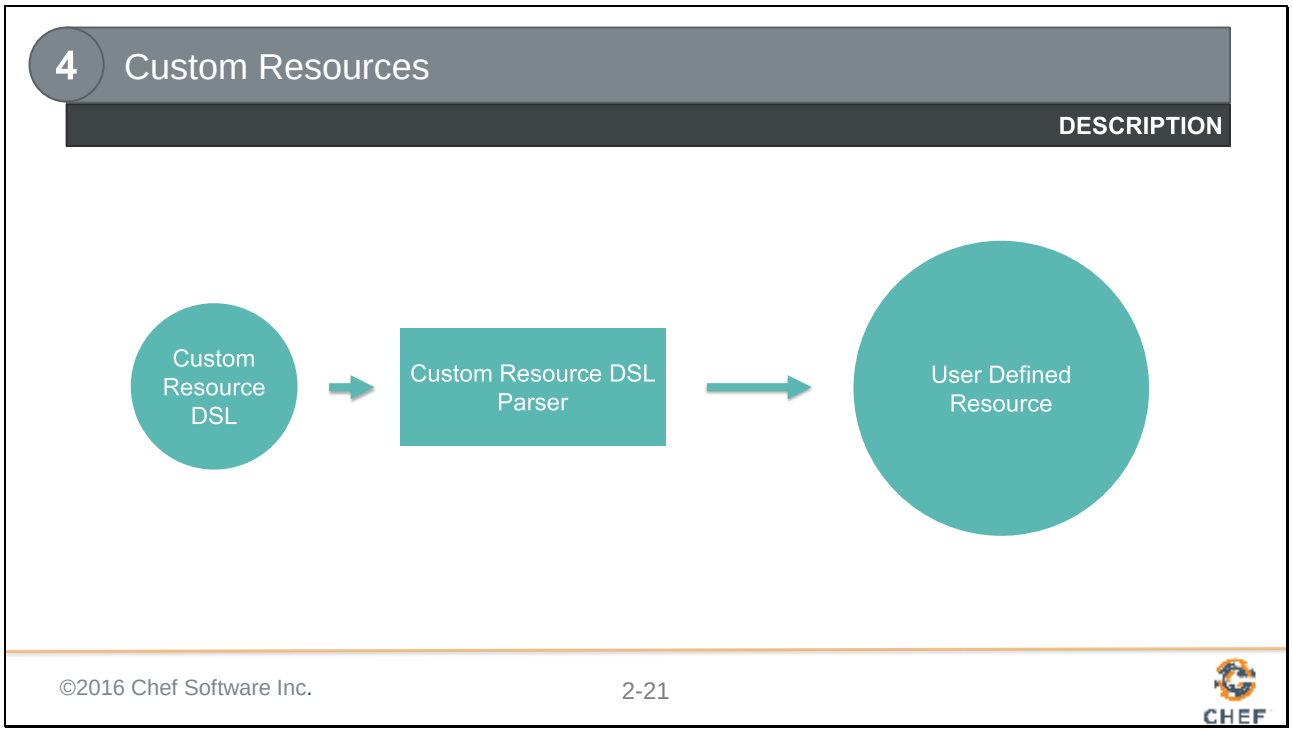
3 Light-Weight Resource-Providers (LWRP)

BENEFITS & DRAWBACKS

- Available in 0.7.12 version of Chef
- Allows for a real resource definition without understanding Ruby (vs. HWRP)
- Resource and provider implementation require learning a new DSL
- Complete resource definition is spread across two files

Implementing resources with LWRP is not the favored way to develop a resource in later versions of Chef (Chef 12.5). However, they are still in wide use within older cookbooks like those found within the Chef Supermarket.

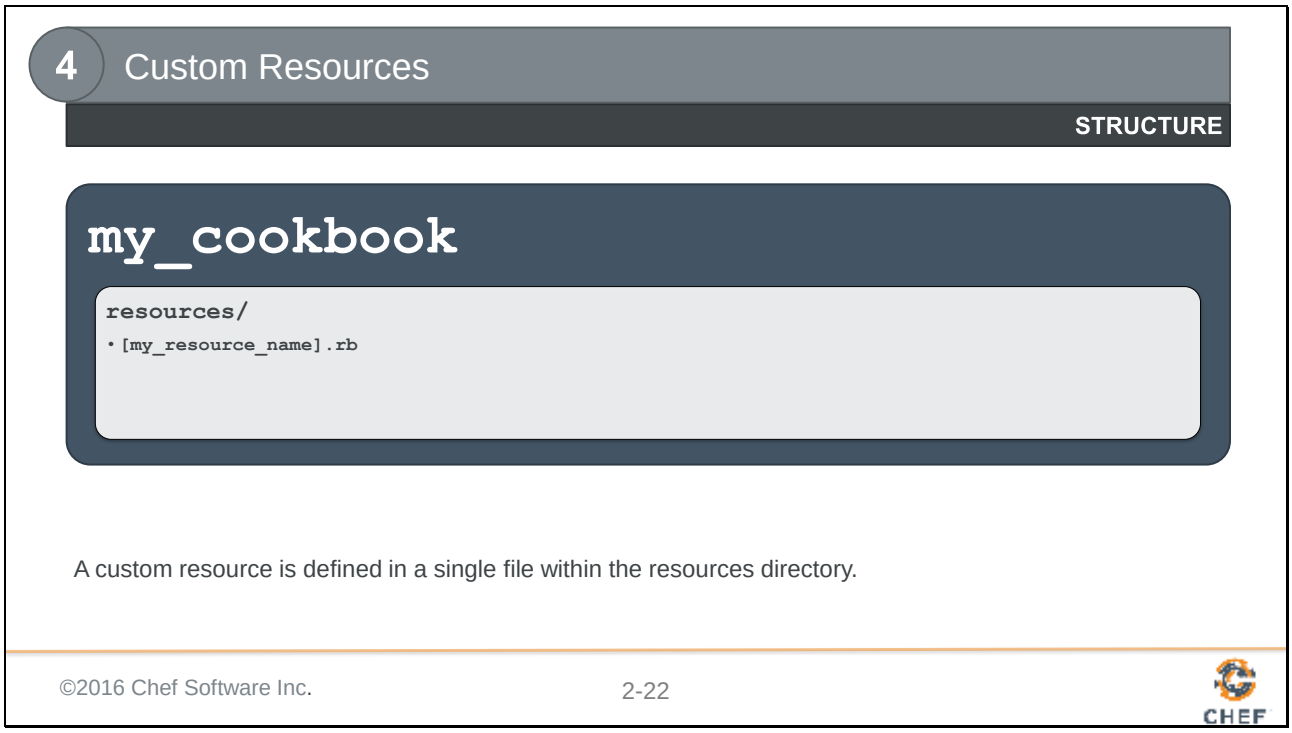
Slide 21



Custom Resources are Chef resources defined in a Domain Specific Language (DSL) that allow you to create resources without having to understand the complexity presented by HWRP. At its core it is a simplification of the work done with LWRP.

An custom resource is as much a resource as the core resources defined in Chef. A custom resource definition is defined in a single file that resides in the 'resources' directory. This file is parsed after the cookbook is synchronized and loaded. The custom resource DSL is then converted into Ruby class at runtime.

Slide 22



Within the file in the 'resources' directory you define the interface and the implementation for the custom resource. This is written in a custom resource DSL where you can specify the name of the resource, the default action, the properties that may be set, and all the actions that the resource supports.

Slide 23

The custom resource implementation is similar to the LWRP except all of the details that describe the resource are combined into a single file. The custom resource DSL is similar to one defined for the LWRP resource and LWRP provider DSL. It is an evolution of the LWRP implementation with some minor changes. The attributes are instead called properties and when used within the action implementations they no longer require the 'new_resource' local variable or method. The default action is assumed to be the first action defined in this file: create.

Slide 24

The result is the same here as the HWRP and LWRP.

The default action is determined by the first action listed in the custom resource definition.

Slide 25

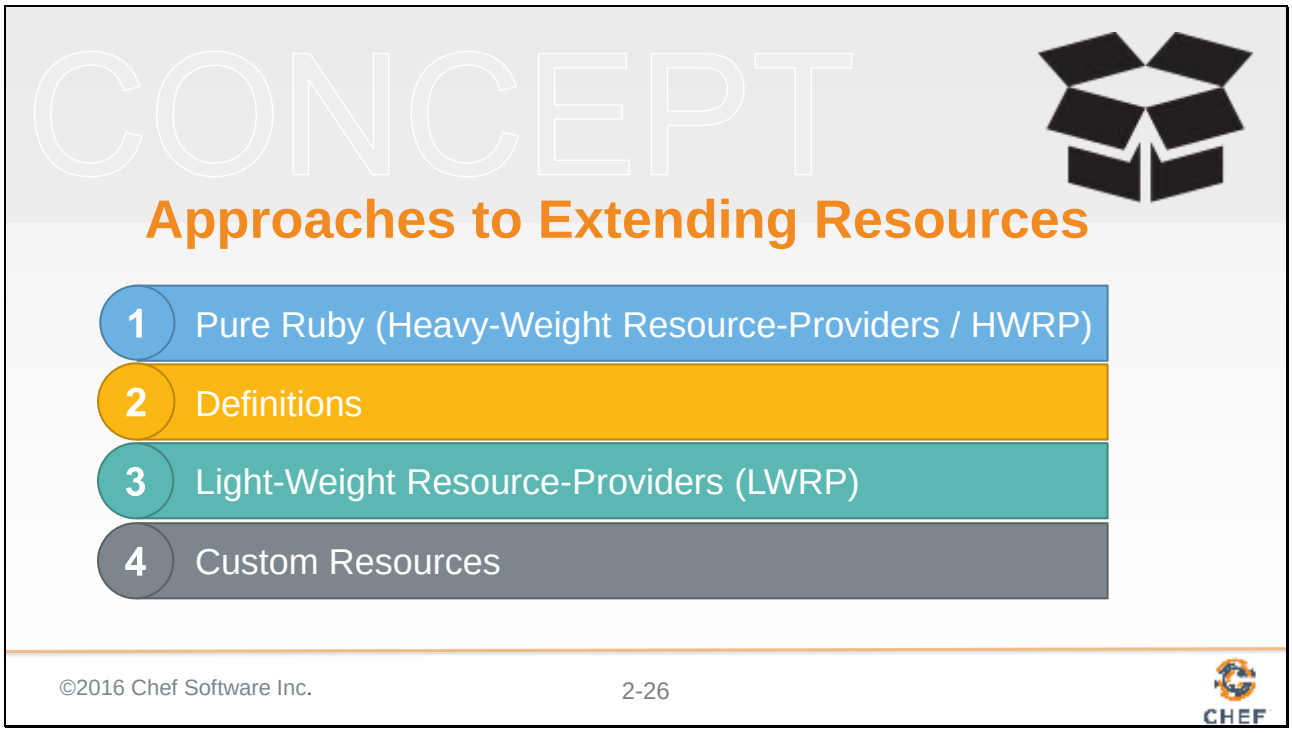
4 Custom Resources

BENEFITS & DRAWBACKS

- Available in 12.5.0 version of Chef
- Allows for a real resource definition without understanding Ruby (vs. HWRP)
- Complete resource definition is defined in a single file (vs. LWRP)
- Custom resource implementation require learning a new DSL

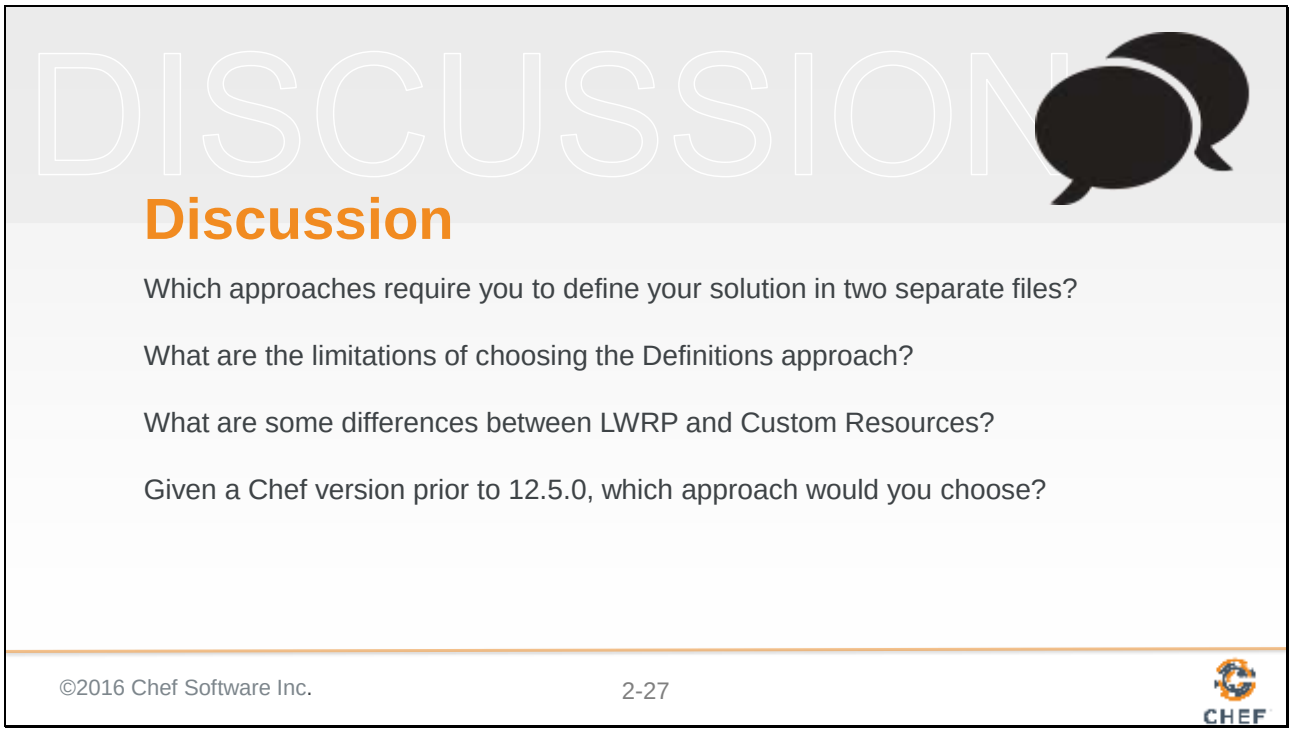
Implementing resources with a custom resource is the current favored way to develop a resource for versions of Chef 12.5.X or greater. They are easier to implement than a pure Ruby implementation and are defined in a single file compared to the LWRP implementation.

Slide 26



As you can see there are more than a few ways to extend Chef and create a resource or resource-like implementation within your recipes.

Slide 27



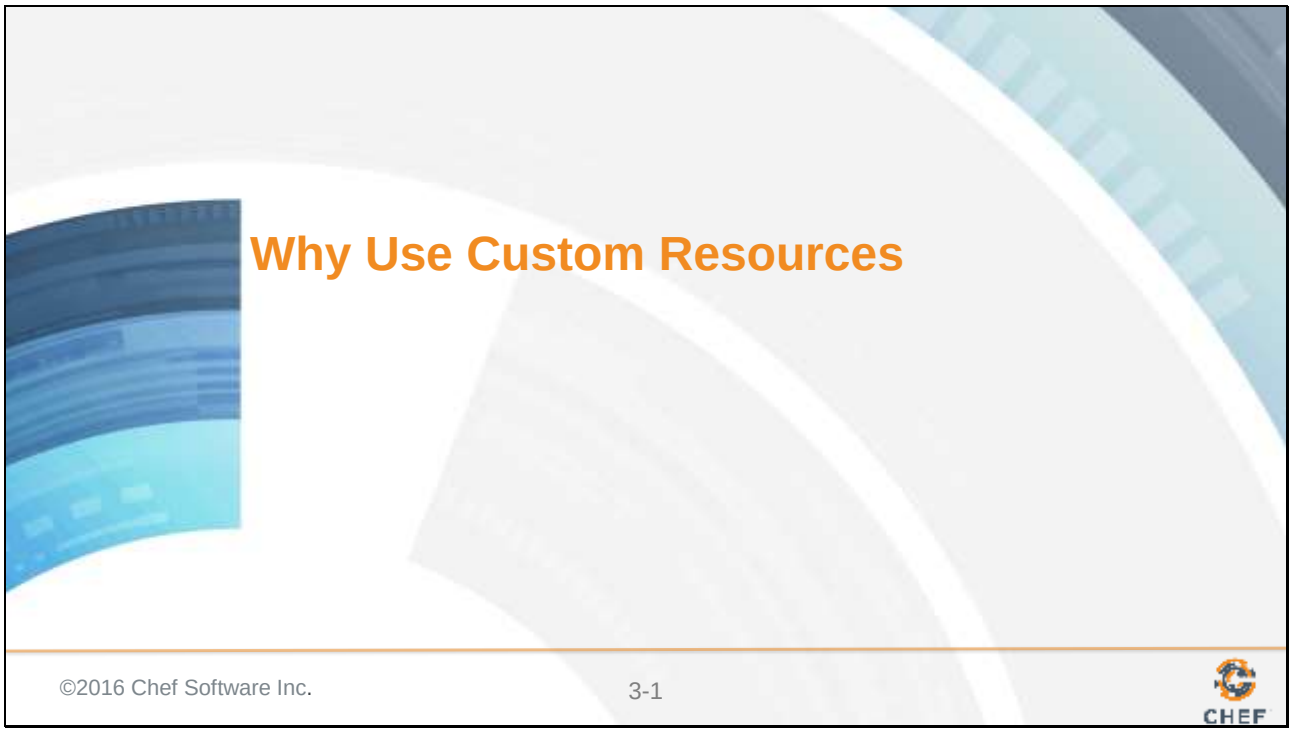
As a group, let's answer these questions.

Slide 28

What questions can we answer for you?

Slide 29

3: Why Use Custom Resources



As you can see there are more than a few ways to extend Chef and create a resource or resource-like implementation within your recipes. But before we do that, it is important to understand the value that a custom resource brings to a recipes.

Slide 2

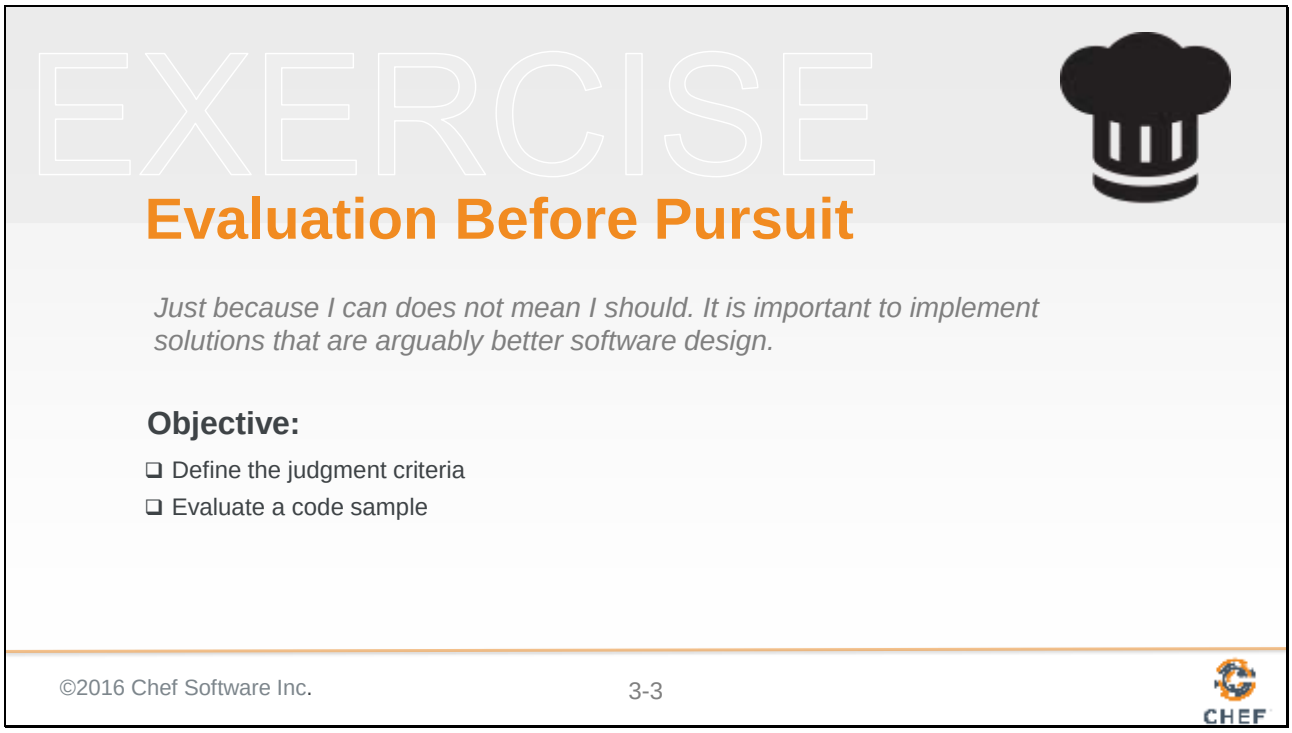
Objectives

After completing this module, you should be able to:

- Determine when a Custom Resource would be beneficial for clarity and reusability

After completing this module you will be able to describe when a Custom Resource would be beneficial for clarity and reusability.

Slide 3



As an group exercise we are going to look at a series of resources and discuss their quality. Quality can be rather variable unless we select a criteria for which to judge it.

Slide 4

When defining resources within our recipes we are writing software. Software has a number of quality characteristics that have already been defined. ISO/IEC 9126 is an international standard for evaluation of software quality.

Slide 5

This standard identifies 6 main quality characteristics. Let's talk about each one of these so that we have a shared understanding of what we mean when using them in this exercise.

Slide 6

CONCEPT

Software Quality Standards

- **Functionality**
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Does the code accomplish what it is designed to accomplish?

CHEF

Functionality is the essential purpose of any product or service. Does the code accomplish what it is designed to accomplish? Functionality may also be concerned with if it does so securely and within compliance guidelines.

Slide 7

CONCEPT

Software Quality Standards

- Functionality
- **Reliability**
- Usability
- Efficiency
- Maintainability
- Portability

Is the solution able to withstand fault and recover from a failure?

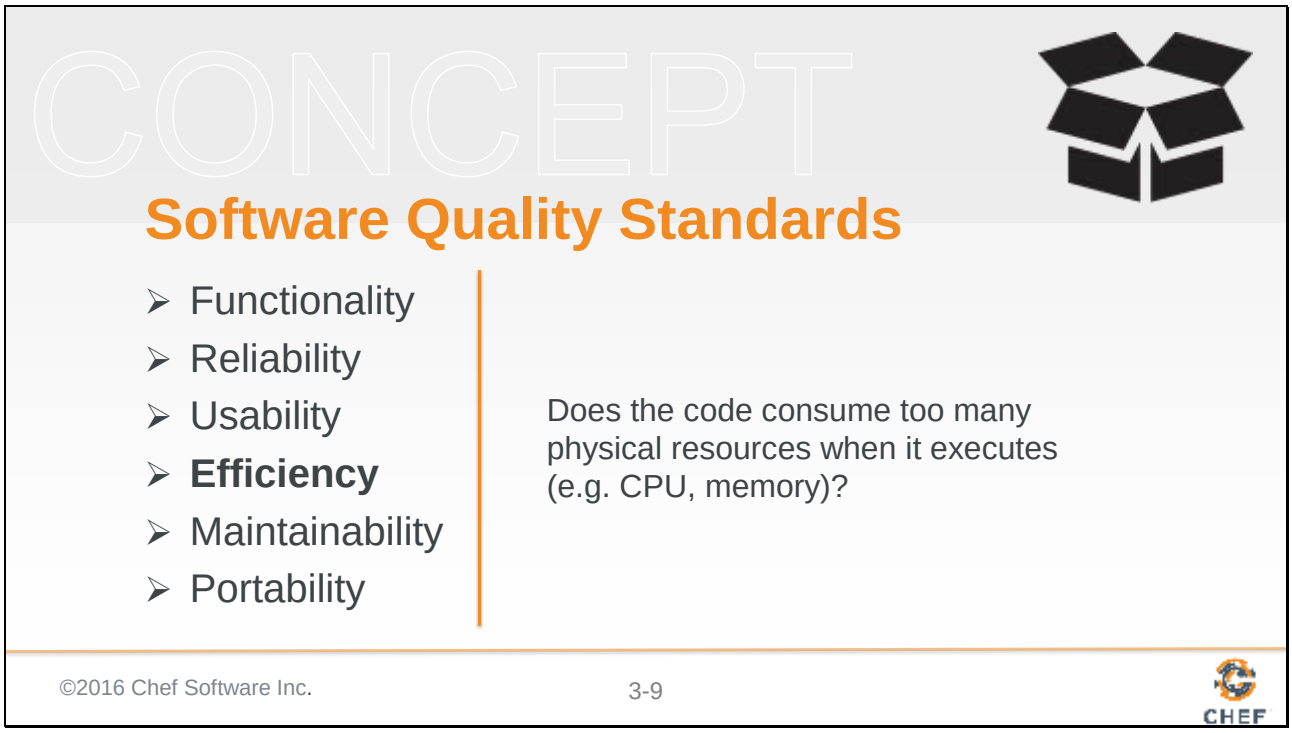
CHEF

Reliability is a judgment of whether the code accomplishes its functional goal consistently, is able to withstand fault, and recover from a failure.

Slide 8

Usability refers to the ease of use for the given code. Is the code easy to understand? Is it easy to learn? Does it adhere to common team standards?

Slide 9



Efficiency is concerned with the system resources required to achieve the functionality. We may consider the time, CPU, memory, network requirements, or physical space it takes to accomplish the intended operation.

Slide 10

CONCEPT

Software Quality Standards

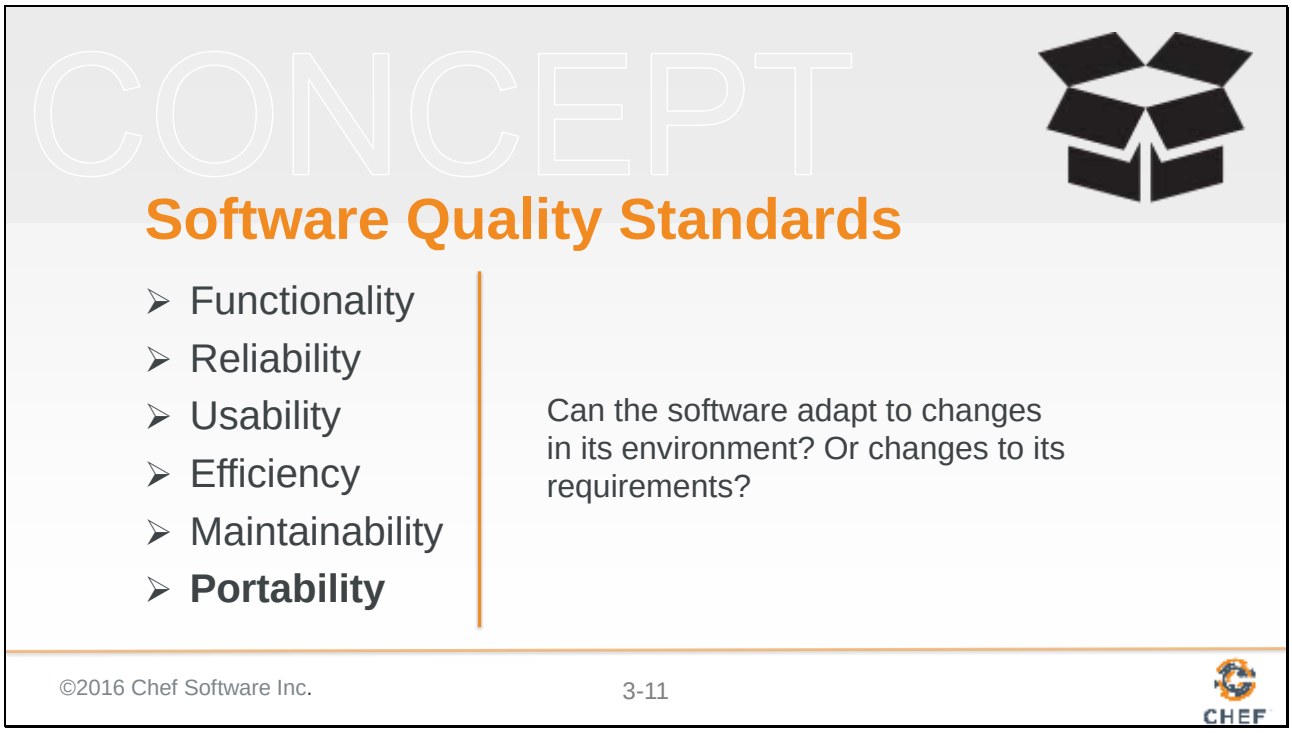
- Functionality
- Reliability
- Usability
- Efficiency
- **Maintainability**
- Portability

Are you able to easily adapt the solution? Is it testable?

CHEF

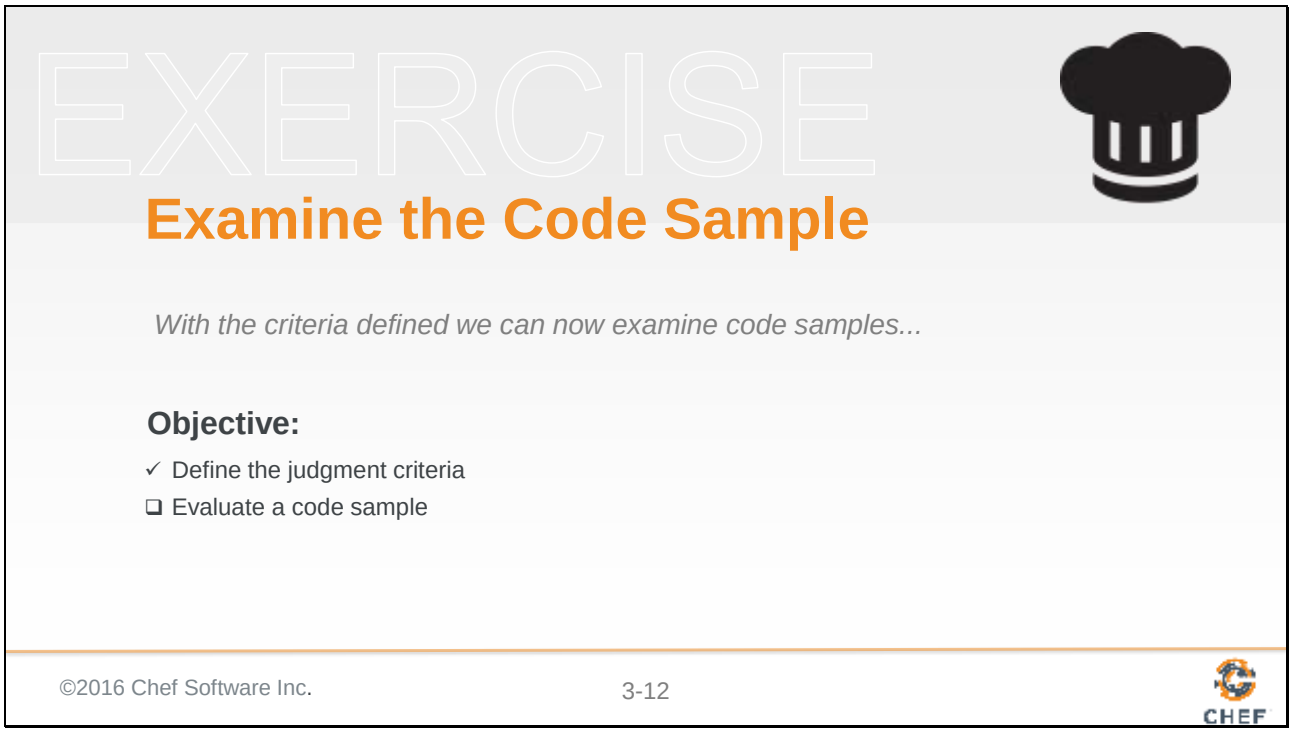
Maintainability measures the code to see if it is supportable. If there is a failure are you able to quickly identify the issue? Are you able to easily adapt the solution? Is it testable?

Slide 11



Portability refers to how well the software can adapt to changes in its environment or with its requirements. This may also include evaluating code for its adaptability and maybe even be easily replaced.

Slide 12



Let's examine this first example and apply the criteria that we have defined.

Resource Implementation v Custom Resource

```
directory '/srv/apache/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/apache/admins/ht
ml',
port: 8080)
  notifies :restart, 'service[httpd]'
end

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

```
apache_vhost 'admins' do
  site_port 8080
end
```

Functionality | Reliability | Usability | Efficiency | Maintainability | Portability

Does the code accomplish what it is designed to accomplish?

CHEF

Resource Implementation v Custom Resource

```
directory '/srv/apache/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/apache/admins/ht
ml',
port: 8080)
  notifies :restart, 'service[httpd]'
end

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

```
apache_vhost 'admins' do
  site_port 8080
end
```

Functionality | **Reliability** | Usability | Efficiency | Maintainability | Portability

Is the solution able to withstand fault and recover from a failure?

CHEF

Resource Implementation v Custom Resource

```
directory '/srv/apache/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/apache/admins/ht
ml',
port: 8080)
  notifies :restart, 'service[httpd]'
end

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

```
apache_vhost 'admins' do
  site_port 8080
end
```

Functionality | Reliability | **Usability** | Efficiency | Maintainability | Portability

Is the code easy to understand? Is it easy to learn?

Resource Implementation v Custom Resource

```
directory '/srv/apache/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/apache/admins/ht
ml',
port: 8080)
  notifies :restart, 'service[httpd]'
end

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

```
apache_vhost 'admins' do
  site_port 8080
end
```

Functionality | Reliability | Usability | **Efficiency** | Maintainability | Portability

Does the code consume too many physical resources when it executes (e.g. CPU, memory)?

CHEF

Resource Implementation v Custom Resource

```
directory '/srv/apache/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/apache/admins/ht
ml',
port: 8080)
  notifies :restart, 'service[httpd]'
end

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

```
apache_vhost 'admins' do
  site_port 8080
end
```

Functionality | Reliability | Usability | Efficiency | **Maintainability** | Portability

Are you able to easily adapt the solution? Is it testable?

CHEF

Resource Implementation v Custom Resource

```
directory '/srv/apache/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/apache/admins/ht
ml',
port: 8080)
  notifies :restart, 'service[httpd]'
end

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

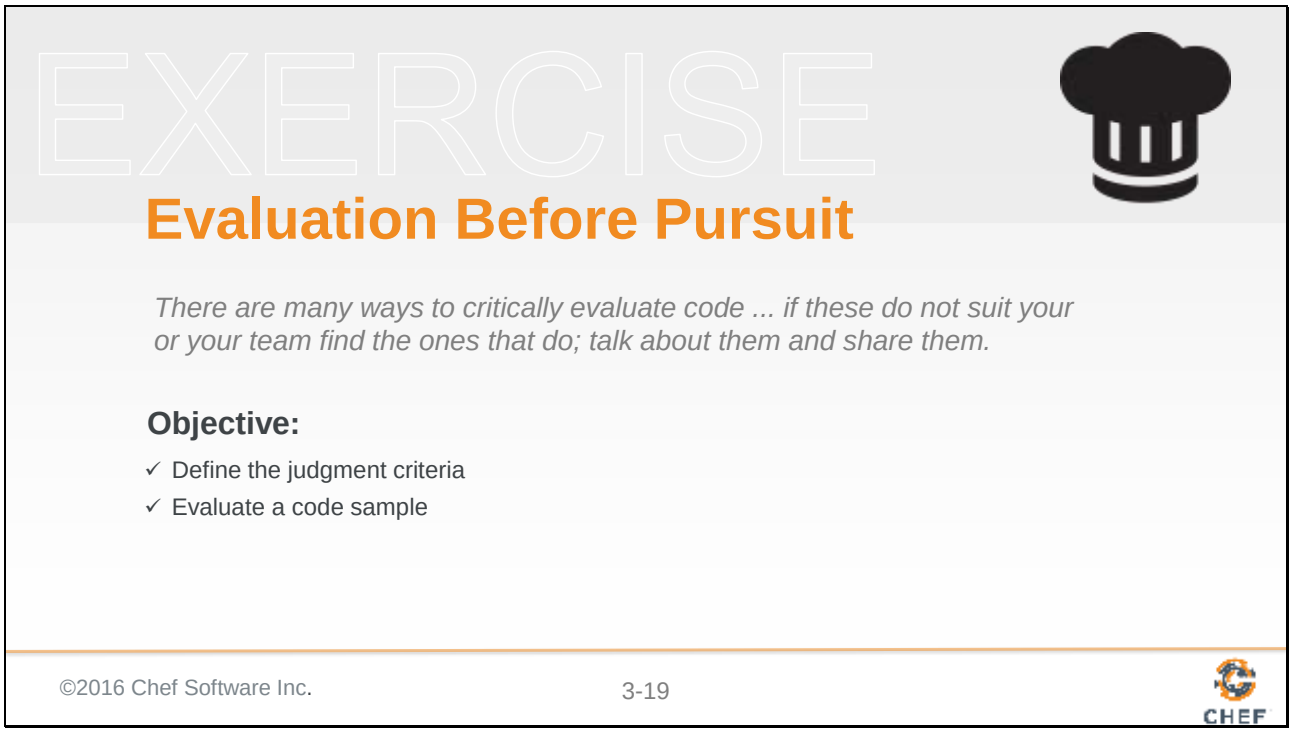
```
apache_vhost 'admins' do
  site_port 8080
end
```

Functionality | Reliability | Usability | Efficiency | Maintainability | **Portability**

Can the software adapt to changes in its environment? Or changes to its requirements?

CHEF

Slide 19



We've evaluated one code sample, let's look at a second one.

DISCUSSION

Discussion

What value does reviewing code for functionality, reliability, usability, efficiency, maintainability, portability bring?

CHEF

Slide 21

DISCUSSION

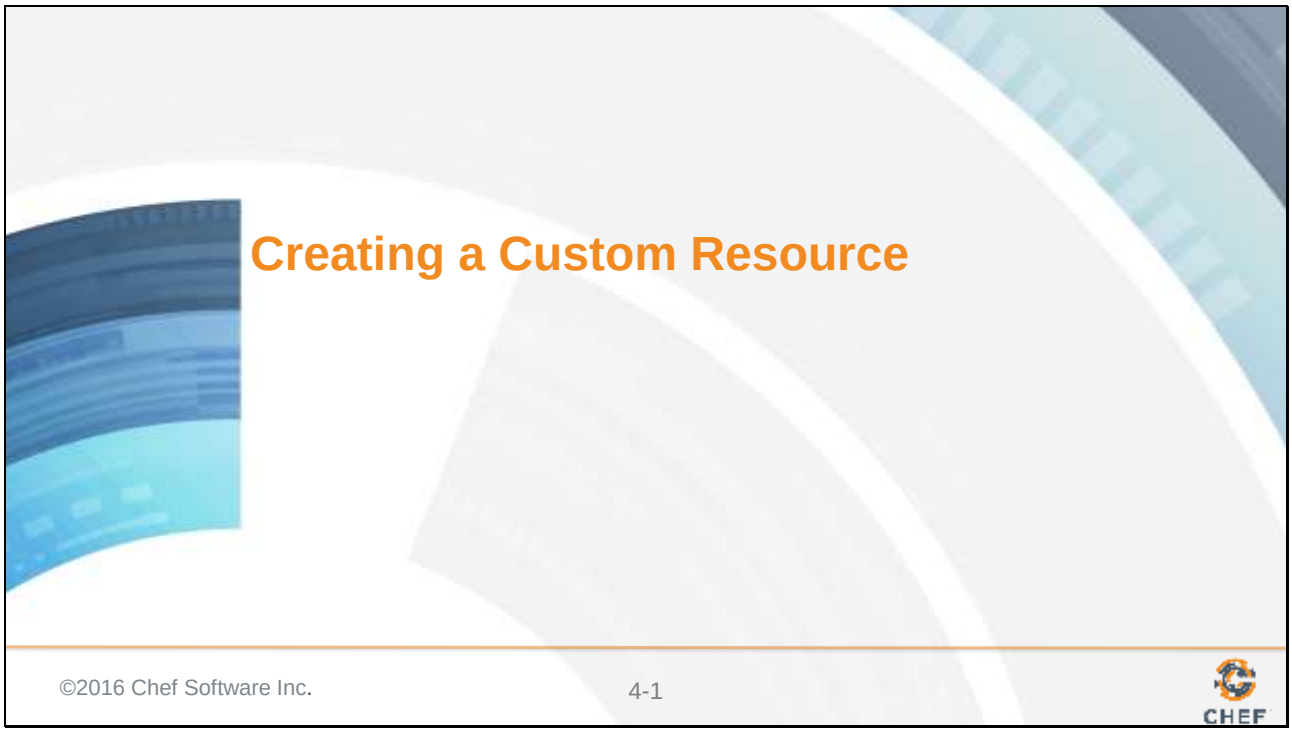
Q&A

What questions can we answer for you?

CHEF

Slide 22

4: Creating a Custom Resource



Slide 2

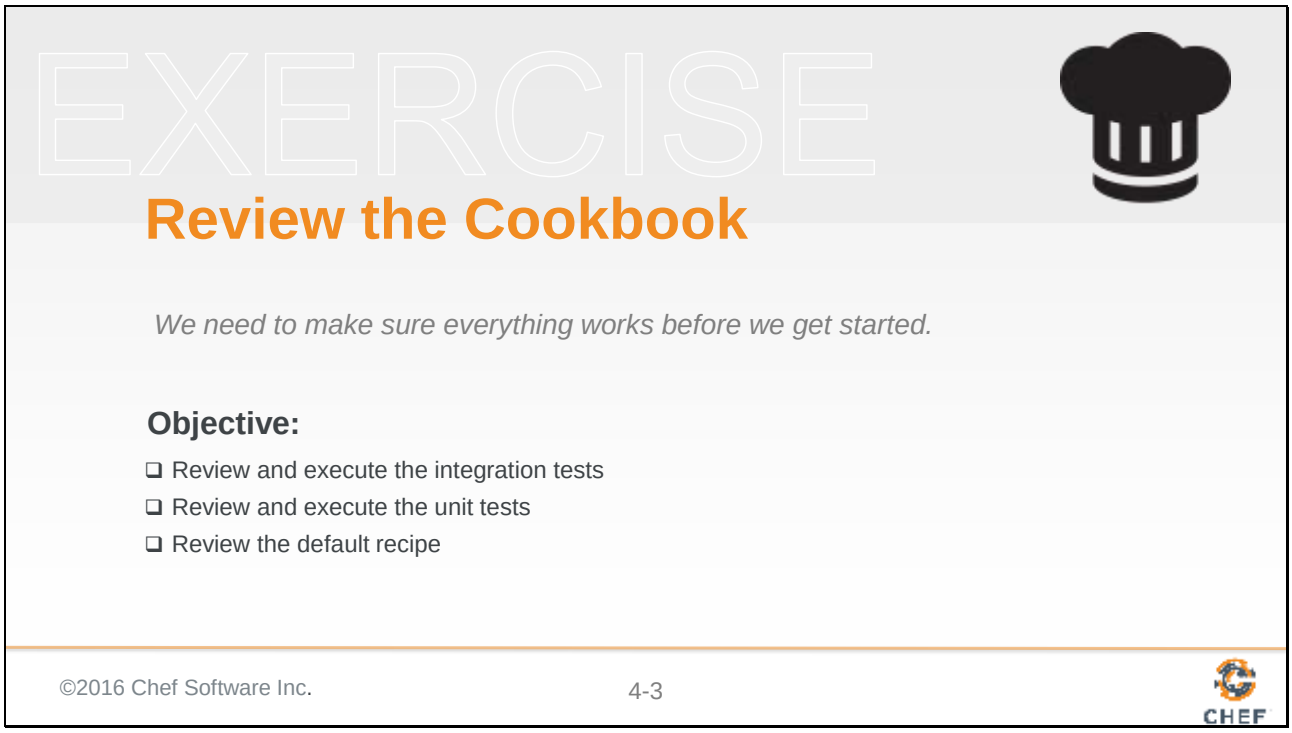
Objectives

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

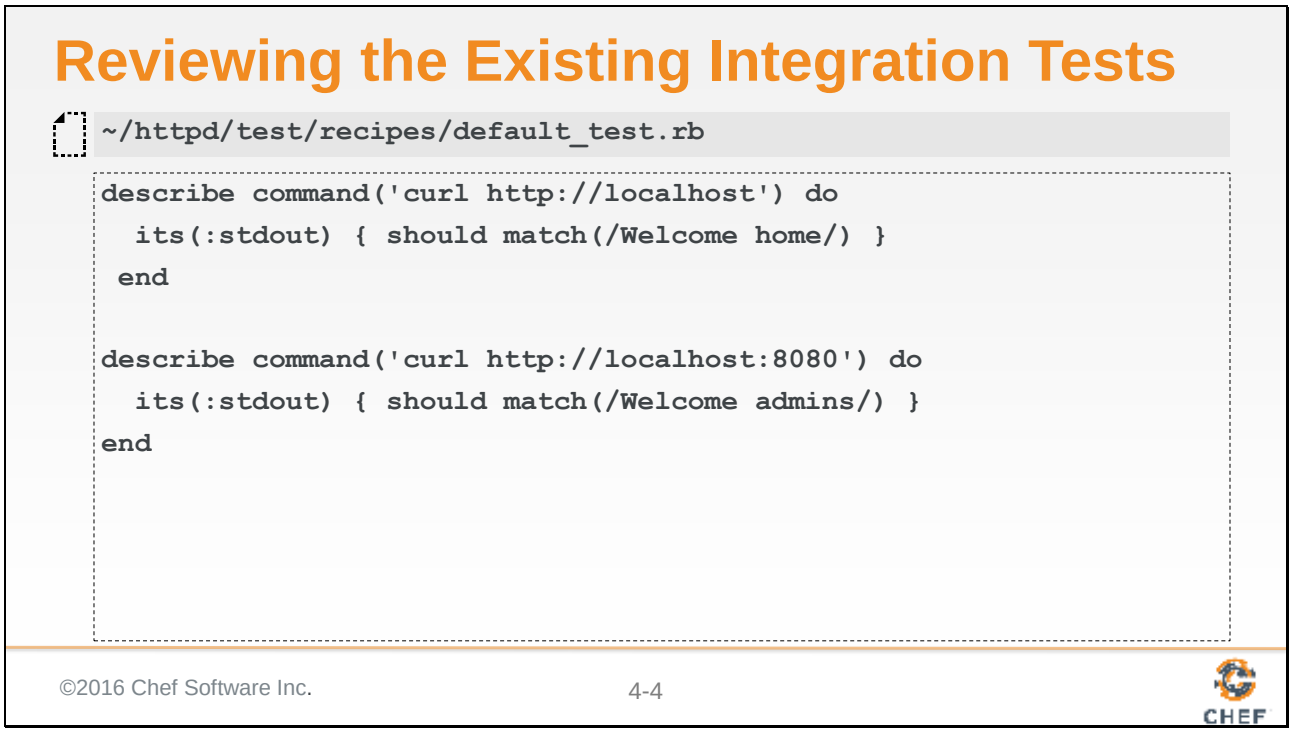
After completing this module you should be able to: create a custom resource file; define a custom resource action; and extract Chef resources into a custom resource action implementation.

Slide 3



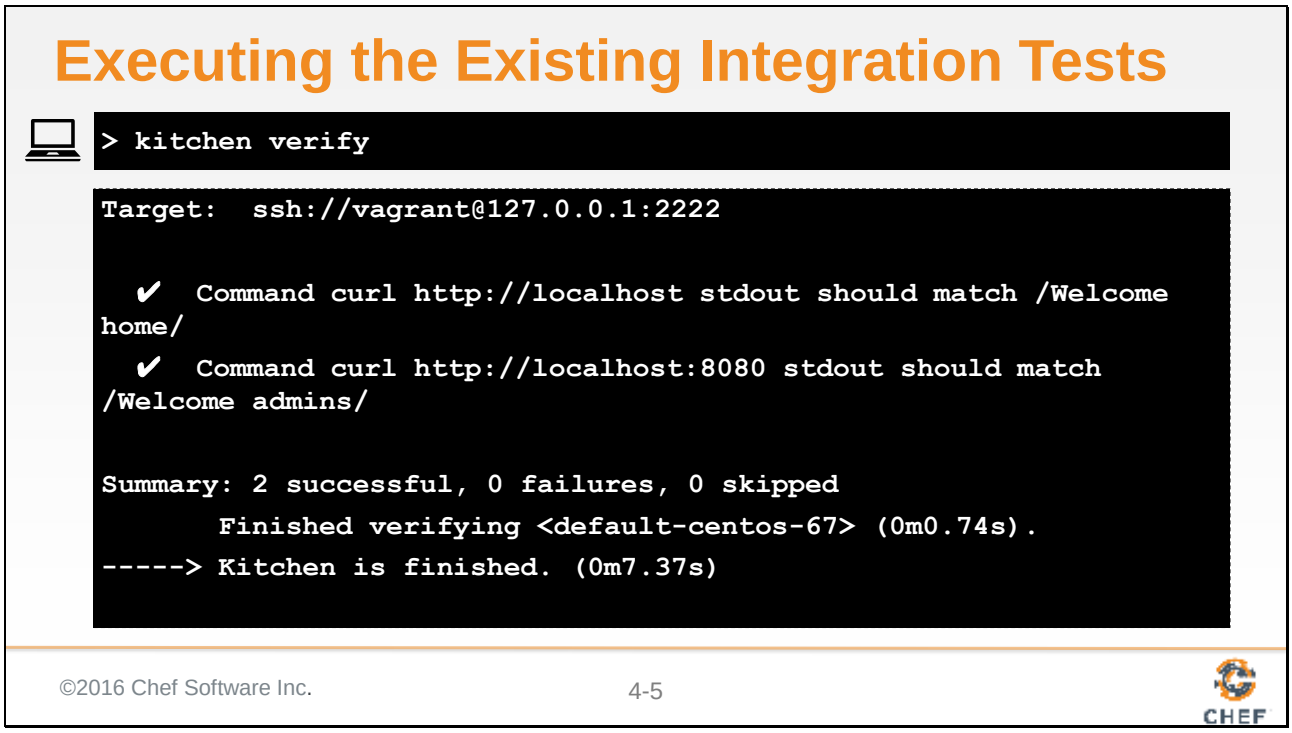
Before we begin creating this custom resource it is important to review the cookbook. We will start looking at the integration tests defined.

Slide 4



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.

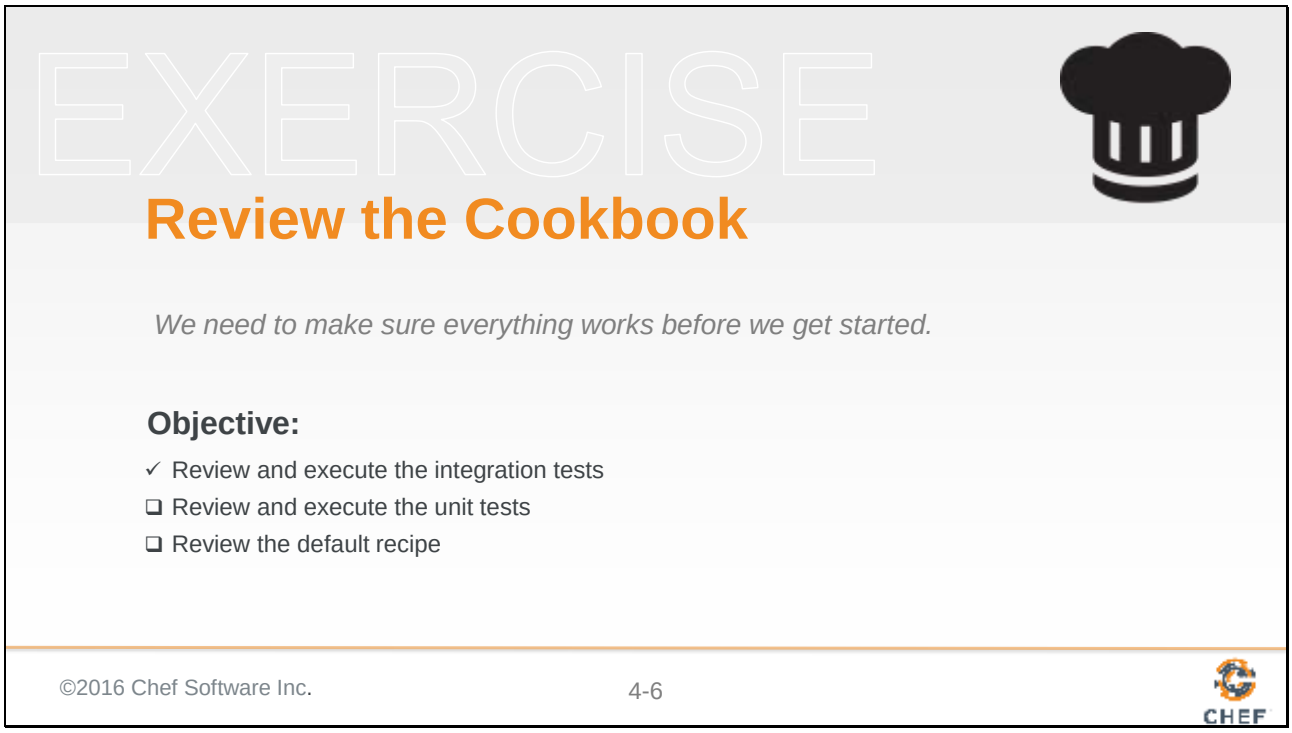
Slide 5



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.

Slide 6



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

Slide 7

Reviewing the Existing Unit Tests

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

```
require 'spec_helper'

describe 'httpd::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

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


CHEF

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.

Slide 8

Reviewing the Existing Unit Tests

~/httpd/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.

Slide 9

Reviewing the Existing Unit Tests

~/httpd/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.

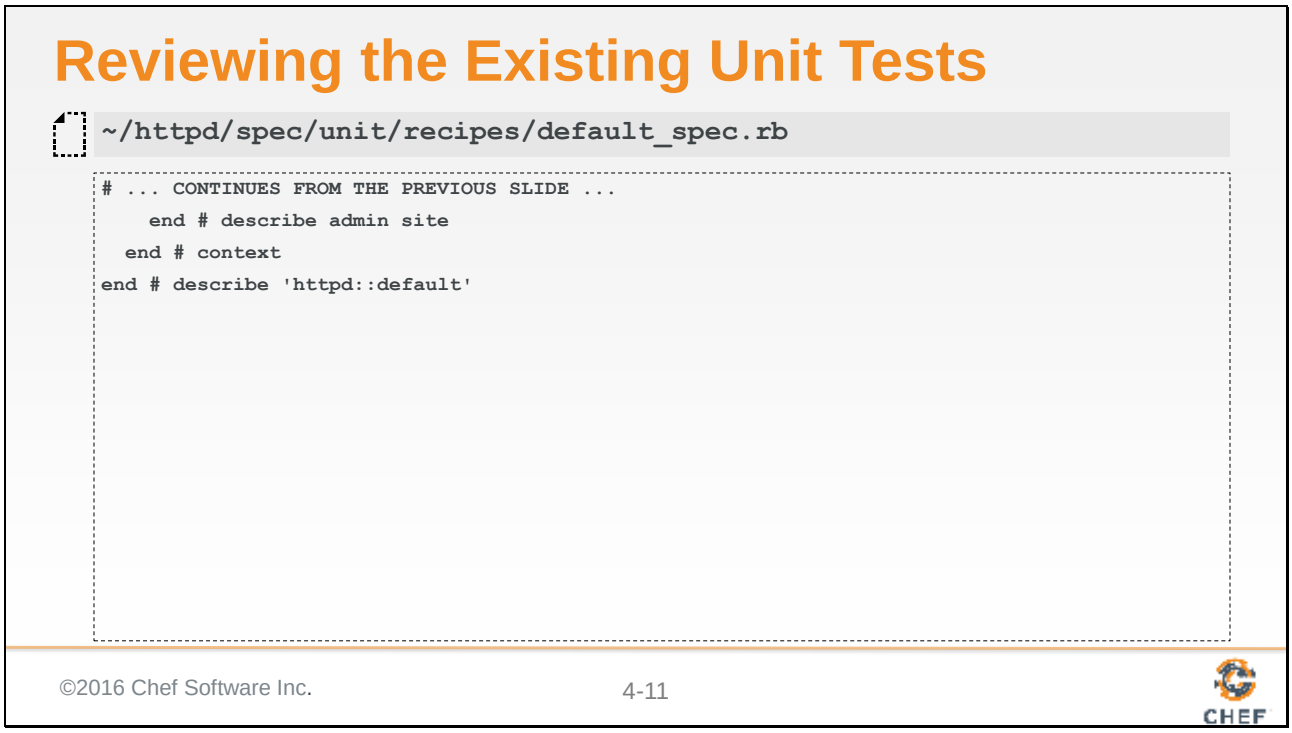
Slide 10

Reviewing the Existing Unit Tests

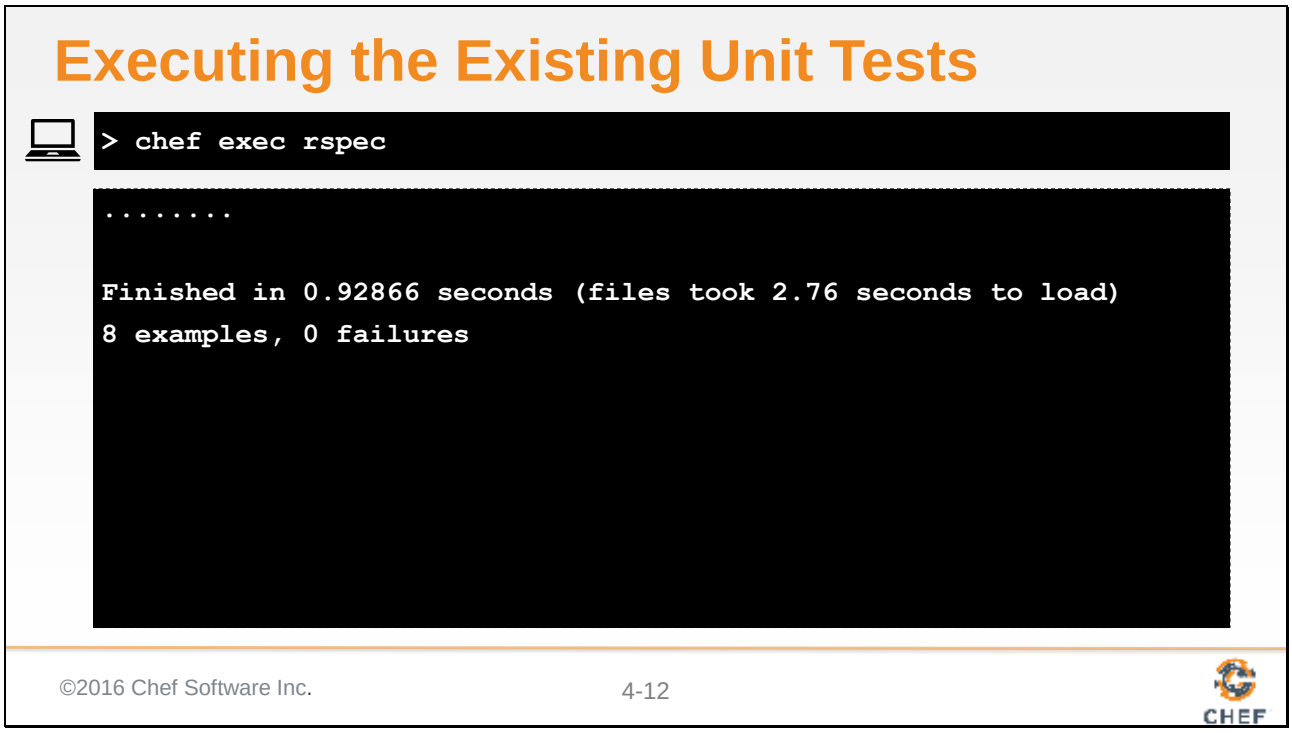
~/httpd/spec/unit/recipes/default_spec.rb

```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...
  describe 'for the admin site' do
    it 'creates the directory' do
      expect(chef_run).to create_directory('/srv/apache/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/apache/admins/html/index.html').wit ('<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.



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.



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

Slide 13

EXERCISE

Review the Cookbook

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

Objective:

- ✓ Review and execute the integration tests
- ✓ Review and execute the unit tests
- ❑ Review the default recipe

CHEF

Finally it is time to review the default recipe.

Slide 14

Reviewing the Default Recipe

~/httpd/recipes/default.rb

```
#
# Cookbook Name:: httpd
# Recipe:: default
#
# Copyright (c) 2016 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 ...
```

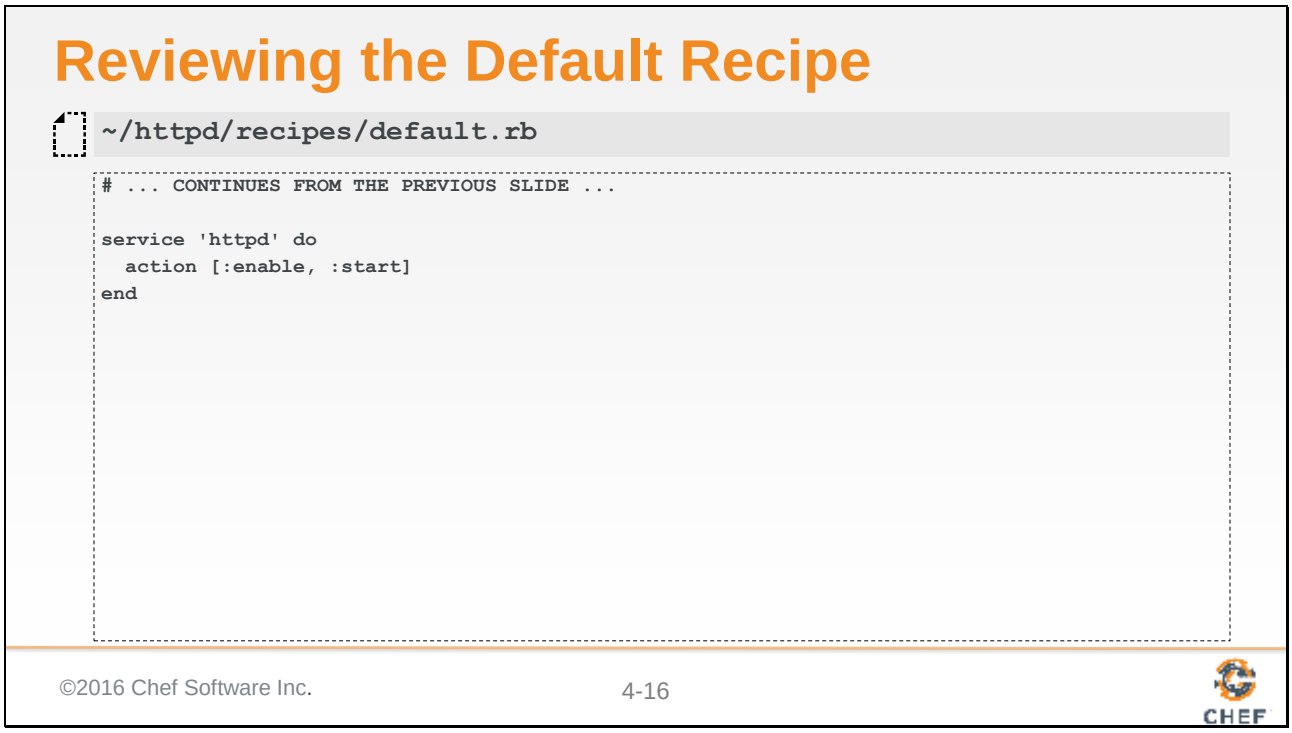

First we see that the recipe installs the necessary packages to install the web server. An html page is written out for the default site to contain the appropriate welcome message.

Reviewing the Default Recipe

~/httpd/recipes/default.rb

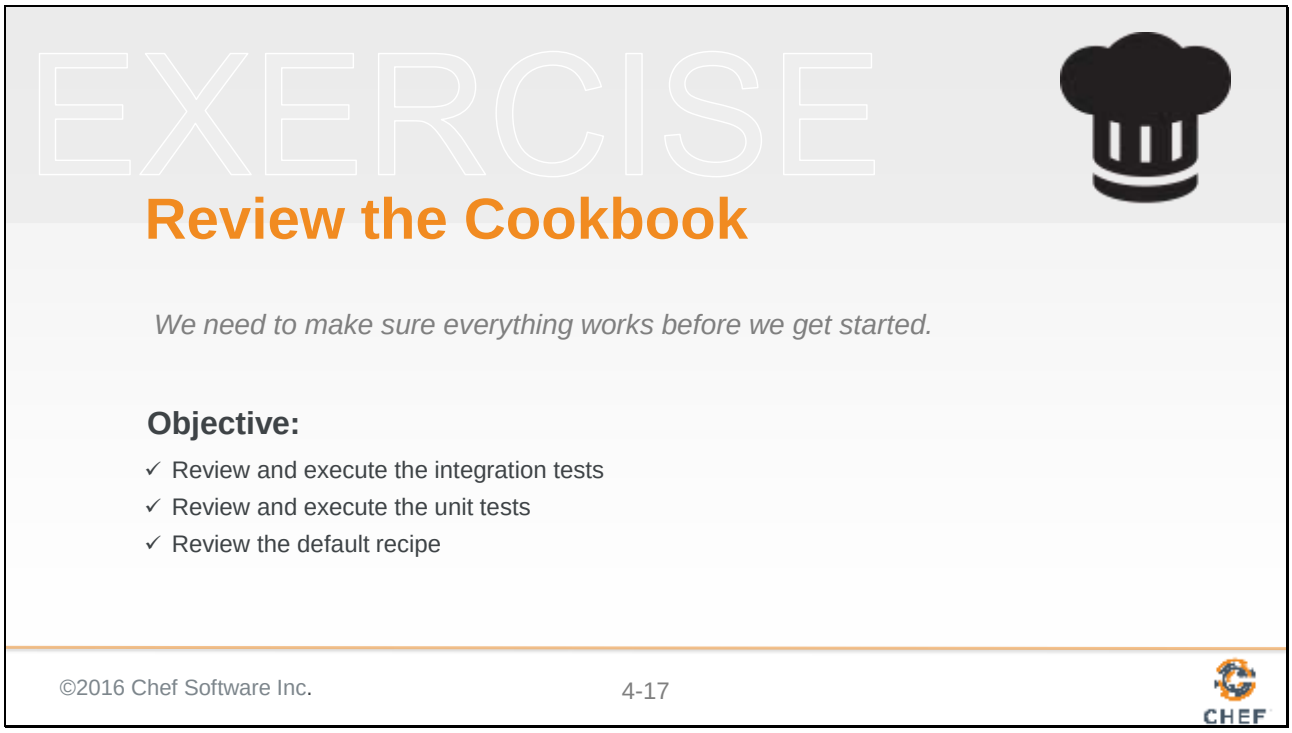
```
# ... CONTINUES FROM THE PREVIOUS SLIDE ...
directory '/srv/apache/admins/html' do
  recursive true
  mode '0755'
end

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

file '/srv/apache/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.



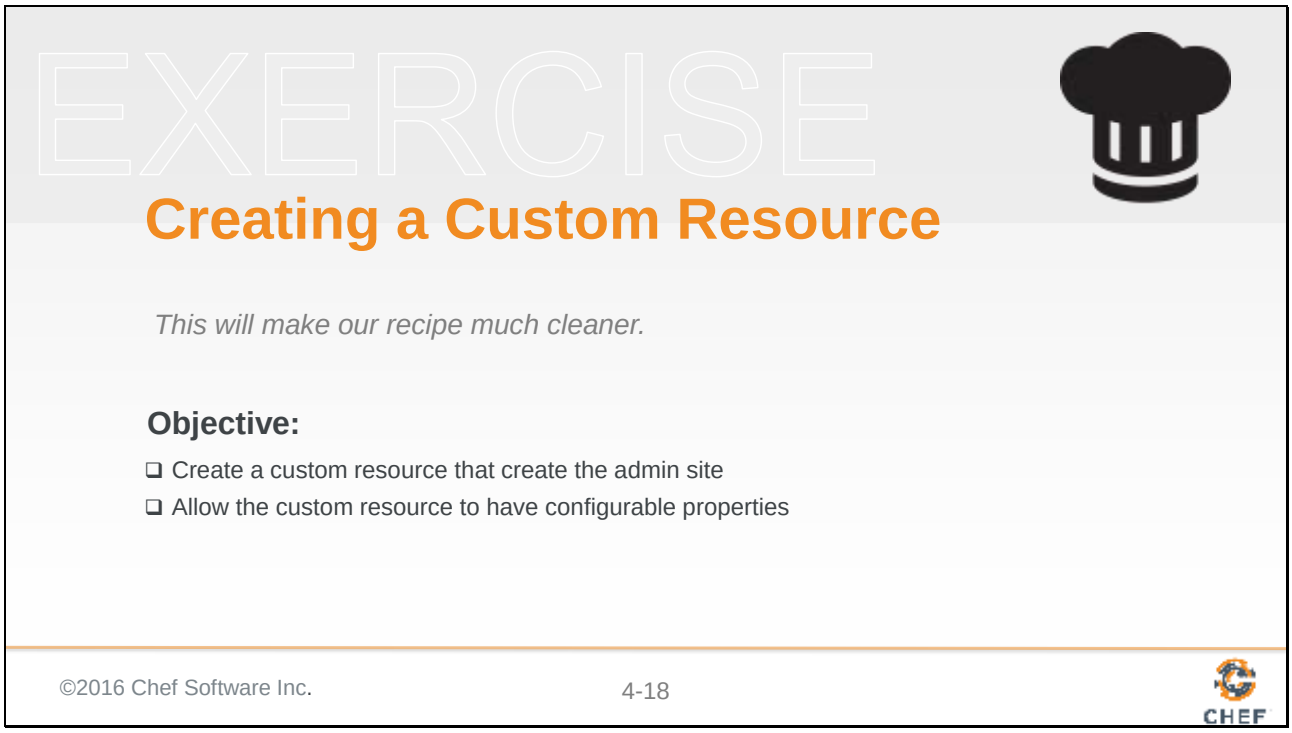
For the webserver to work correctly with the default site and the admin site the service needs to be started. We also enable the service to ensure the web server will start again if the instance this is being executed on happens to reboot.

Slide 17



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

Slide 18



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[/Users/franklinwebber/scratch/extending-cookbook/httpd/resources] action create
    - create new directory /Users/franklinwebber/scratch/extending-cookbook/httpd/resources
  * template[/Users/franklinwebber/scratch/extending-cookbook/httpd/resources/vhost.rb]
action create
    - create new file /Users/franklinwebber/scratch/extending-
cookbook/httpd/resources/vhost.rb
    - update content in file /Users/franklinwebber/scratch/extending-
cookbook/httpd/resources/vhost.rb from none to e3b0c4
    (diff output suppressed by config)
  * directory[/Users/franklinwebber/scratch/extending-cookbook/httpd/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.

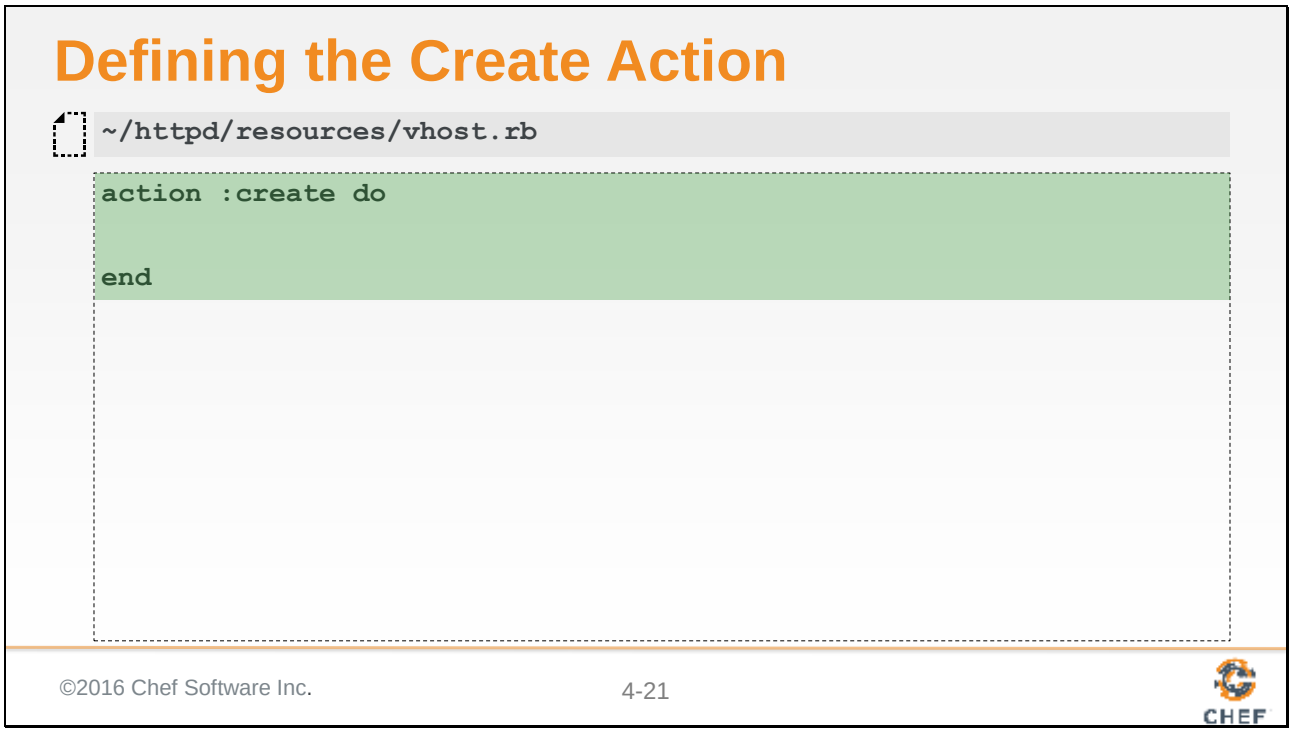
We call these multiple different sites, available on different ports, a virtual host. This is often abbreviated as 'vhost'. Create a custom resource with the name 'vhost'.

Slide 20

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.

Slide 21



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.

Slide 22

Implementing the Create Action

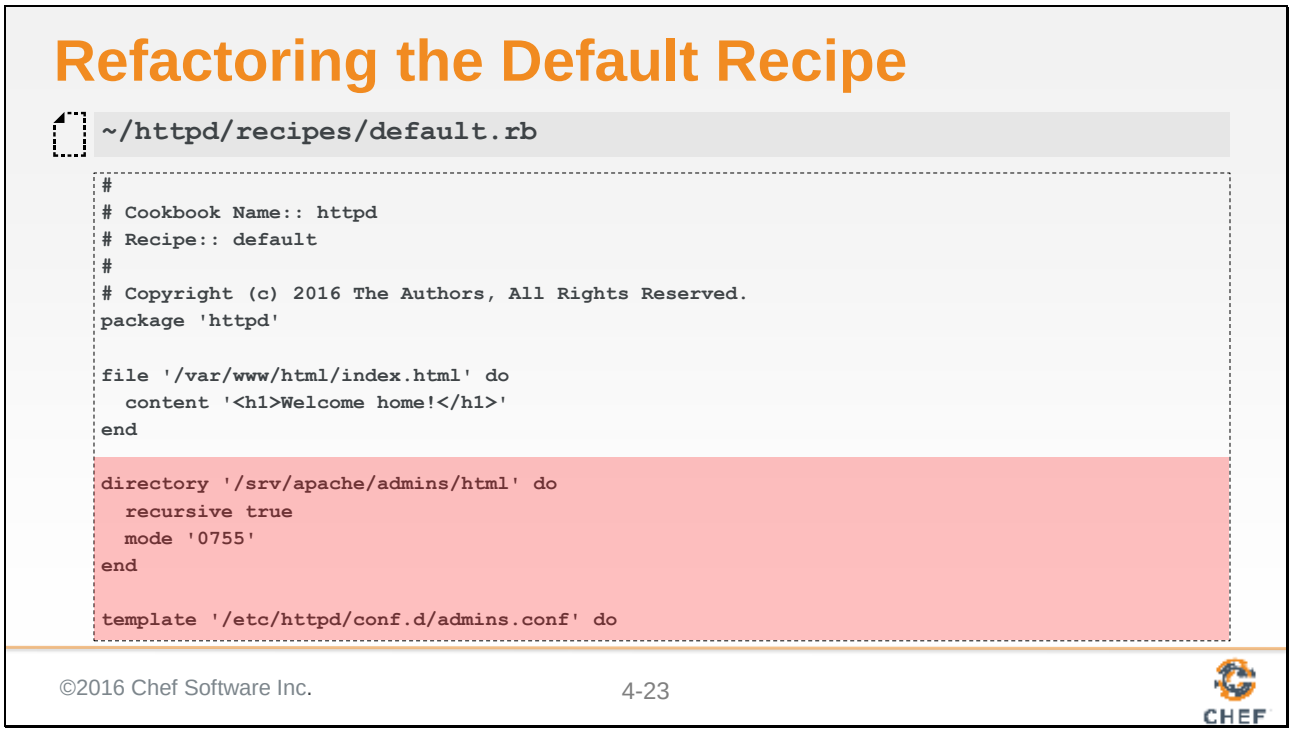
~/httpd/resources/vhost.rb

```
action :create do
  directory '/srv/apache/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/apache/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.

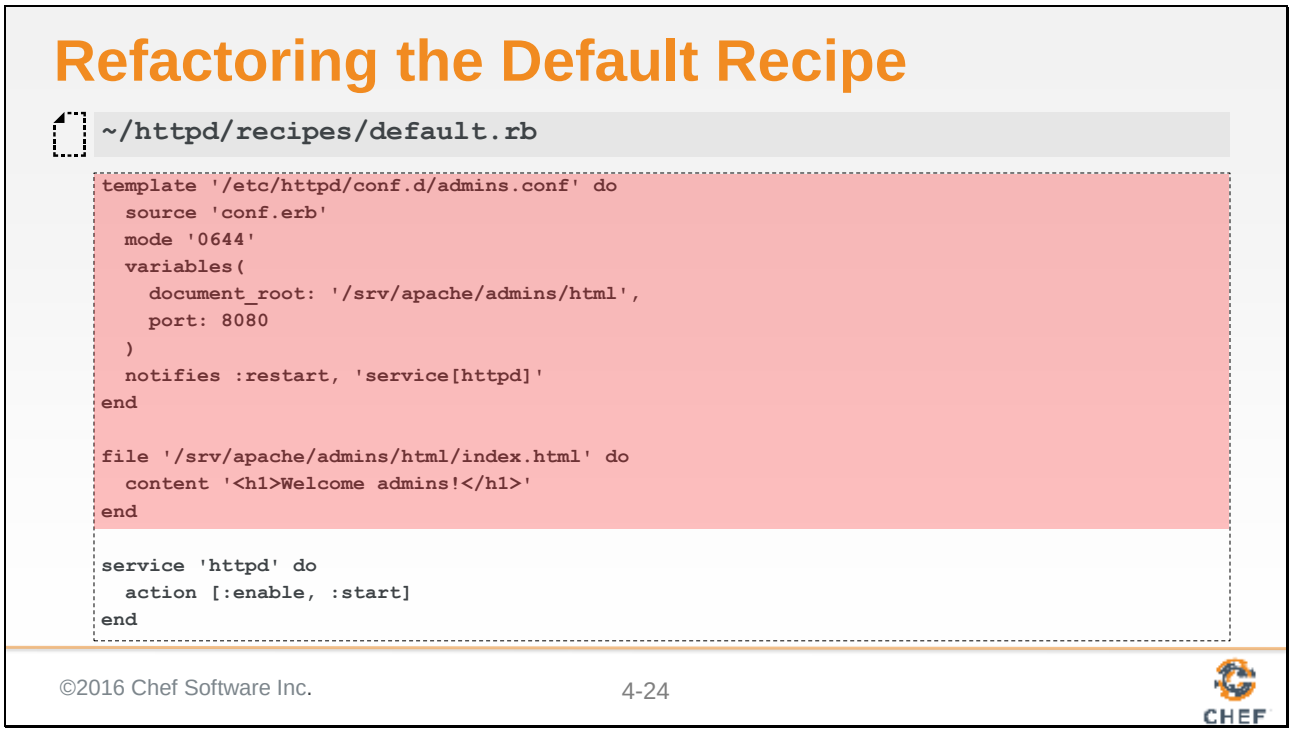
Slide 23



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.

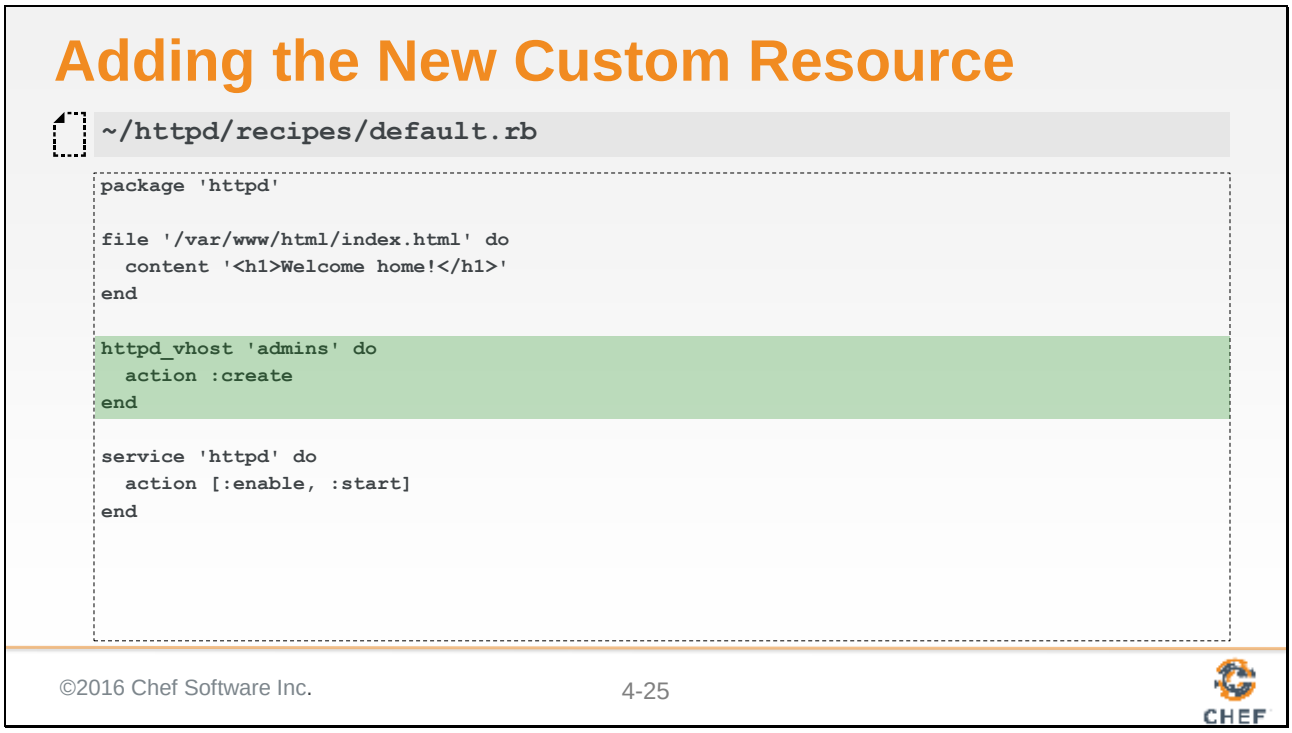
Remove the directory resource, the template resource, and the file resource that generate the admin site.

Slide 24



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

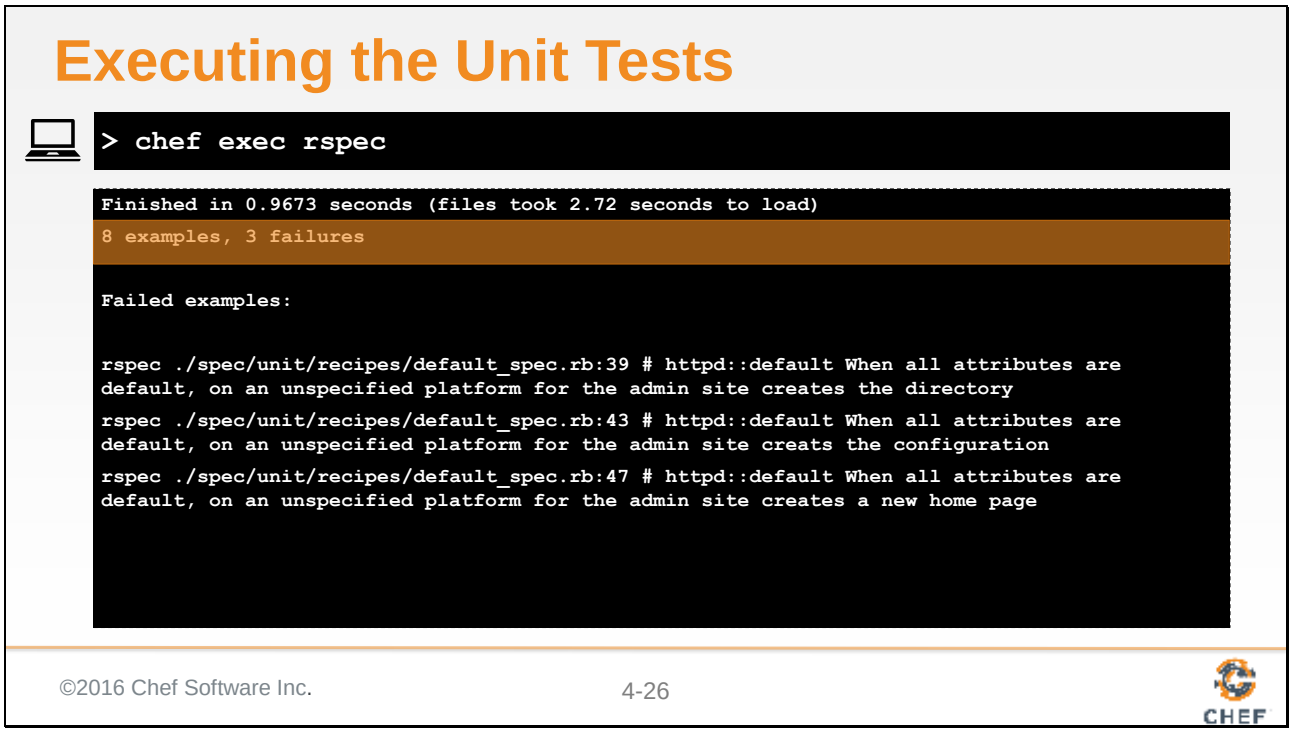
Slide 25



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 'httpd' and the ruby file is named 'vhost' so the default name for the resource is 'httpd_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.

Slide 26

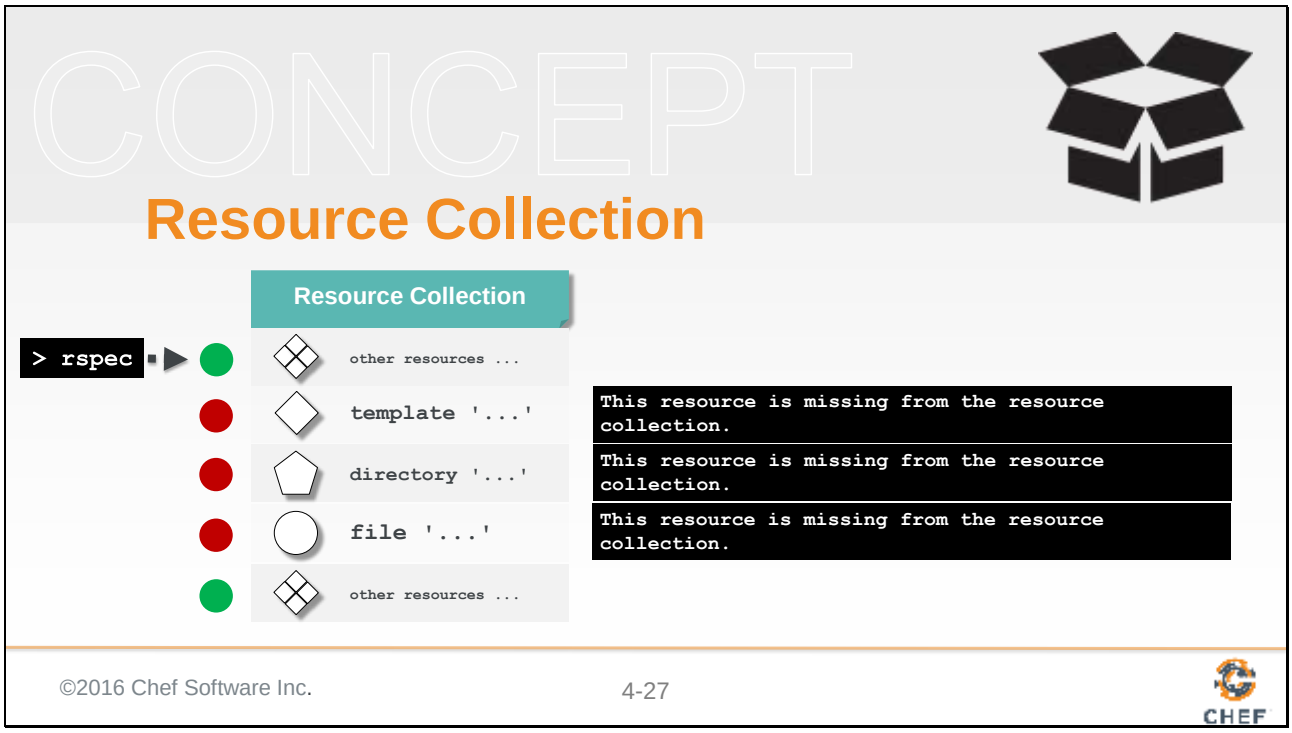


With the custom resource defined now within the default recipe it is time to run our unit tests to ensure that we have not broken our implementation.

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.

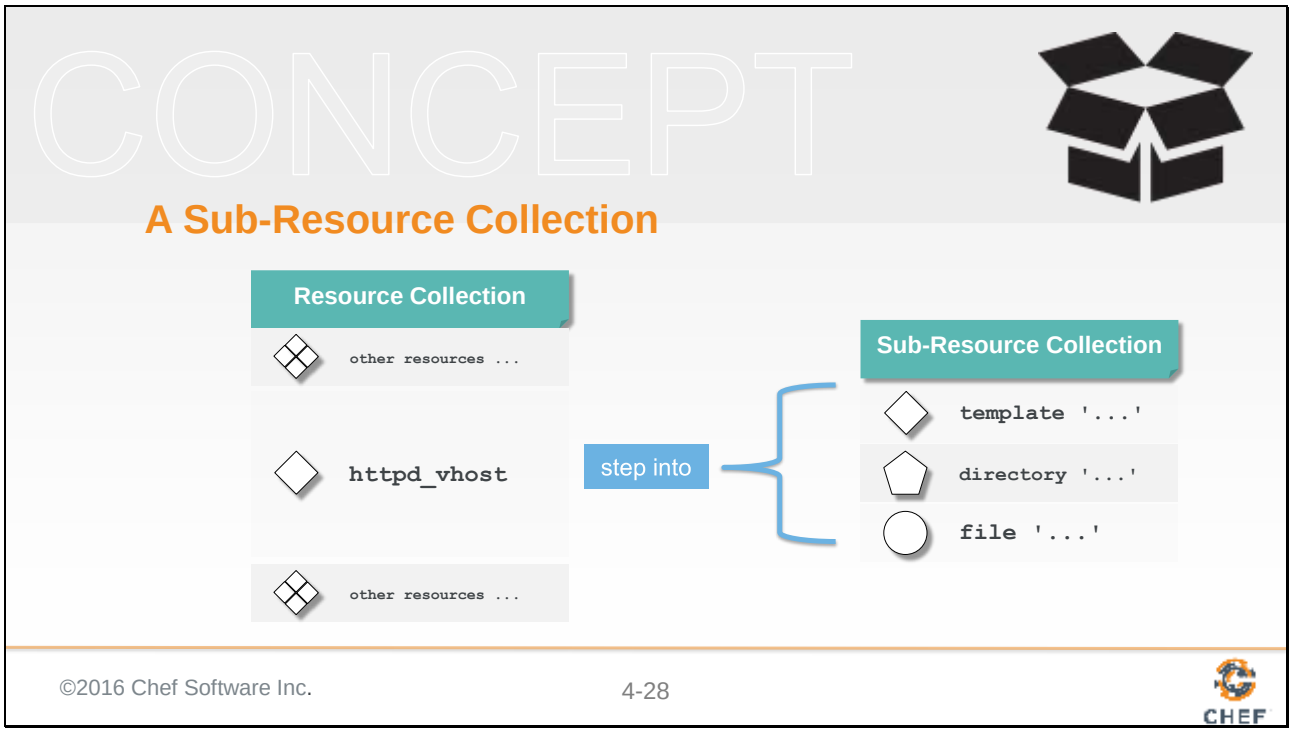
Slide 27



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.

Slide 28



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.

Slide 29

Updating the Unit Tests to Verify Custom Resource

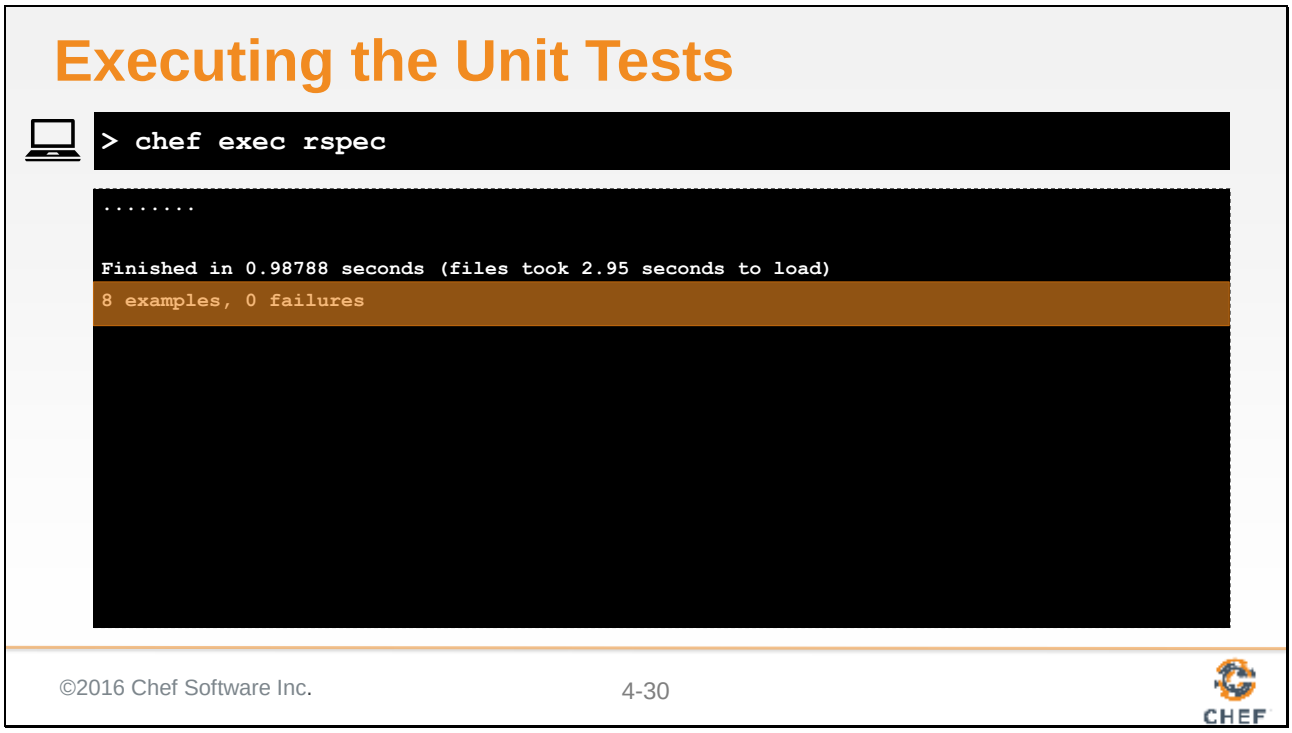
~/httpd/spec/unit/recipes/default_spec.rb

```
require 'spec_helper'

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

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

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

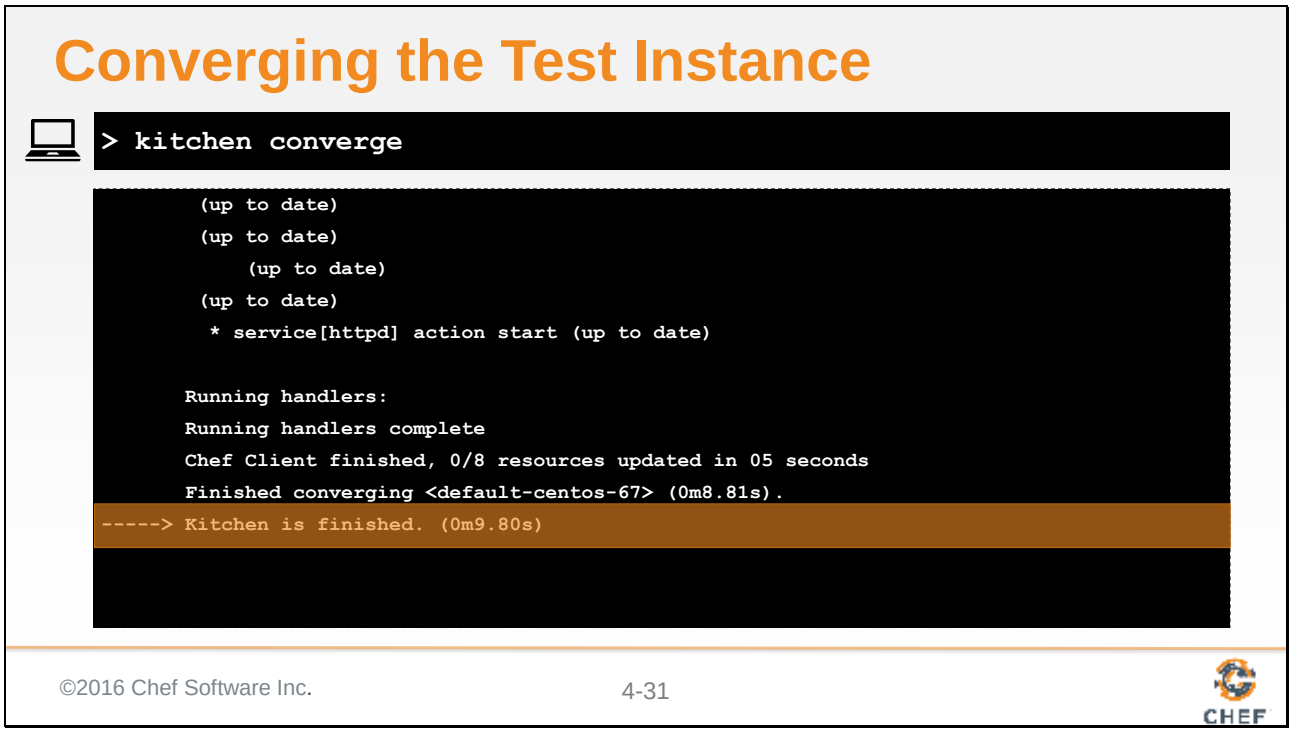

The unit tests fail because the resources defined within the custom resource are now no longer placed onto the resource collection. This is because the custom resource is placed on the resource collection and the resources internally within it are placed on a secondary resource collection that the custom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

Slide 30

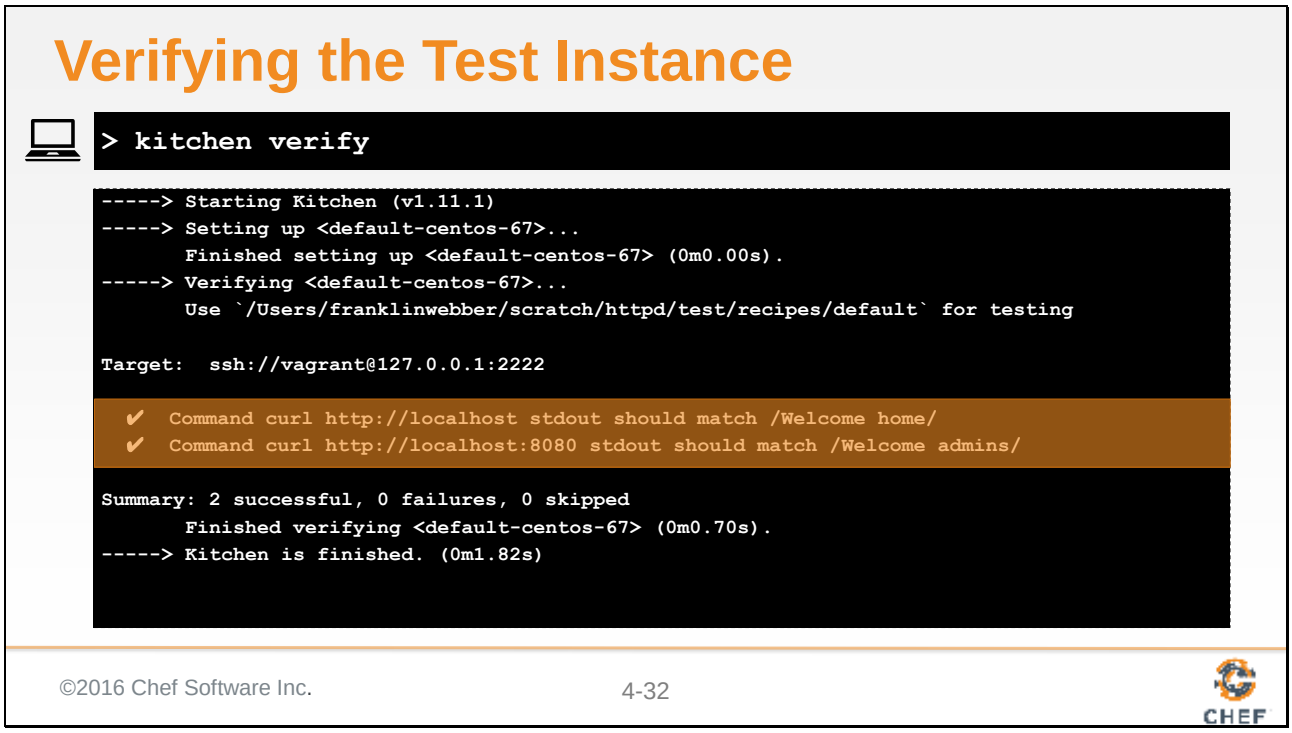


Running the unit tests again should show all the expectations have been met.



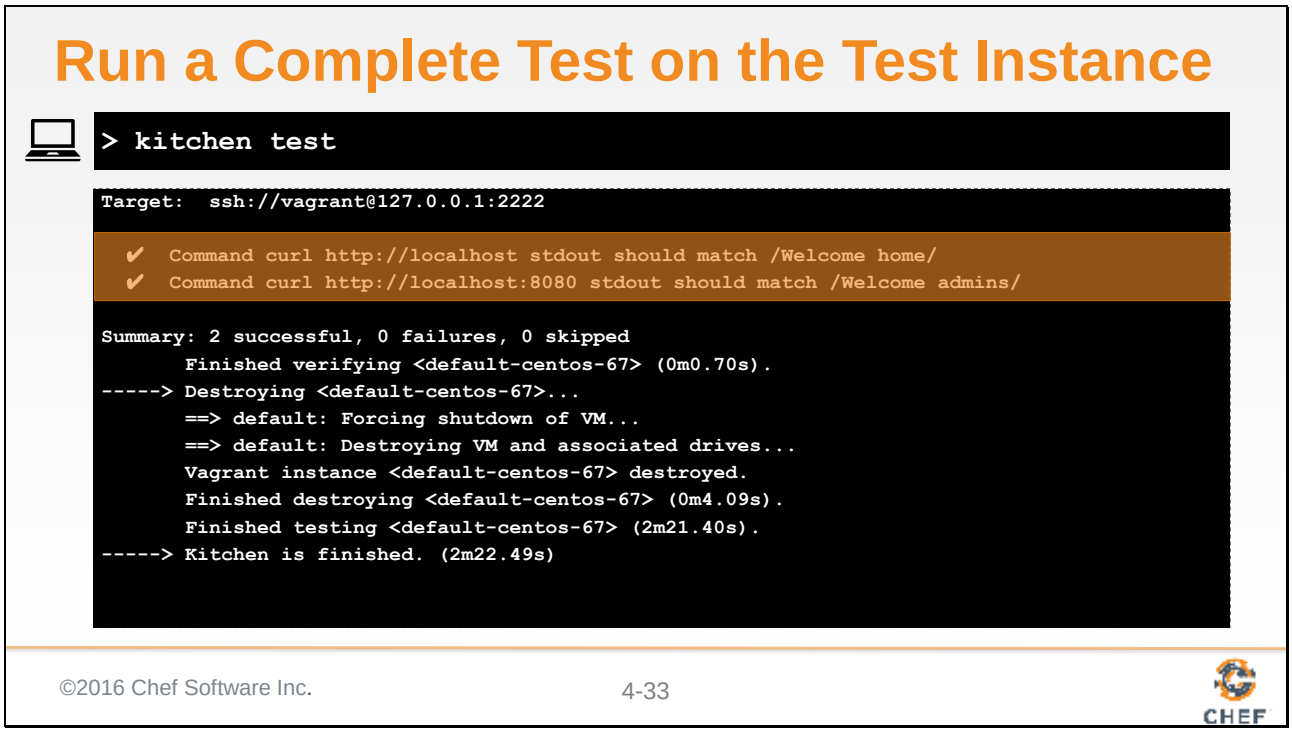
It is also important to execute the integration tests defined. First converging the test instance to ensure the recipe is defined correctly and converges successfully.

Slide 32



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

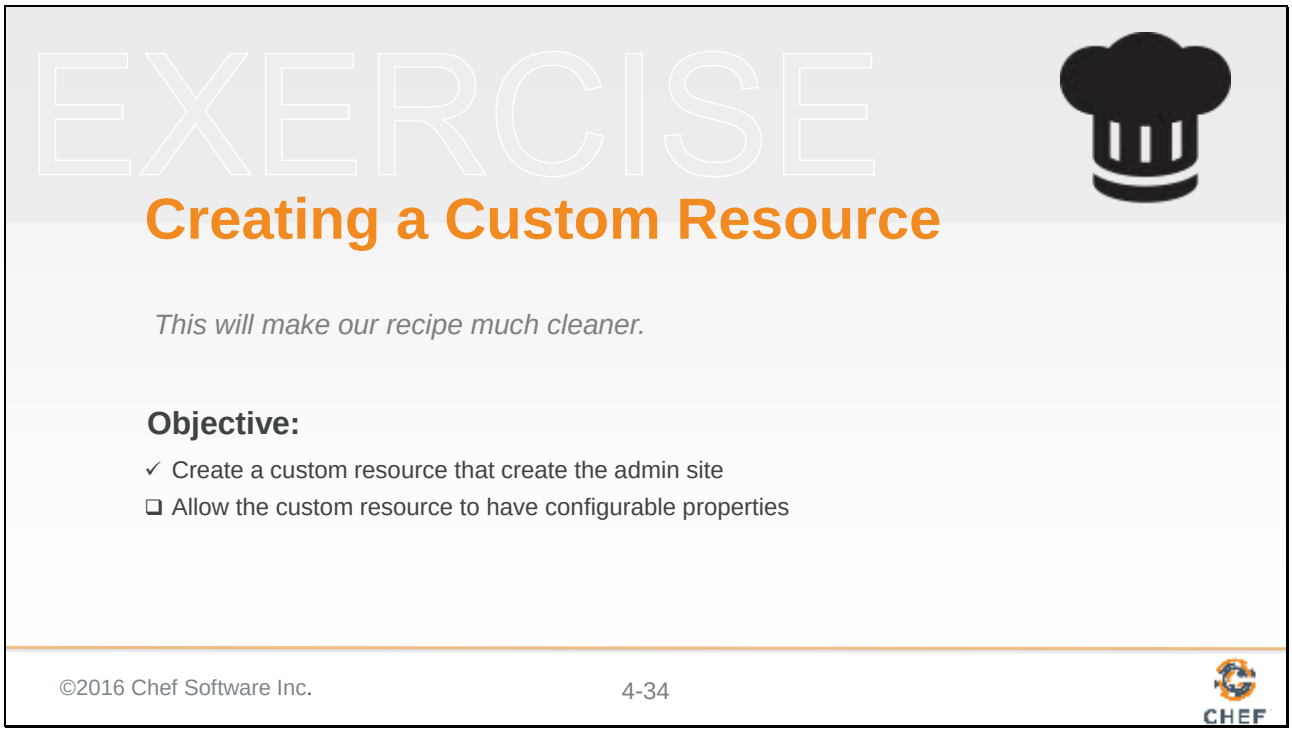
Slide 33



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.

Slide 34

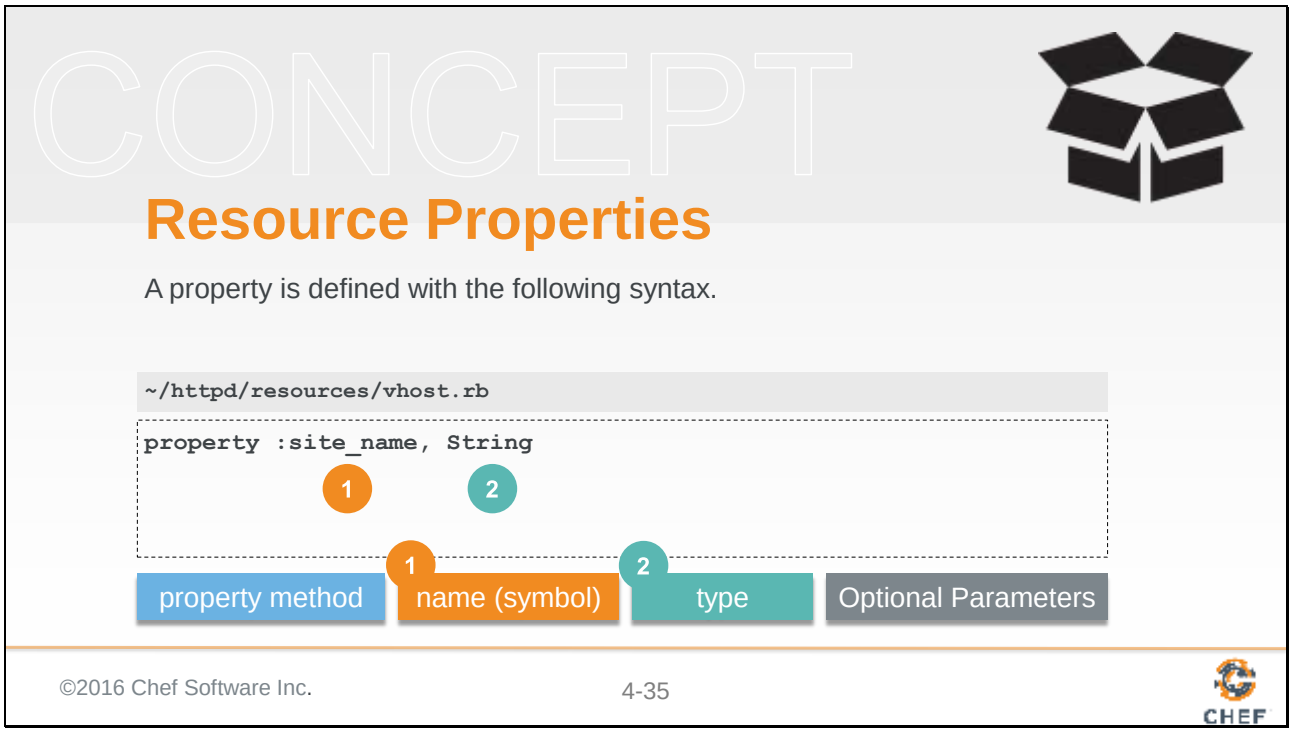


The initial implementation of the custom resource has been created and we have verified that it works by running our two test suites.

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.

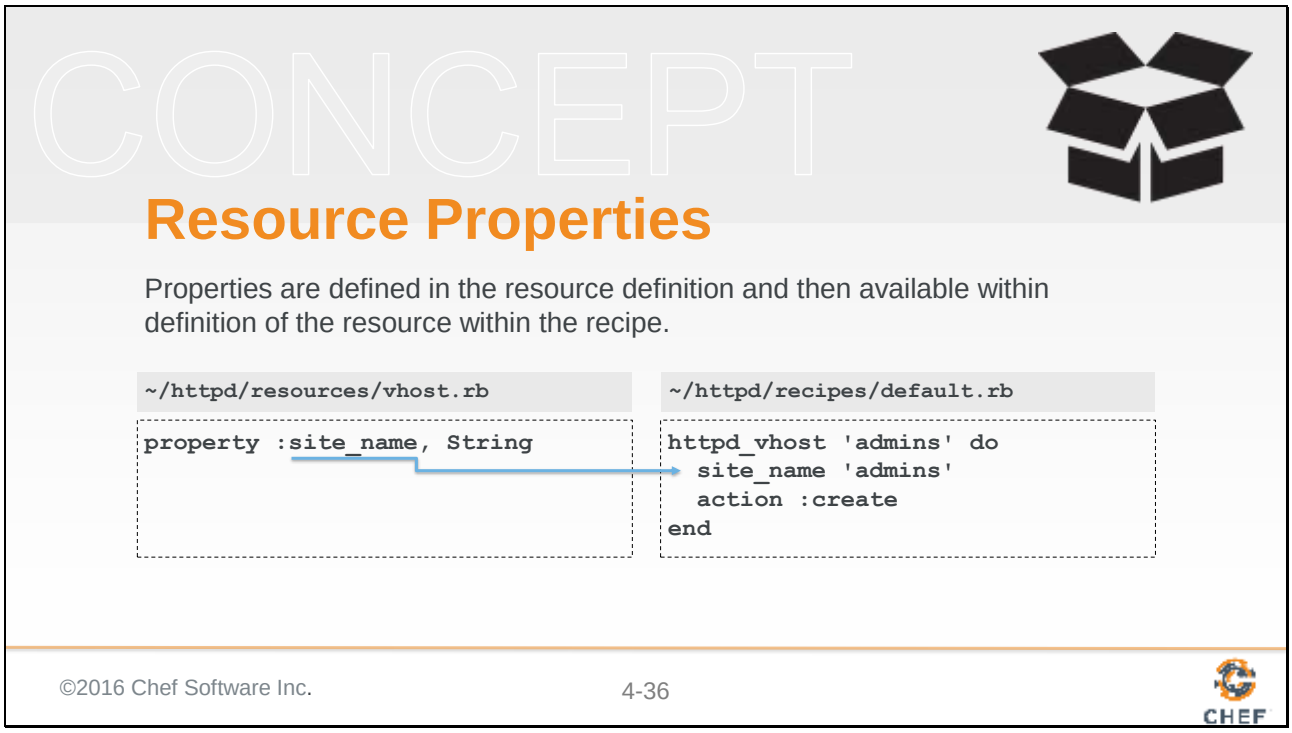
This can be done through properties that you defined on the custom resource.

Slide 35



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.

Slide 36



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

~/httpd/resources/vhost.rb

```
property :site_name, String

action :create do

  directory "/srv/apache/#{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/apache/#{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

~/httpd/resource/vhost.rb

```
action :create do
  directory "/srv/apache/#{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/apache/#{site_name}/html", port: 8080)
    notifies :restart, "service[httpd]"
  end

  file "/srv/apache/#{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.

Slide 39

Updating the Resource to use the Property

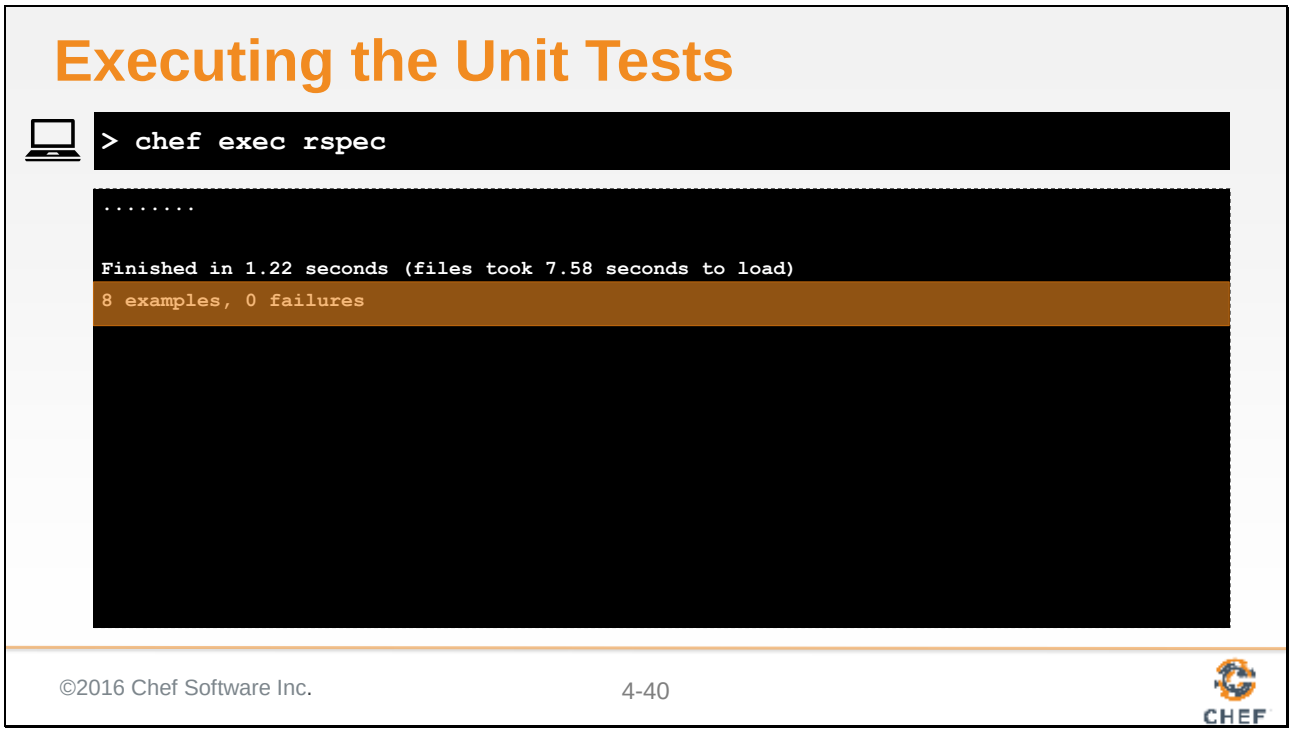
~/httpd/recipes/default.rb

```
package 'httpd'

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

httpd_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 propery to the custom resource implementation with the default recipe. In this case we are adding 'site_name' and specifying the value is 'admins'.

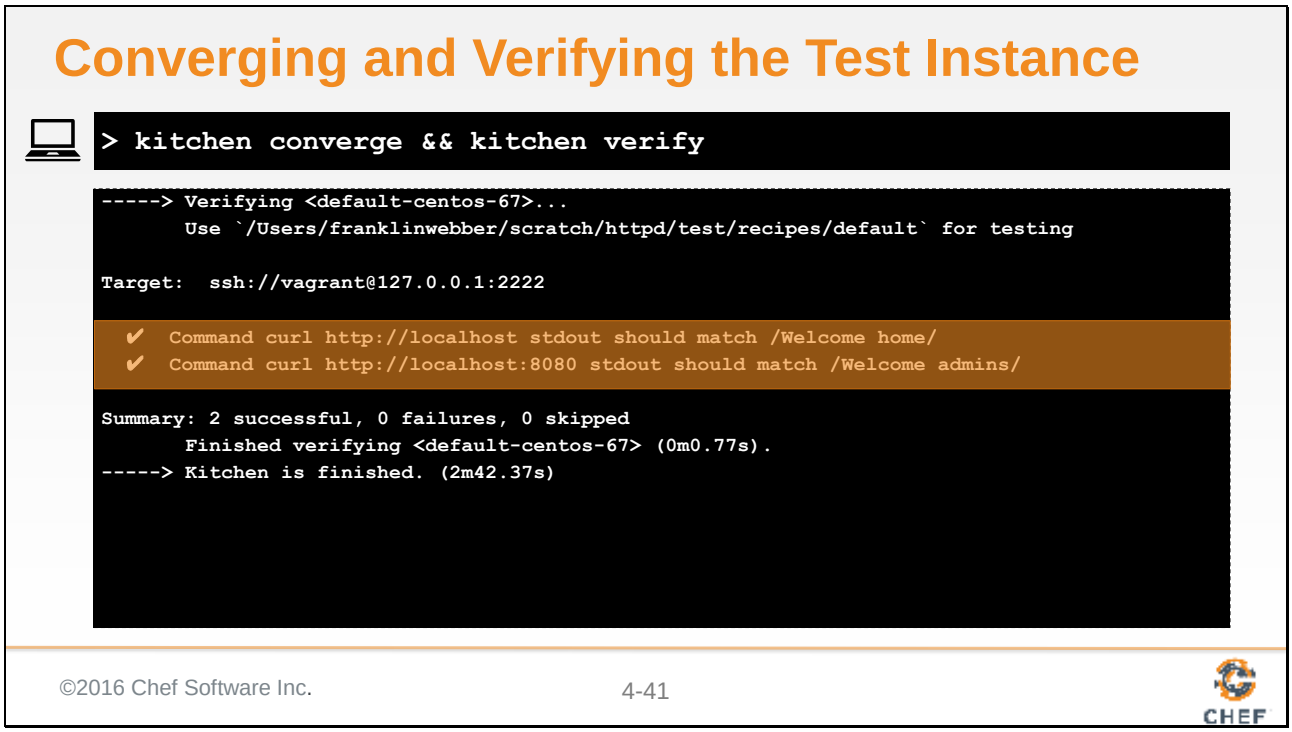
This means our implementation should be exactly the same as before but the details are now configurable through this property instead of being hard-coded to 'admin'.

Slide 40



The unit tests should pass when executed.

Slide 41



The integration tests should pass when executed.

Slide 42

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

CHEF

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

Slide 43

LAB

httpd_vhost - site_port Property

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

CHEF

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.

Defining a Property to Manage the site_port

~/httpd/resource/vhost.rb

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

action :create do
  directory "/srv/apache/#{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/apache/#{site_name}/html", port: site_port)
    notifies :restart, 'service[httpd]'
  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'.

Slide 45

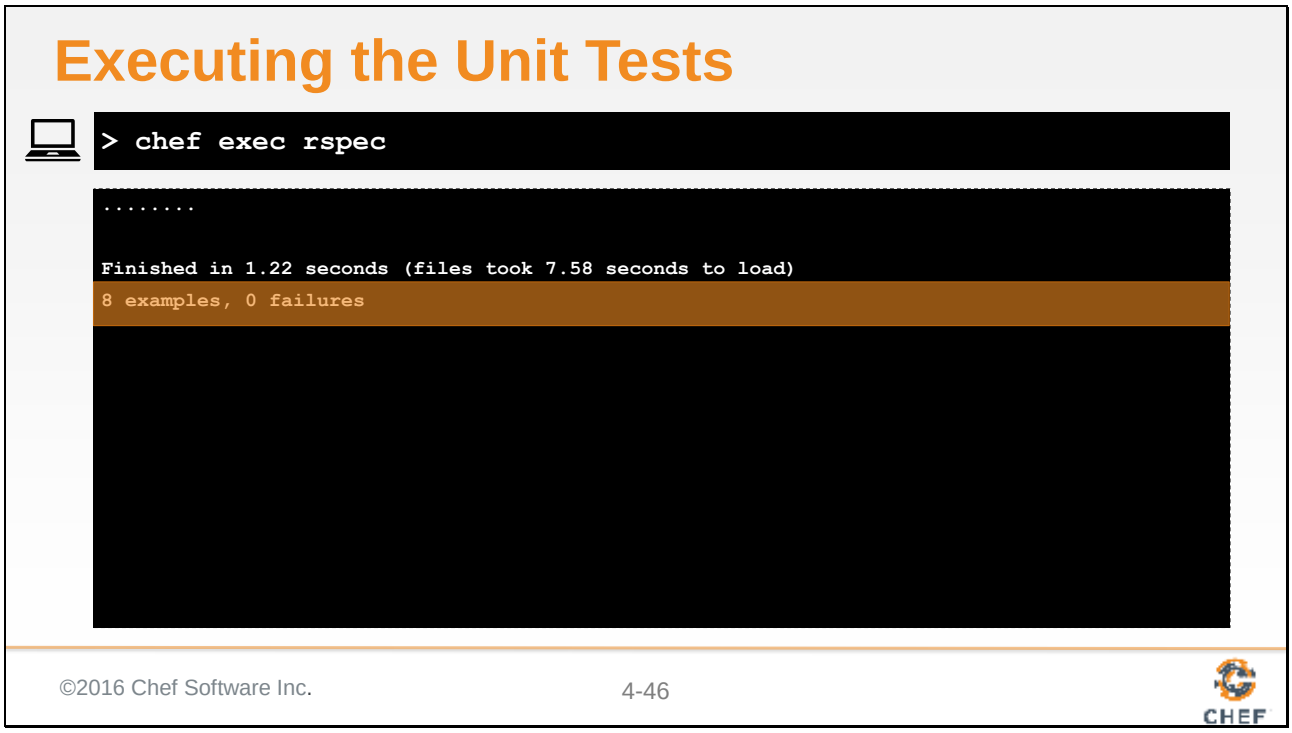
Updating the Resource to use the Property

~/httpd/recipes/default.rb

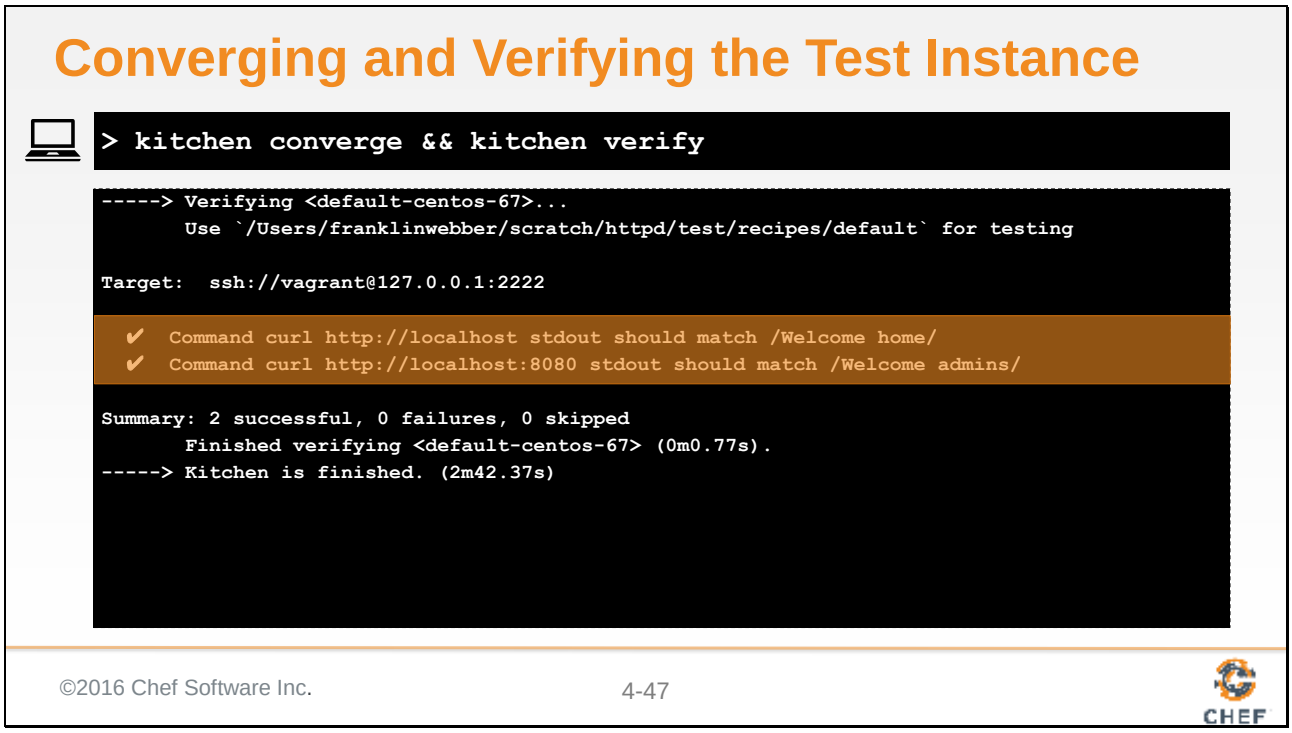
```
package 'httpd'

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

httpd_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 'httpd_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.

Slide 46



The unit tests should pass when executed.



The integration tests should pass when executed.

Slide 48

LAB

httpd_vhost - site_port Property

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

CHEF

With the site_port property developed the 'httpd_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.

Slide 49

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.

Slide 50

LAB

httpd_vhost Remove Action

- Define the 'remove' action for httpd_vhost that defines the policy:
  - **A directory named "/srv/apache/#{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 httpd_vhost resource named 'welcome' is removed**

CHEF

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.

Slide 51

Defining the Resource's Remove Action

~/httpd/resources/vhost.rb

```
# ... CREATE ACTION ...

action :remove do
  directory "/srv/apache/#{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.

Slide 52

Adding the Resource with Remove Action to the Recipe

~/httpd/recipes/default.rb

```
package 'httpd'

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

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

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


When httpd 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.

Slide 53

LAB

httpd_vhost Remove Action

- ✓ Define the 'remove' action for httpd_vhost that defines the policy:
  - **A directory named "/srv/apache/#{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 httpd_vhost resource named 'welcome' is removed**

CHEF

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

Slide 54

LAB

httpd_vhost Remove Action

- Update the default recipe's policy:
  - Remove the file resource named '/var/www/html/index.html'
  - Add an httpd_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 httpd_vhost resource named 'users'
- Update the InSpec tests to expect the default site to "Welcome users!"

CHEF

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.

Slide 55

Removing the Resource from the Recipe

~/httpd/recipes/default.rb

```
package 'httpd'

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

httpd_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 httpd deploys. Manipulating this resource is no longer important so we want to remove this resource.

Slide 56

Adding the Resource to create the users site

~/httpd/recipes/default.rb

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

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

httpd_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 and site_name to ensure we receive the correct message on the correct port.

Slide 57

Removing the Un-needed Unit Test Expectation

~/httpd/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

# ... EXAMPLES DEFINED BELOW ...
```


CHEF

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

Slide 58

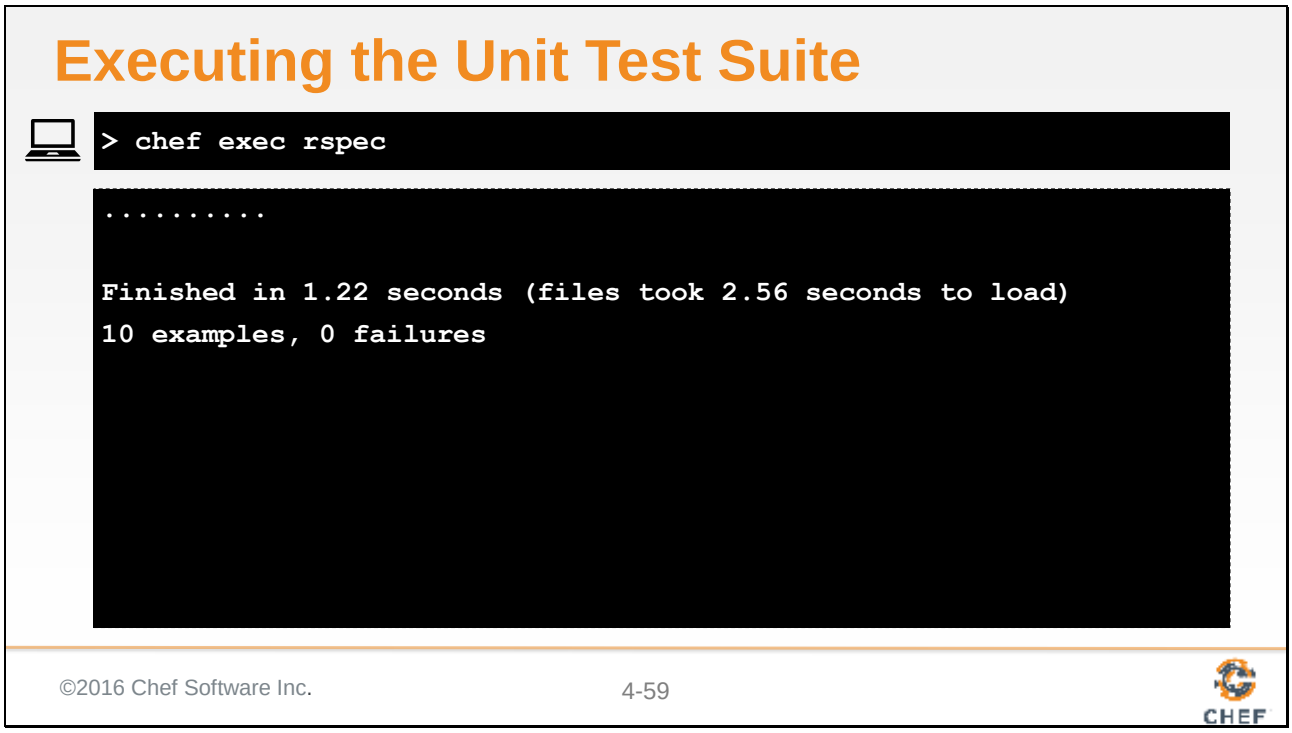
Adding Expectations for the users Site

~/httpd/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/apache/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/apache/users/html/index.html').with_content('<h1>Welcome users!</h1>')
    end
  # ... EXAMPLES DEFINED ABOVE ...
```

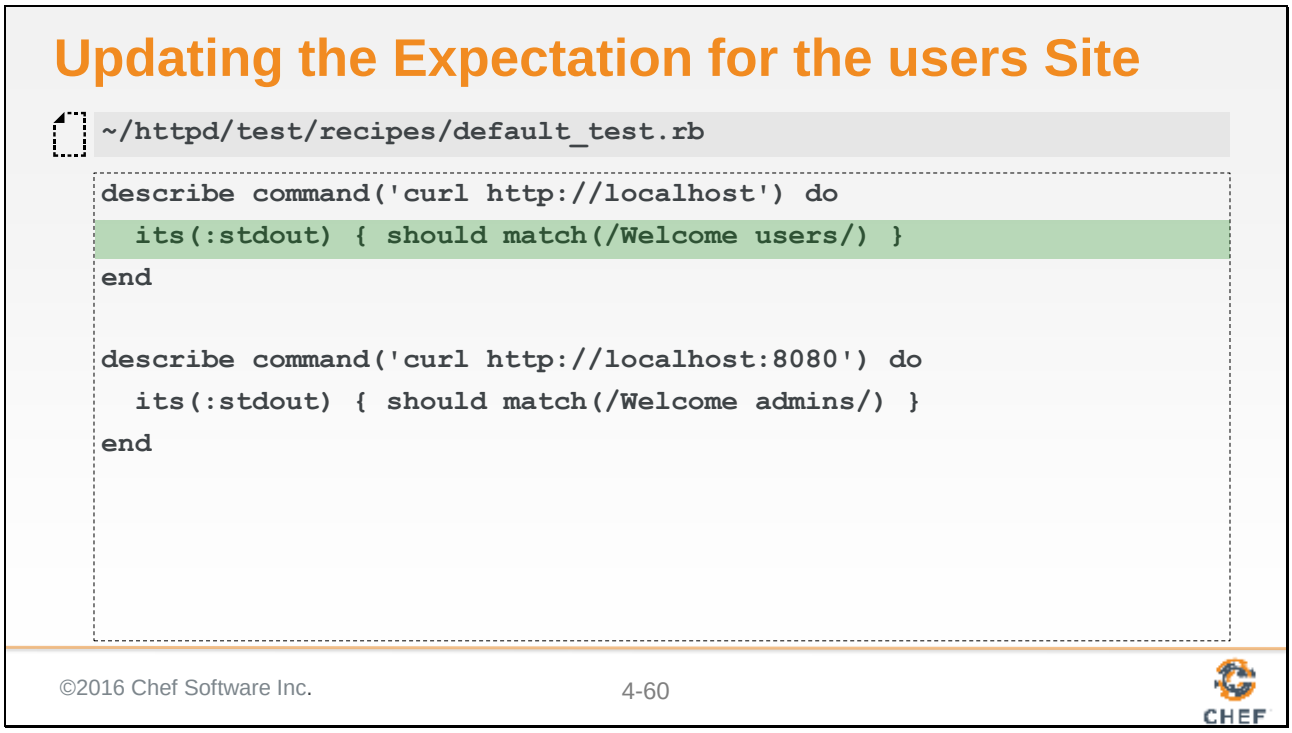

And add a new series of expectations that are very similar to the admins site. We want to ensure that the following are created: a directory to store the html; a configuration file for users; and a new html file that contains a message for the users.

Slide 59



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

Slide 60



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

Slide 61

Converging and Verifying the Test Instance

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

```
-----> Verifying <default-centos-67>...
       Use `/Users/franklinwebber/scratch/httpd/test/recipes/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.

Slide 62

LAB

httpd_vhost Remove Action

- ✓ Update the default recipe's policy:
  - Remove the file resource named '/var/www/html/index.html'
  - Add an httpd_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 httpd_vhost resource named 'users'
- ✓ Update the InSpec tests to expect the default site to "Welcome users!"

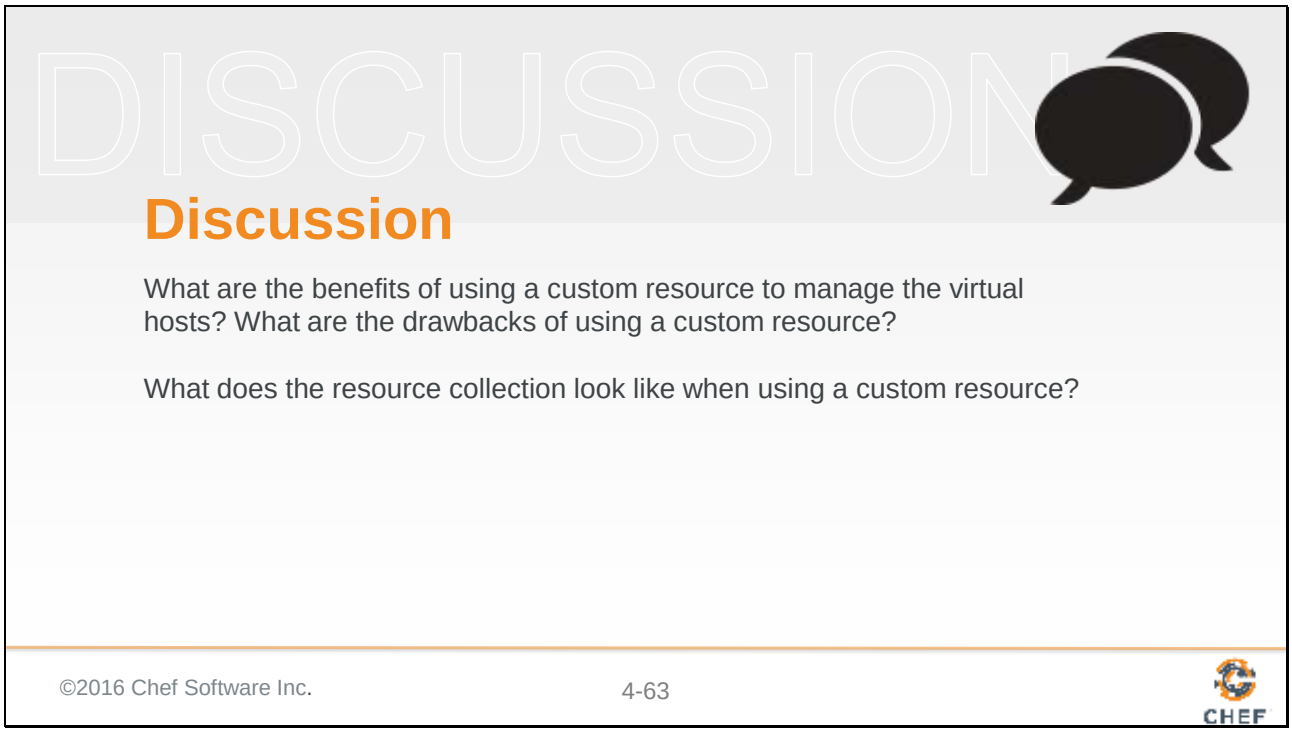
CHEF

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.

.

Slide 63



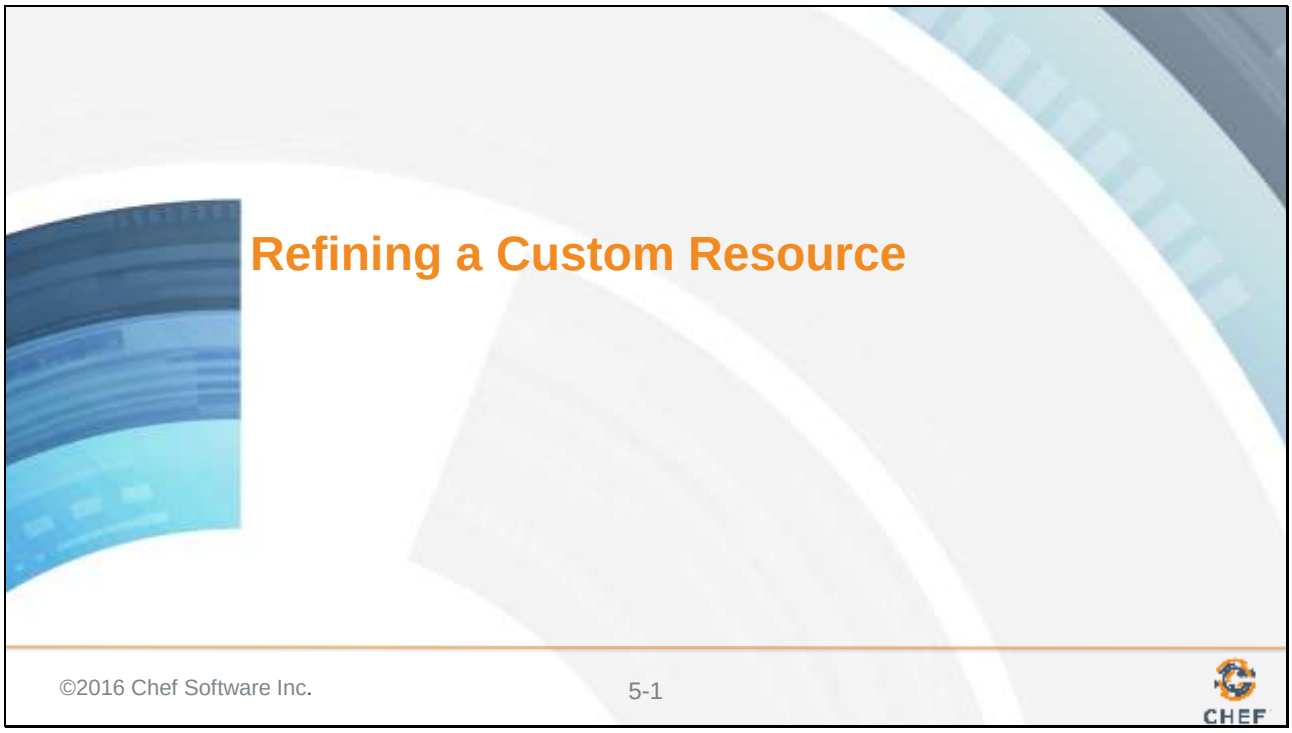
Let's finish this module with a discussion. Answer these questions. Remember that the answer "I don't know! That's why I'm here!" is a great answer.

Slide 64

What questions can we answer for you?

Slide 65

5: Refining a Custom Resource



Slide 2

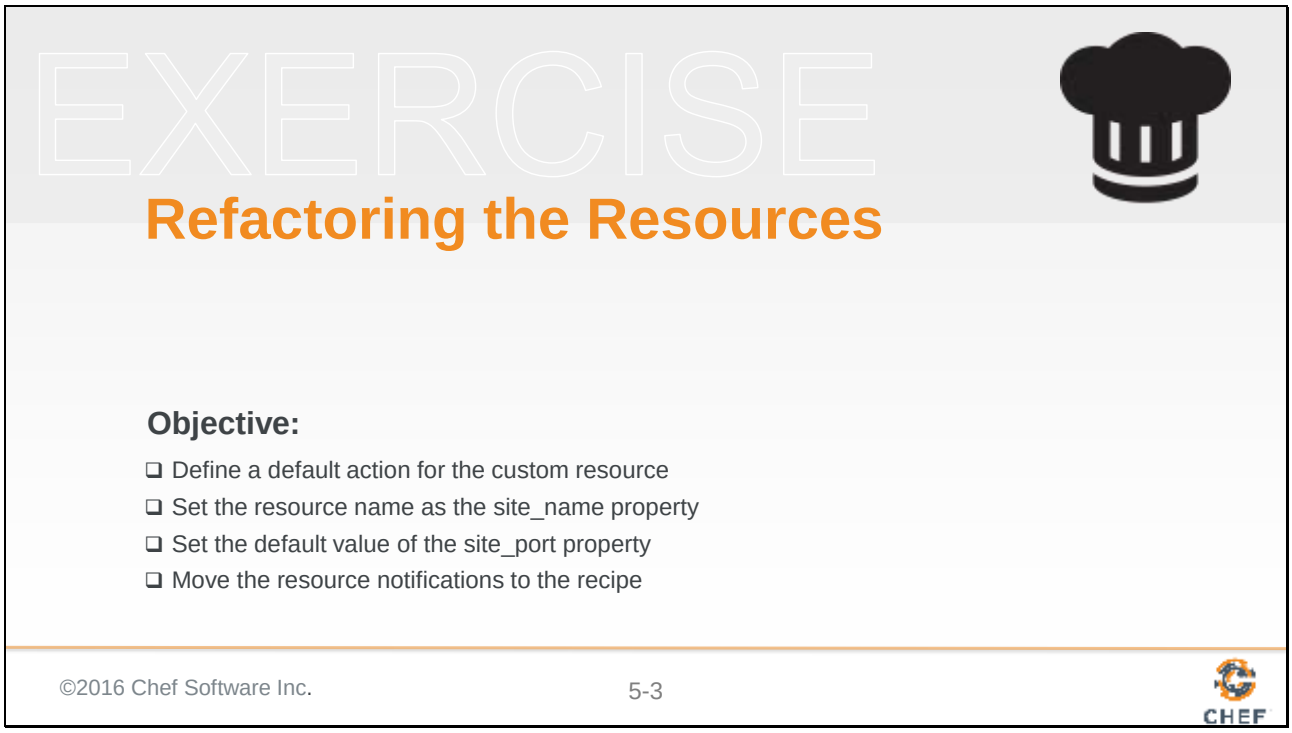
Objectives

After completing this module, you should be able to:

- Set a custom resource's name to a property
- Set a default value for a custom resource property
- Define notifications correctly when creating custom resources

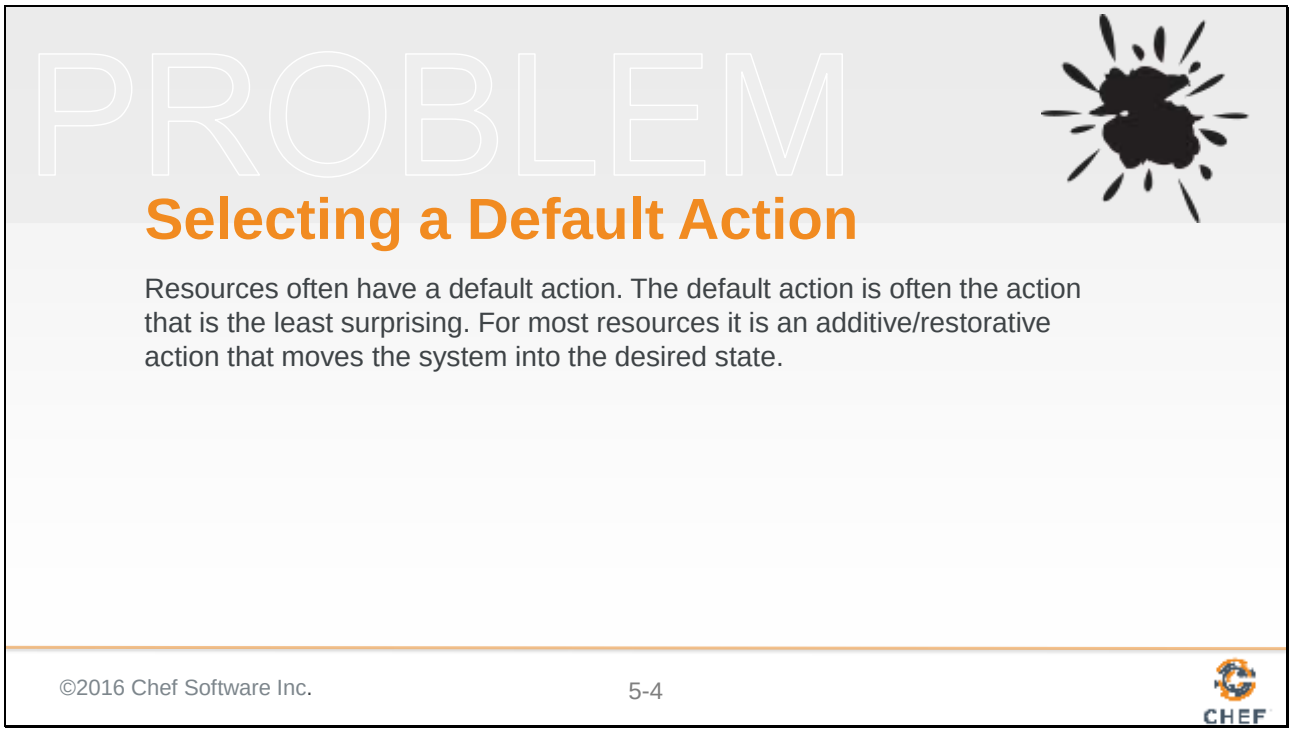
After completing this module you should be able to set a custom resource's name to a property, set a default value if the property is not provided, and define notifications correctly within the custom resource.

Slide 3



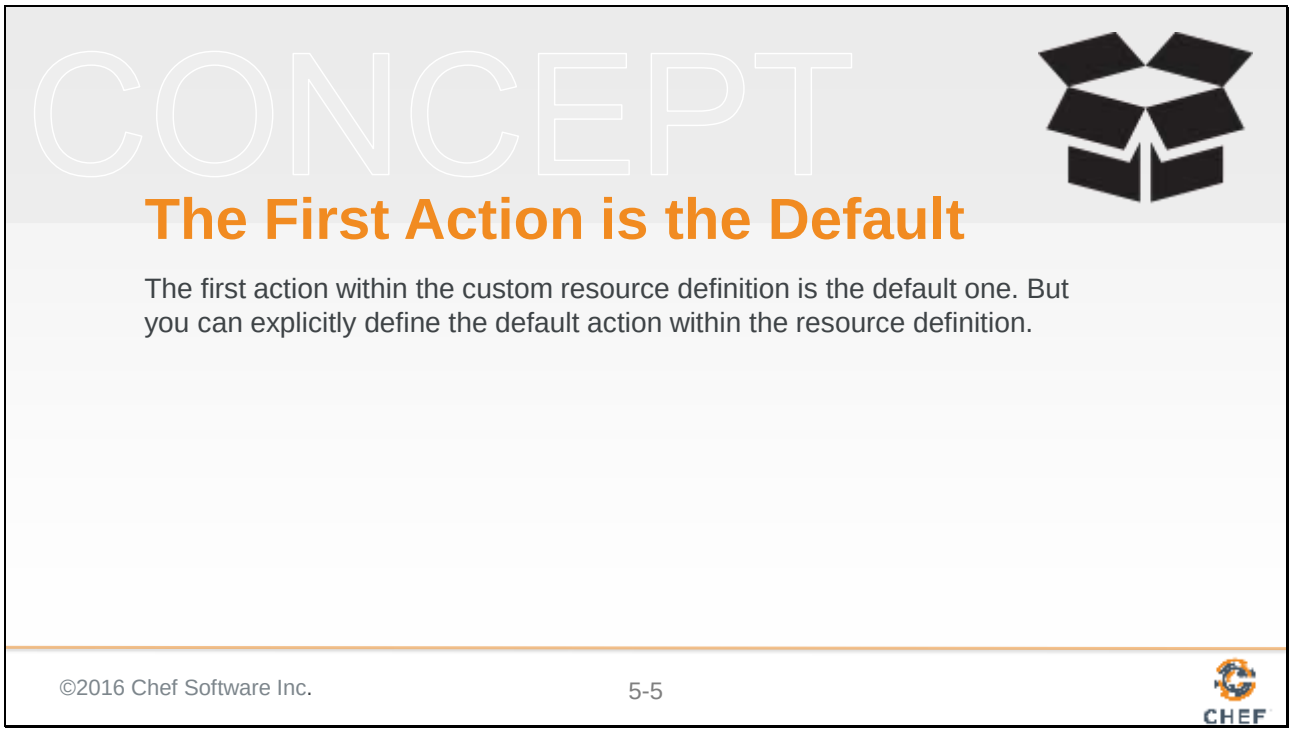
Every resource that you have used within Chef has had a default action.

Slide 4



Our custom resource should be no different. Having a default action that performs a non-surprising operation is important. Of the two actions that we have defined the create action seems like the current correct default action.

Slide 5



If you defined the create action as the first action within a resource definition file it automatically is the default action. The first action is the default action and that probably makes sense to those reading the custom resource definition.

You may also explicitly declare the default action.

Slide 6

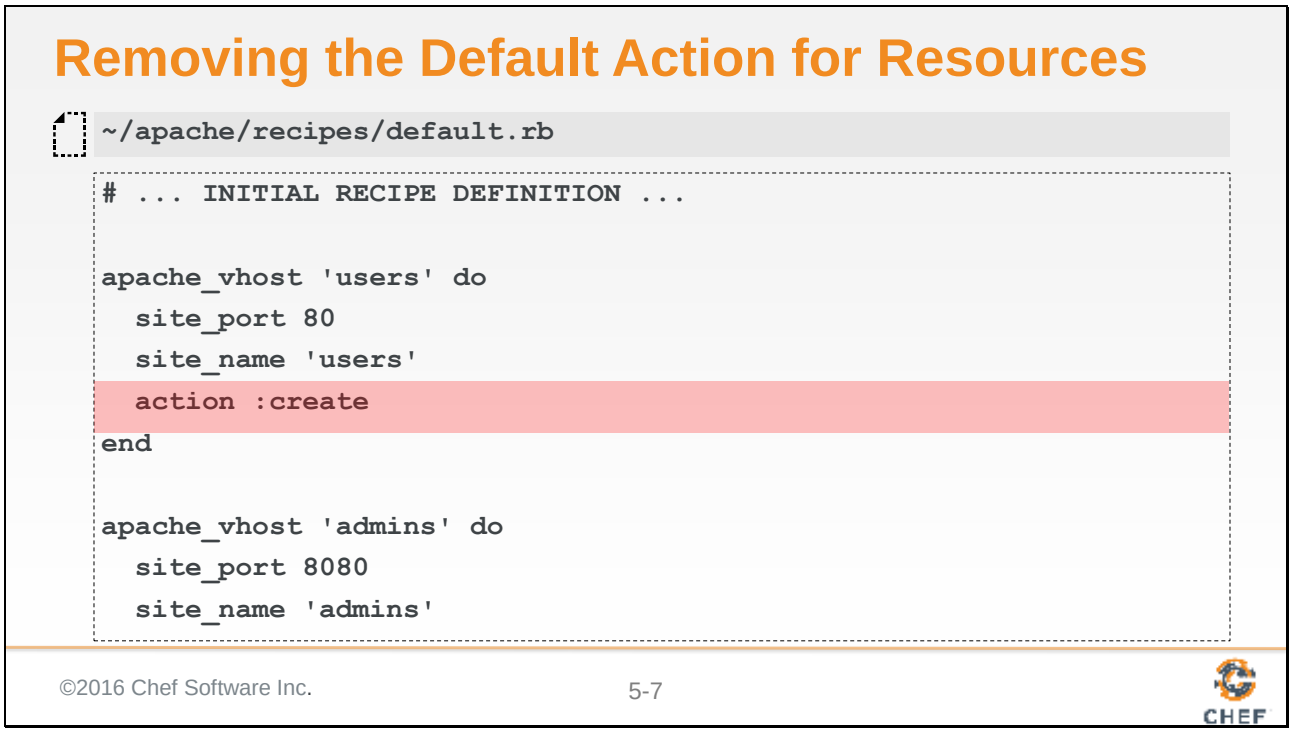
Defining a Default Action for the Resource

~/apache/resources/vhost.rb

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

default_action :create

action :create do
  directory "/srv/apache/#{site_name}/html" do
    recursive true
    mode '0755'
  end
# ... REMAINING RESOURCE DEFINITION ...
```


If the first action is the create action then you do not need to explicitly define the default action. However, you may decide that it makes it clearer for you or those you are collaborating with to specify this within the resource definition.

Slide 7



With the default action defined all uses of the custom resource which explicitly defined it may have those lines removed.

Slide 8

Removing the Default Action for Resources

~/apache/recipes/default.rb

```
# ... RECIPE DEFINITION ...

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

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


With the default action defined all uses of the custom resource which explicitly defined it may have those lines removed.

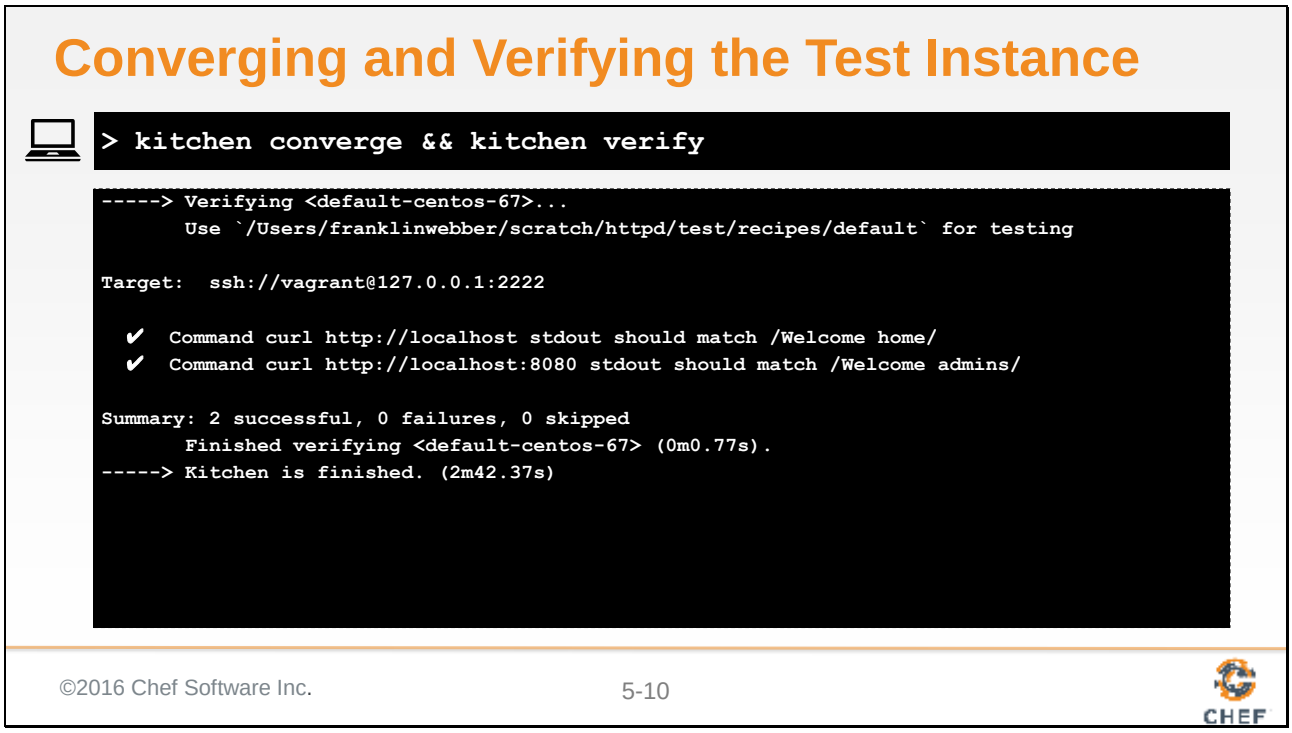
Slide 9

Executing the Unit Test Suite

```
> chef exec rspec
```

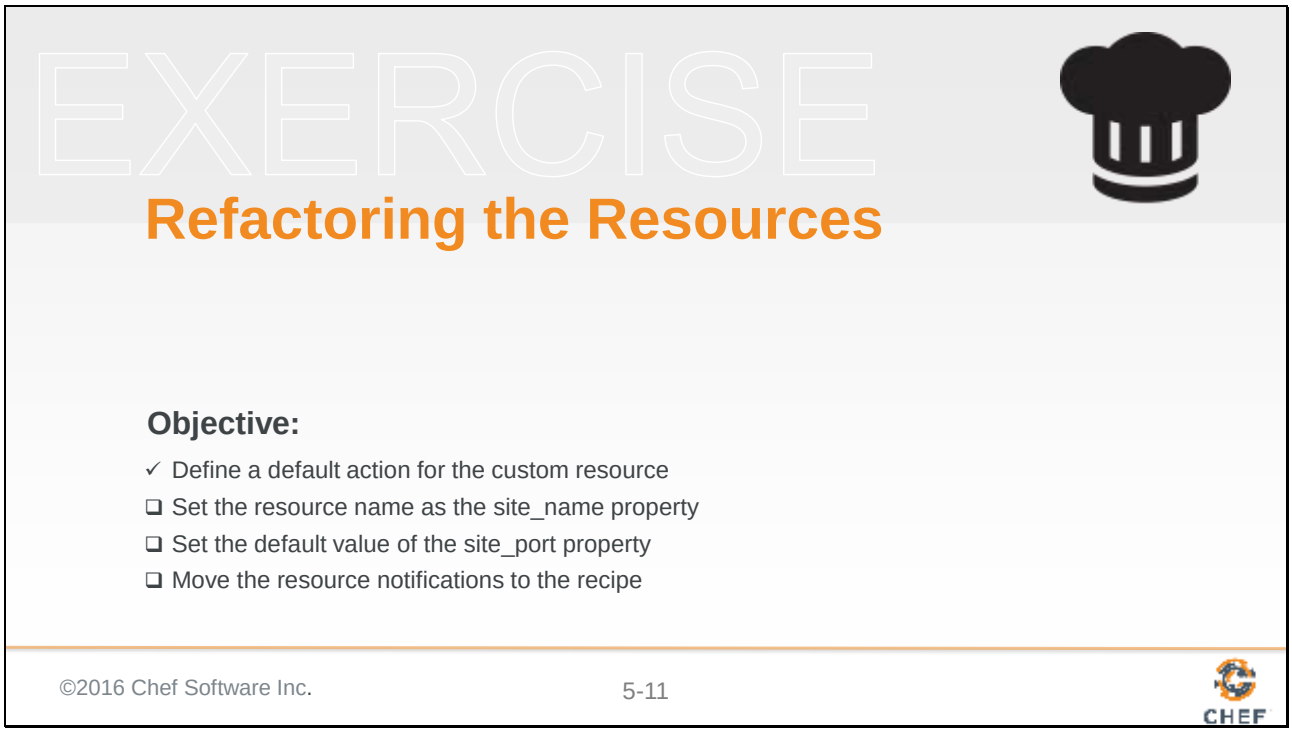
```
..........

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


All the unit tests should pass.



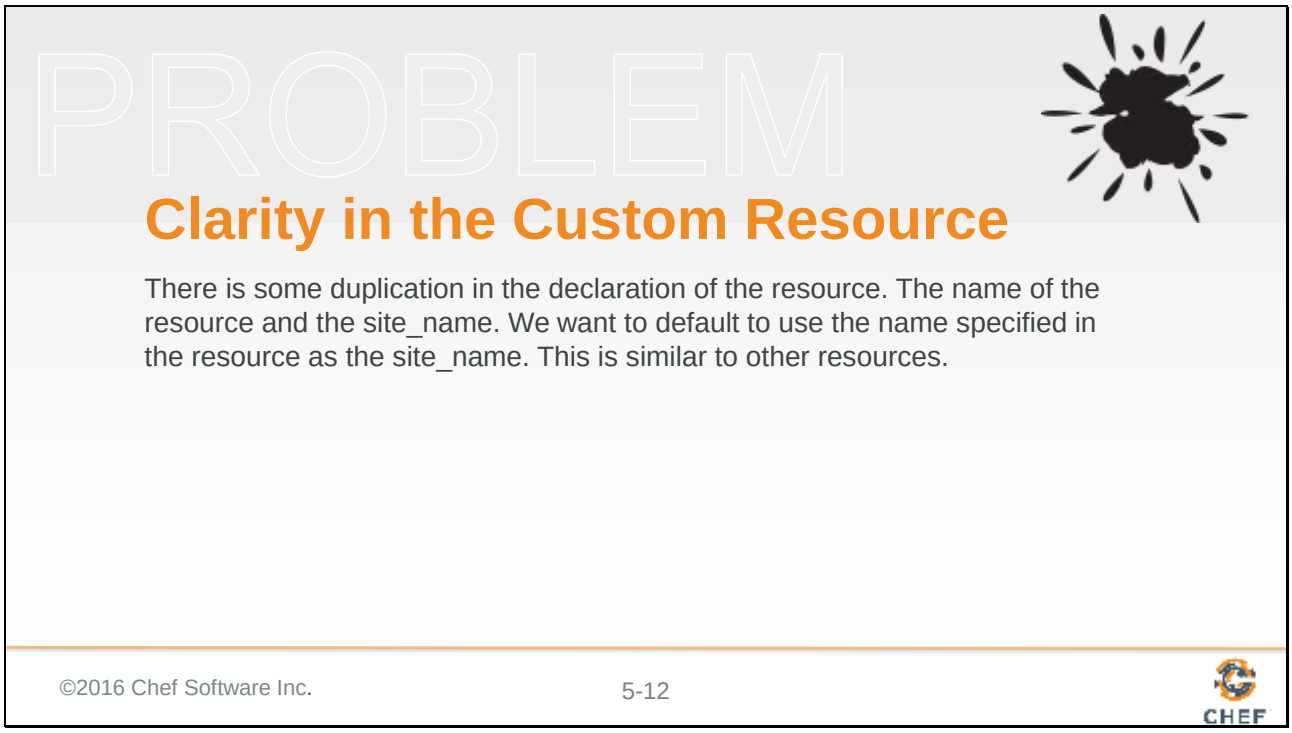
All the integration test should pass.

Slide 11



Now we will look at tying the name provided to the custom resource to the site_name property.

Slide 12



We want to ensure our custom resource is clear and concise. At the moment when you define the resource within the recipe you specify a value as the name of the custom resource and then a property that matches that same name. This seems like a redundancy that we want to remove.

Slide 13

Tying the Resource Name to the Property

~/apache/resources/vhost.rb

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

default_action :create

action :create do
  directory "/srv/apache/#{site_name}/html" do
    recursive true
    mode '0755'
  end
# ... REMAINING RESOURCE DEFINITION ...
```


One property may have the 'name_attribute' option set to true. This property will be automatically populated from the name of the resource. Using the name of the resource will allow us to remove the need to specify that property which is repeating a value within the use of the custom resource.

Removing the site_name Property

~/apache/recipes/default.rb

```
package 'apache'

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

apache_vhost 'admins' do
  site_port 80
  site_name 'users'
end
```

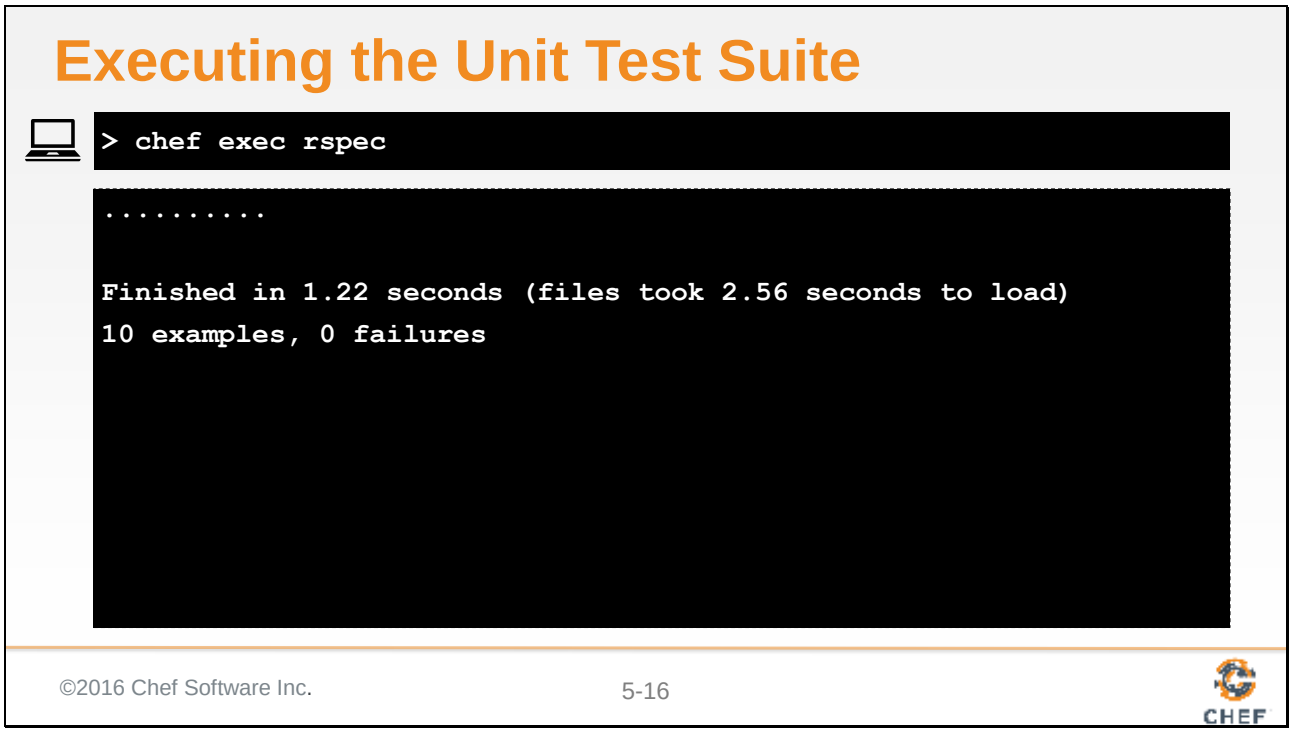

Now that the site_name property is set as the name attribute we can remove the site name property from each use of the custom resource.

Removing the site_name Property

~/apache/recipes/default.rb

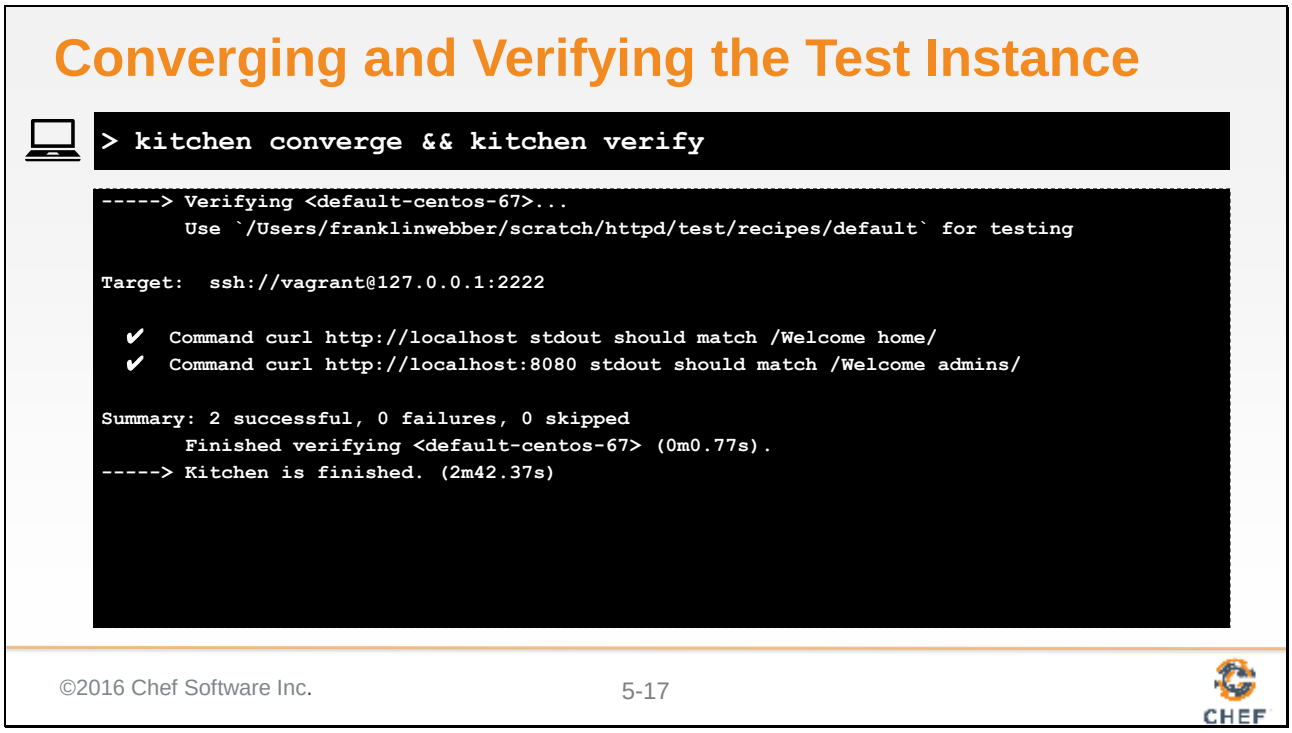
```
# ... INITIAL RECIPE DEFINITION ...

apache_vhost 'admins' do
  site_port 8080
  site_name 'admins'
end

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


Now that the site_name property is set as the name attribute we can remove the site name property from each use of the custom resource.



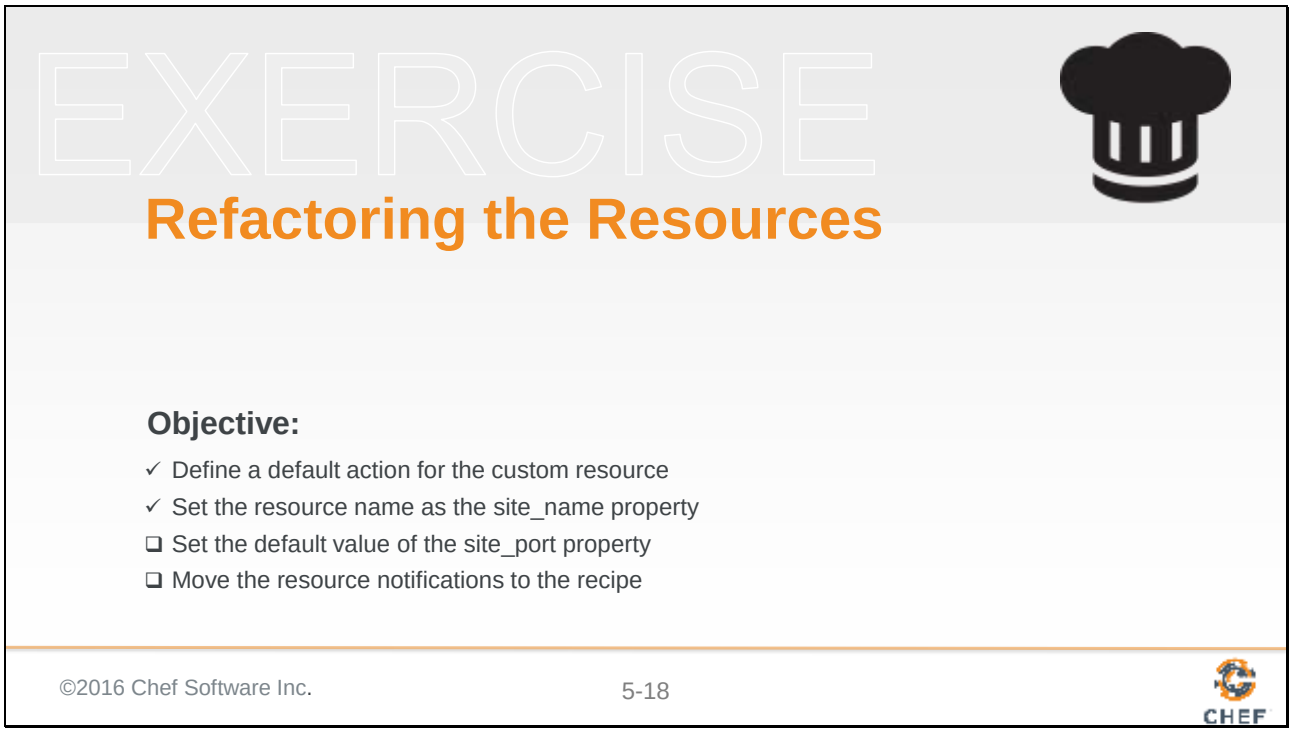
All the unit tests should pass.

Slide 17



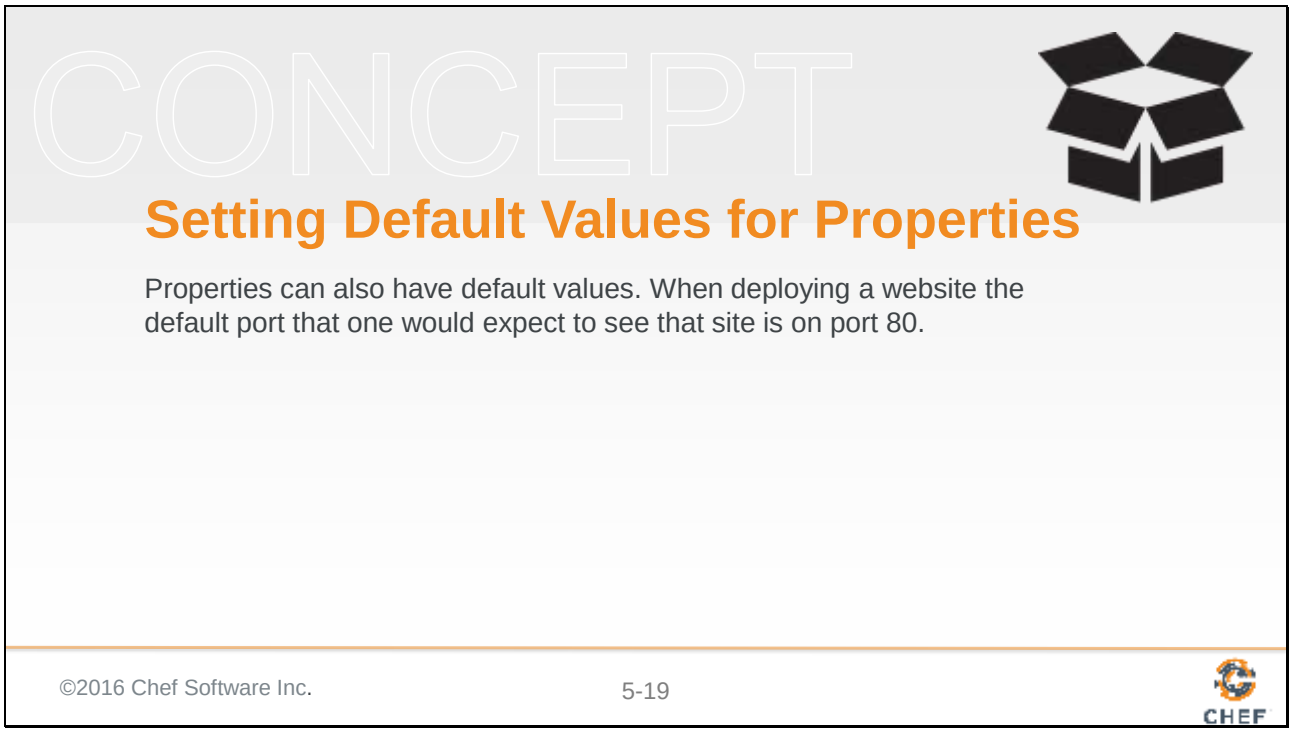
All the integration test should pass.

Slide 18



Resource properties can also have default values setup for them. Lets explore setting a default value for the site_port property.

Slide 19



Setting a default value for a property allows you to define the resource and if you omit setting the property then the default value is used. We can use this behavior for our site_port, choosing to say that if you do not specify a port we want the port to be 80.

Slide 20

Setting a Default Value for the Property

~/apache/resources/vhost.rb

```
property :site_name, String, name_attribute: true
property :site_port, Fixnum, default: 80

default_action :create

action :create do
  directory "/srv/apache/#{site_name}/html" do
    recursive true
    mode '0755'
  end
# ... REMAINING RESOURCE DEFINITION ...
```


All properties may have a default value. To add one requires that you simply add the option 'default' with its corresponding default value. Here we are setting the 'site_port' value to 80.

Slide 21

Removing the site_port Property

~/apache/recipes/default.rb

```
package 'apache'

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

apache_vhost 'users' do
  site_port 80
end
```


This allows us to remove the value from a single resource.

Slide 22

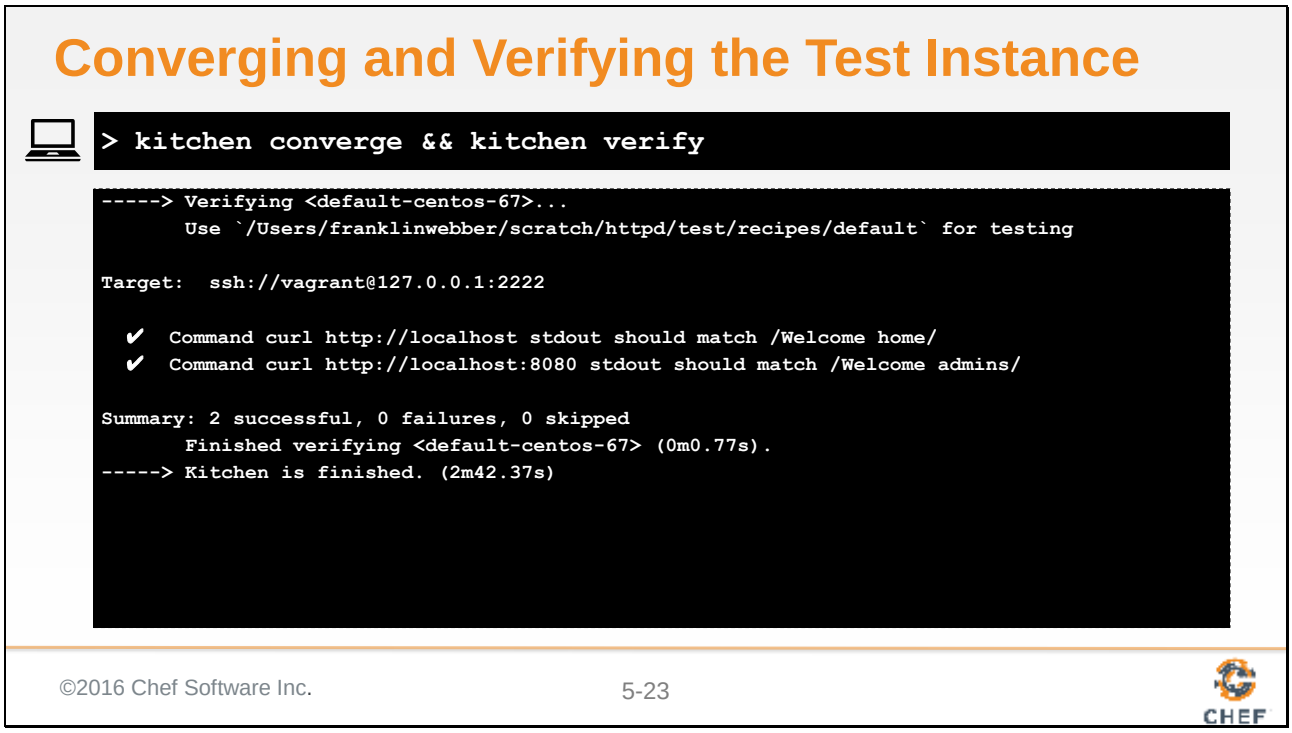
Executing the Unit Test Suite

```
> chef exec rspec
```

```
..........

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

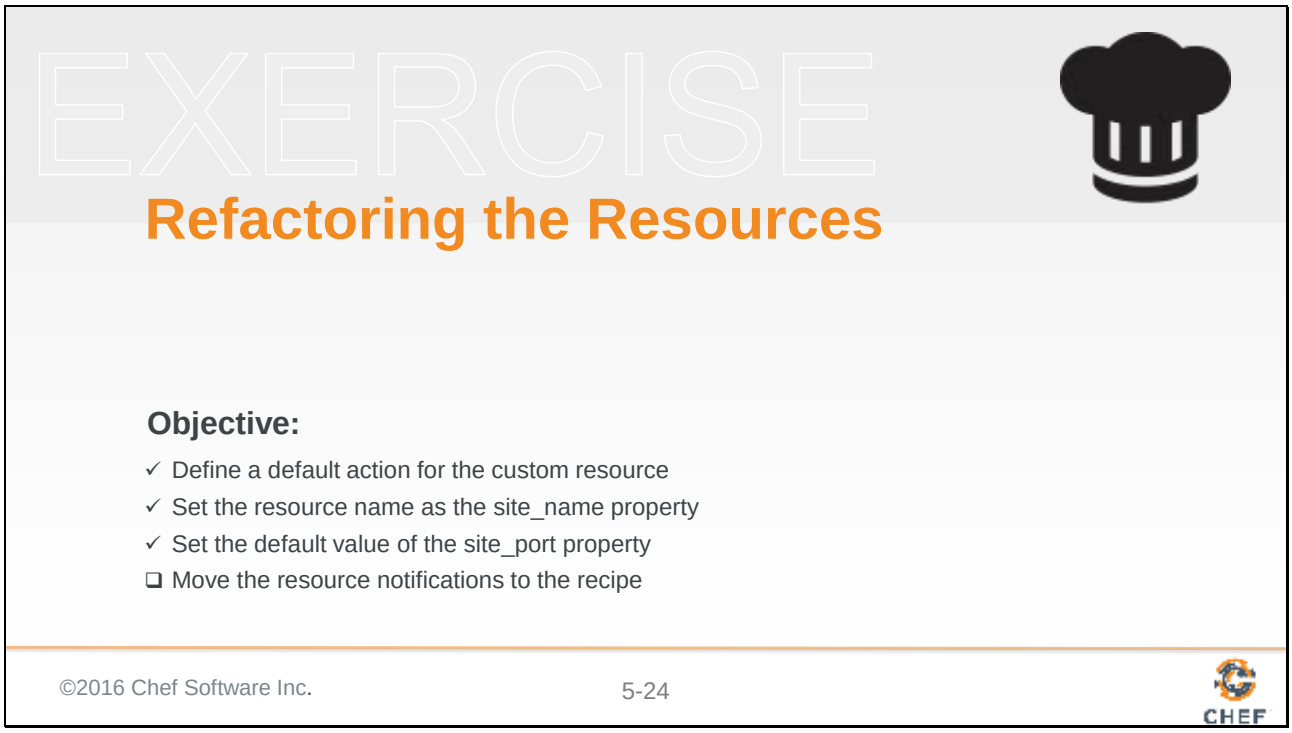

All the unit tests should pass.

Slide 23



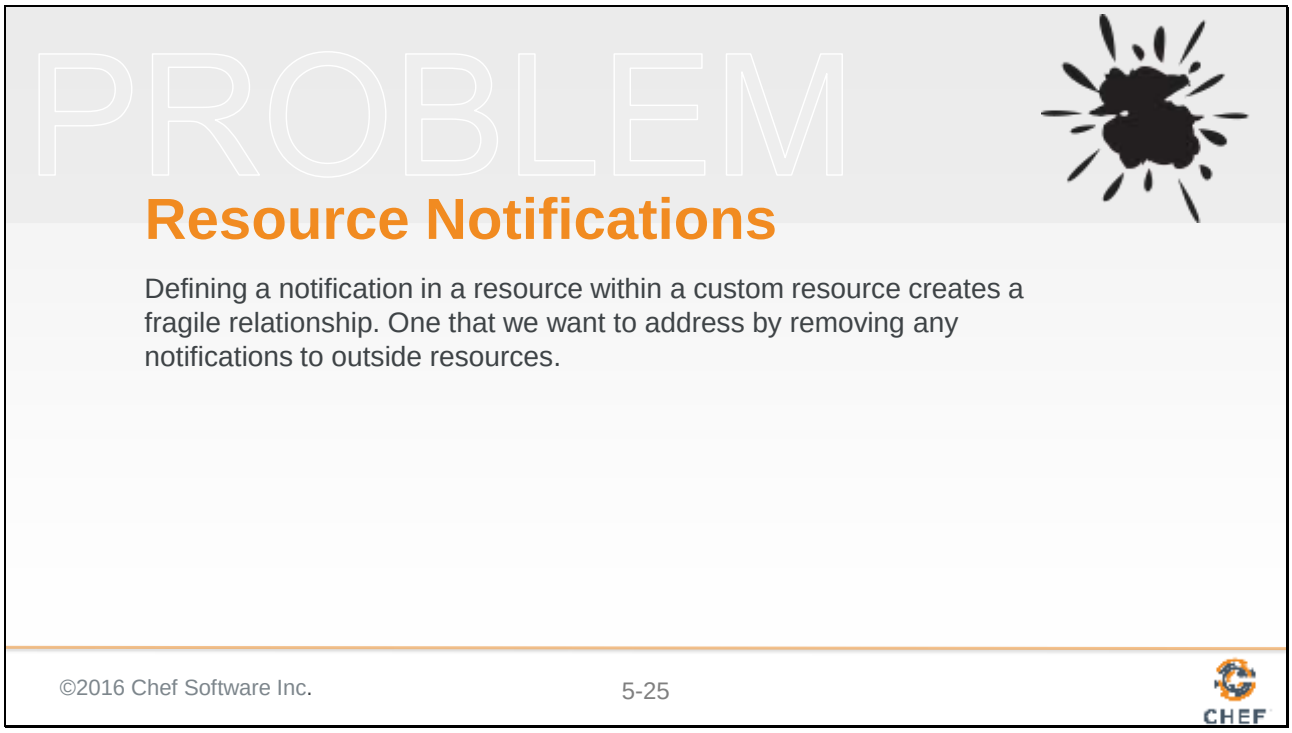
All the integration test should pass.

Slide 24



Finally we want to properly address how the custom resource handles notifications.

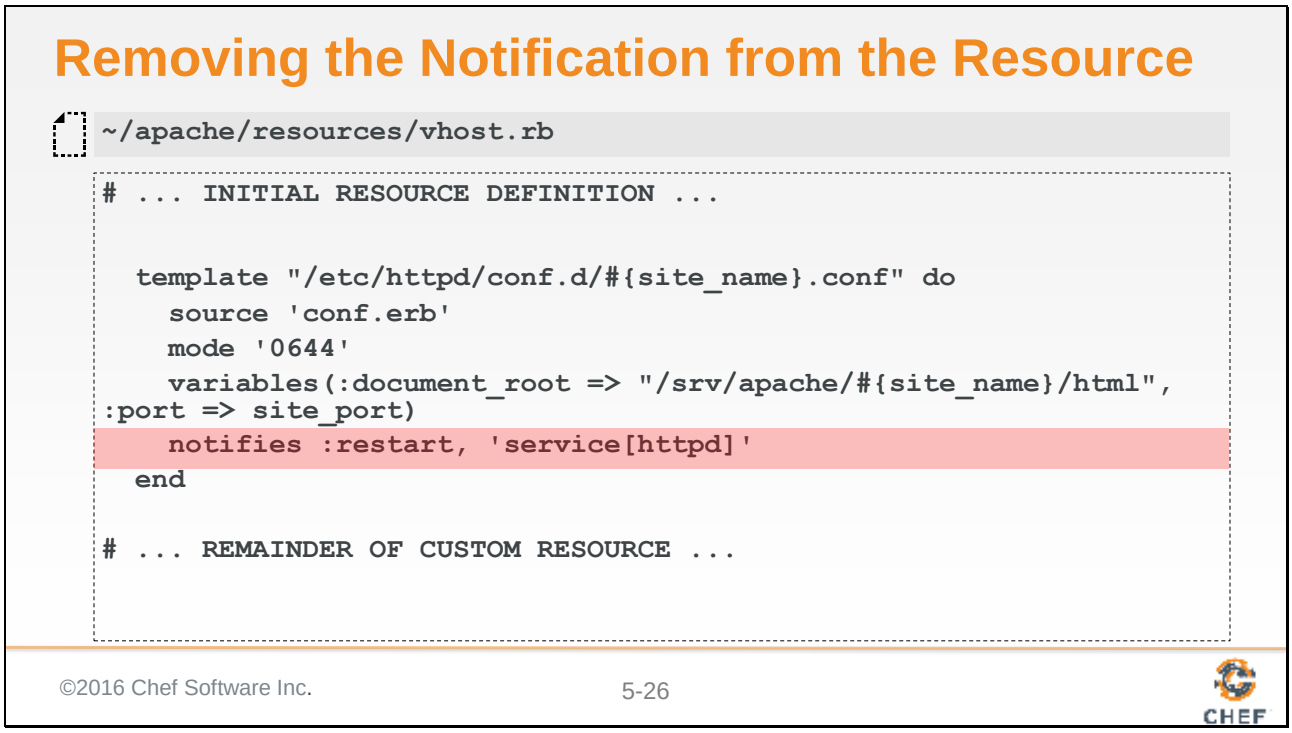
Slide 25



When defining notifications within a resource action if you reference a resource outside of the action implementation there is a chance that your code may break if that resource's name were to change or simply not be implemented at all.

If any resource were to take action within the custom resource then the custom resource considers itself as taking action. We often say that any changed resource events are sent the parent custom resource and from that custom resource you can define your notifications.

Slide 26



First we remove the dependency on a resource not present within the action implementation of the custom resource.

Slide 27

Adding the Notification to the Resources

~/apache/recipes/default.rb

```
package 'apache'

apache_vhost 'welcome' do
  notifies :restart, 'service[httpd]'
  action :remove
end

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


We then add the notifications to the custom resource implementations within the default recipe.

Slide 28

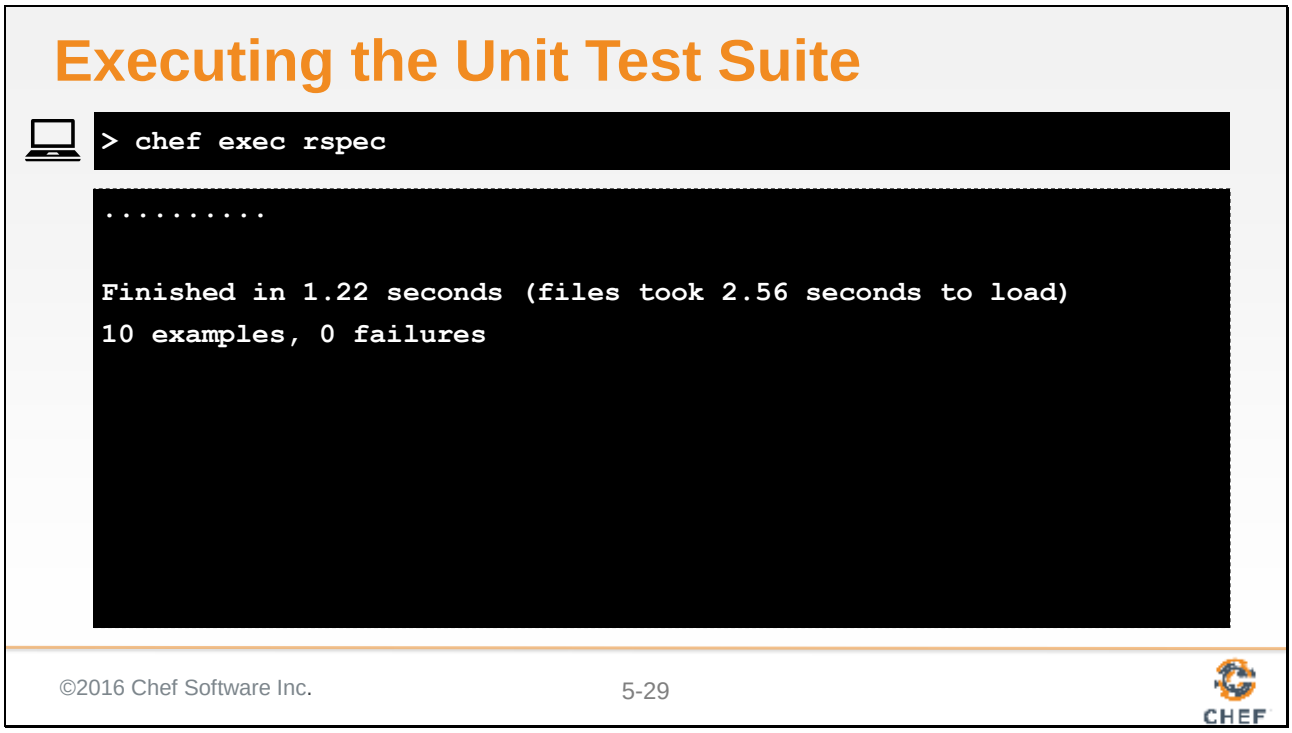
Adding the Notification to the Resources

~/apache/recipes/default.rb

```
# ... INITIAL RECIPE DEFINITION ...

apache_vhost 'admins' do
  notifies :restart, 'service[httpd]'
  site_port 8080
end

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

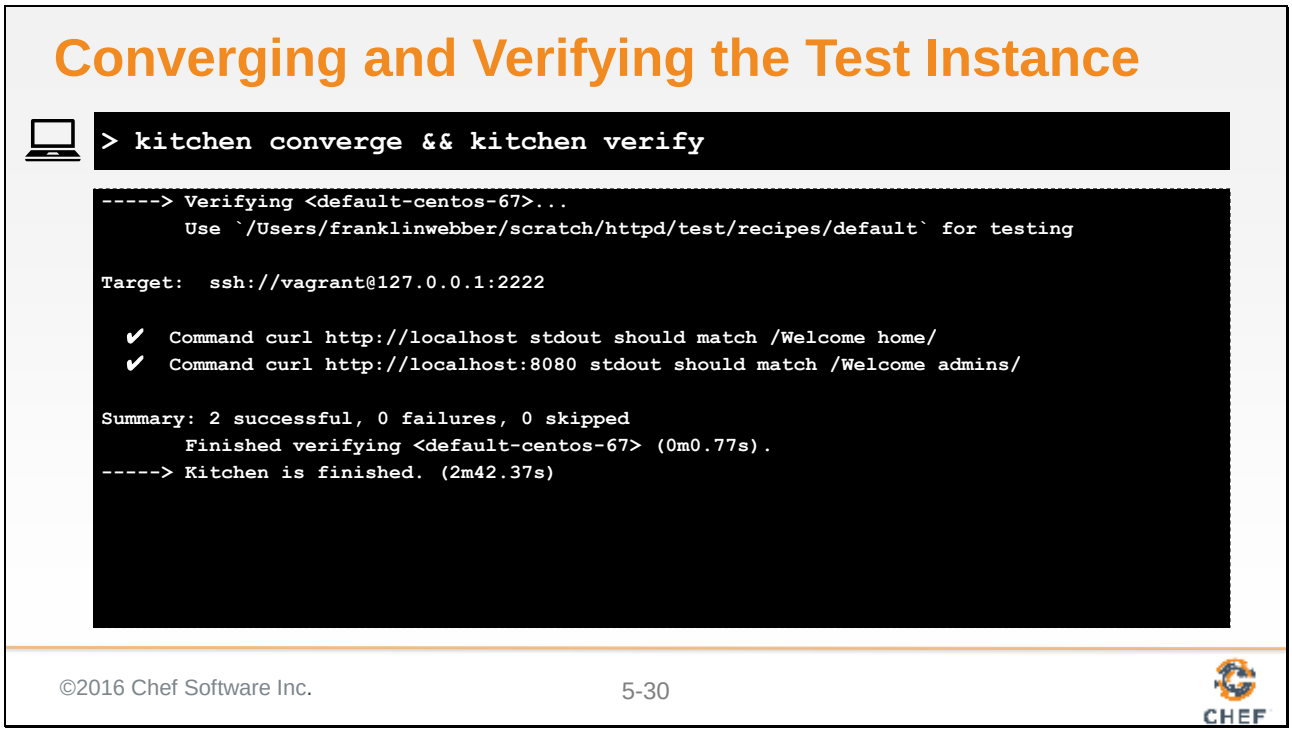

We then add the notifications to the custom resource implementations within the default recipe.

Slide 29



All the unit tests should pass.

Slide 30



All the integration test should pass.

Slide 31

EXERCISE

Refactoring the Resources

Objective:

- ✓ Define a default action for the custom resource
- ✓ Set the resource name as the site_name property
- ✓ Set the default value of the site_port property
- ✓ Move the resource notifications to the recipe

CHEF

We now have refactored the custom resource to have it behave more like other resources we are familiar within Chef.

Slide 32

What questions can we answer for you?

Slide 33

6: Ohai

Before you set out to start managing your nodes it is important to understand the current state of your nodes. As Chef, a platform agnostic tool, is written in Ruby, a platform agnostic language, it is useful to understand what is or is not installed on the system. This information is helpful in helping a resource select the correct provider or for that provider to determine which version of the tool or language is at it disposal.

Slide 2

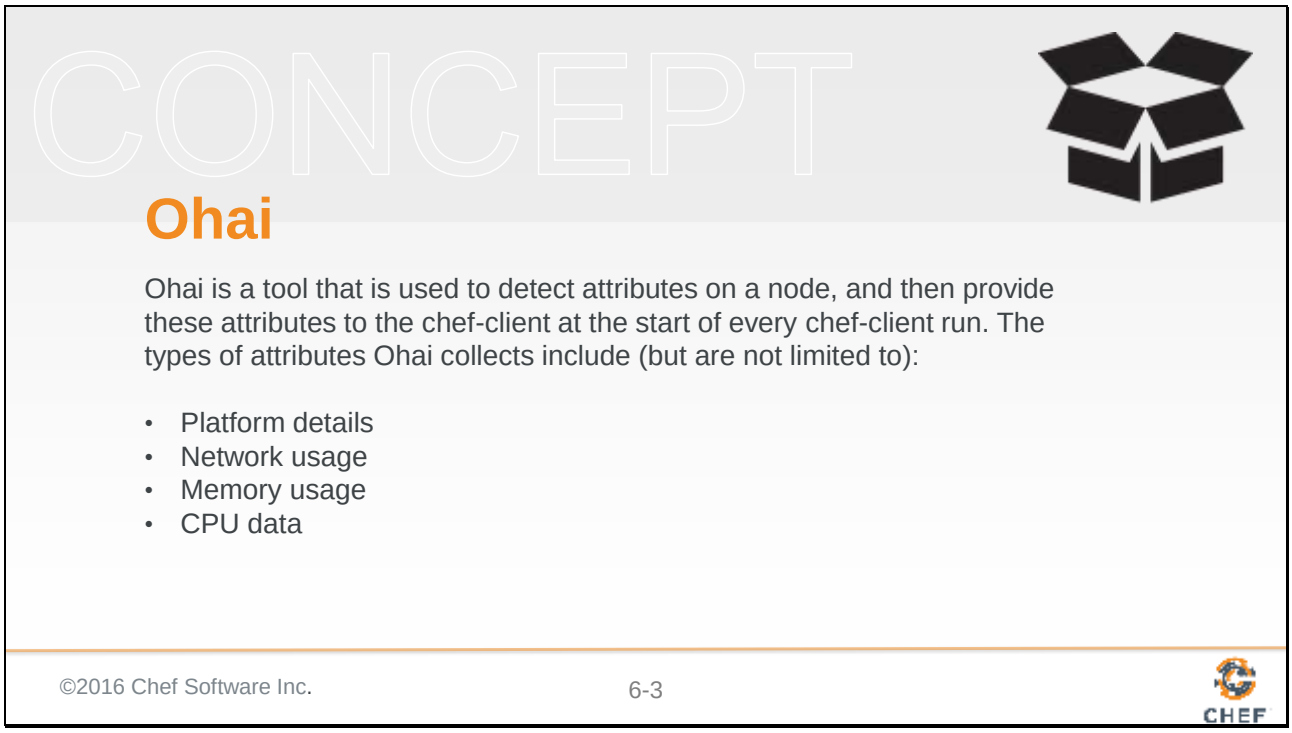
Objectives

After completing this module, you should be able to:

- Execute the Ohai command-line tool to return an attribute
- Describe when Ohai is loaded in the chef-client run
- Describe when new attributes for the node are stored
- Describe precedence of attributes collected by Ohai

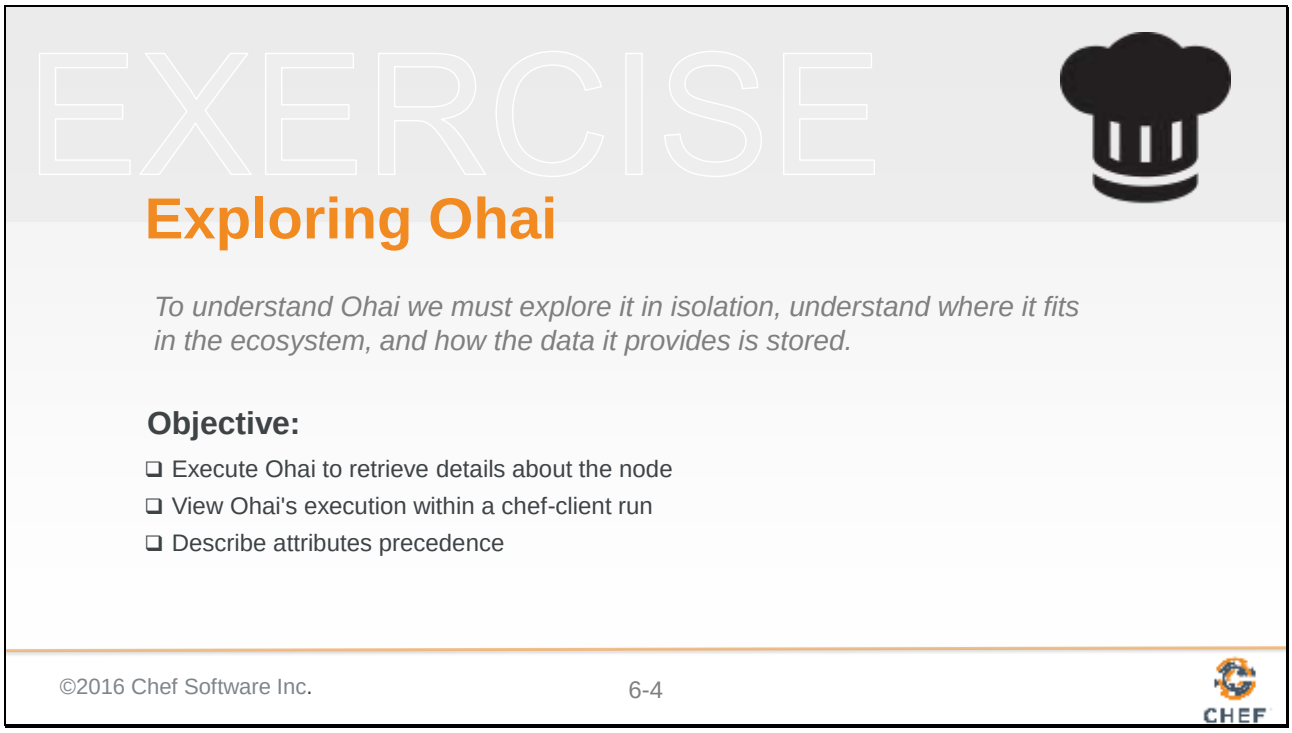
After completing this module you will be able to execute the Ohai command-line tool to return an attribute, describe when Ohai is loaded in the chef-client run, when new attributes for the node are stored in that chef-client run, and be able to describe the attribute precedence for attributes collect by Ohai.

Slide 3



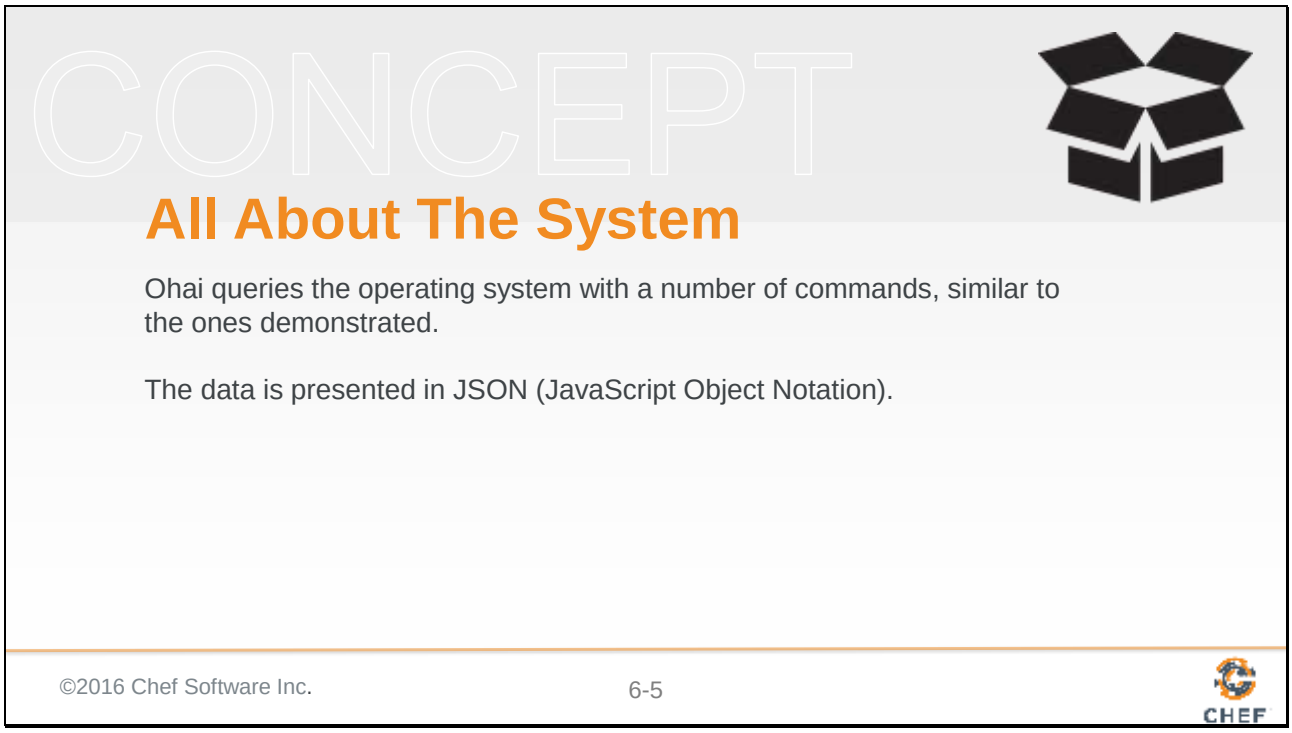
Ohai is a tool that is used to detect attributes on a node, and then provide these attributes to the chef-client at the start of every chef-client run. Ohai is required by the chef-client and must be present on a node. (Ohai is installed on a node as part of the chef-client install process.)

Slide 4



As a group we will explore using Ohai from the command-line then view how it is executed within a chef-client run and then talk about the attributes that it collects. We'll start with ohai the command-line tool.

Slide 5



Ohai, the command-line application, will output all the system details represented in JavaScript Object Notation (JSON).

Slide 6

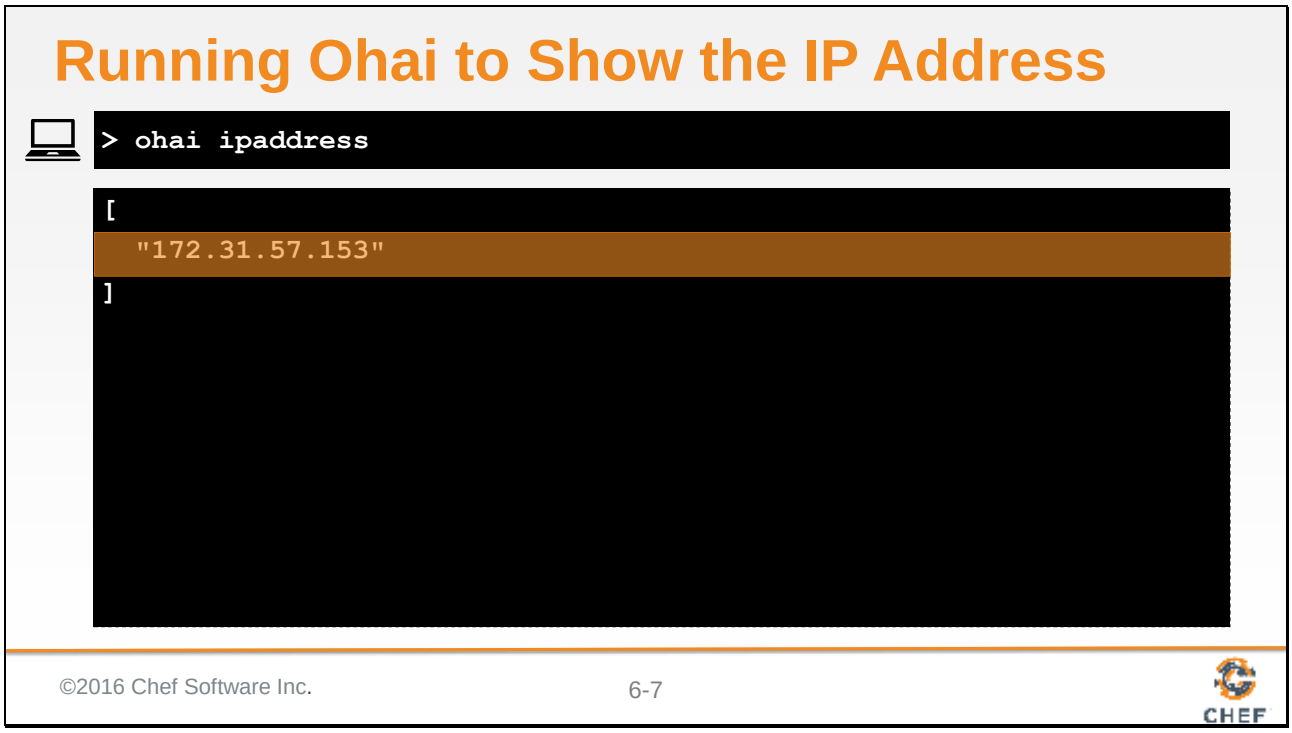
Running Ohai to Show All Attributes

```
> ohai
```


```
{
  "kernel": {
    "name": "Linux",
    "release": "2.6.32-431.1.2.0.1.el6.x86_64",
    "version": "#1 SMP Fri Dec 13 13:06:13 UTC 2013",
    "machine": "x86_64",
    "os": "GNU/Linux",
    "modules": {
      "veth": {
        "size": "5040",
        "refcount": "0"
      },
      "ipt_addrtype": {
```

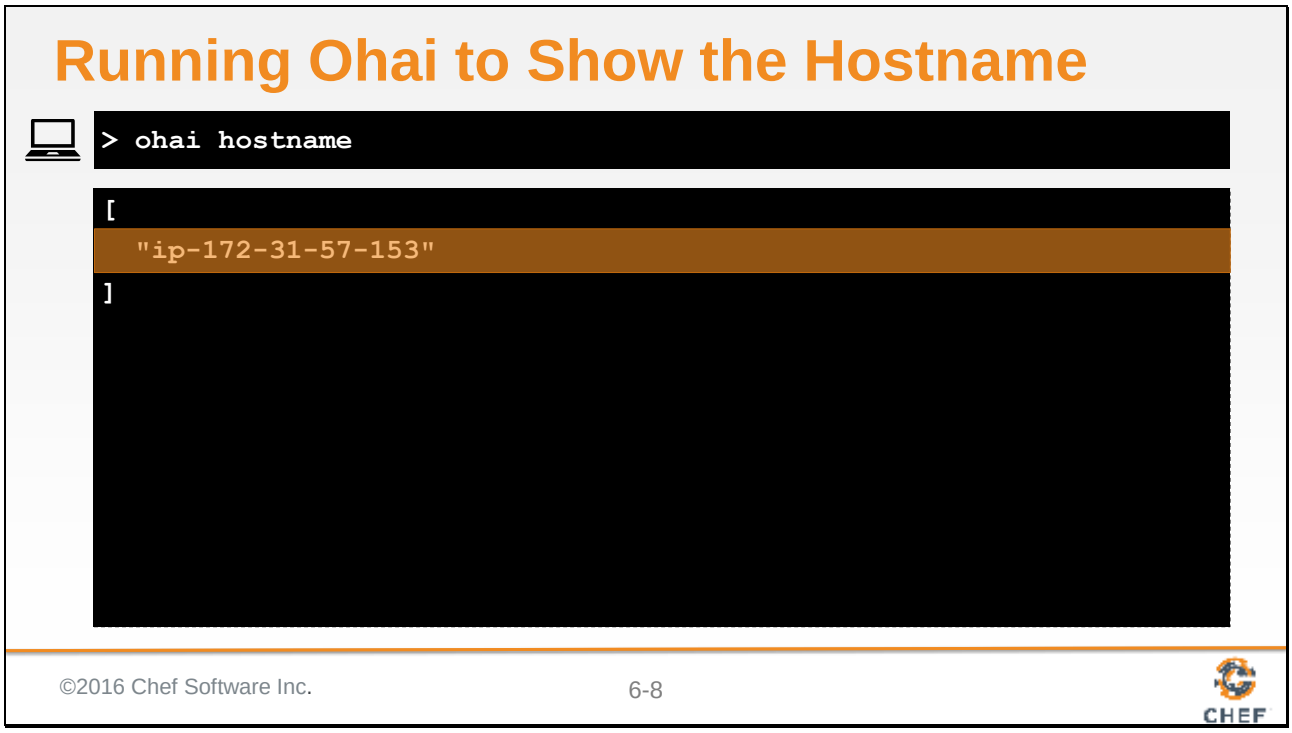

Ohai is also a command-line application that is part of the ChefDK. When you run it you will see the entire JSON representation of the system.

Slide 7



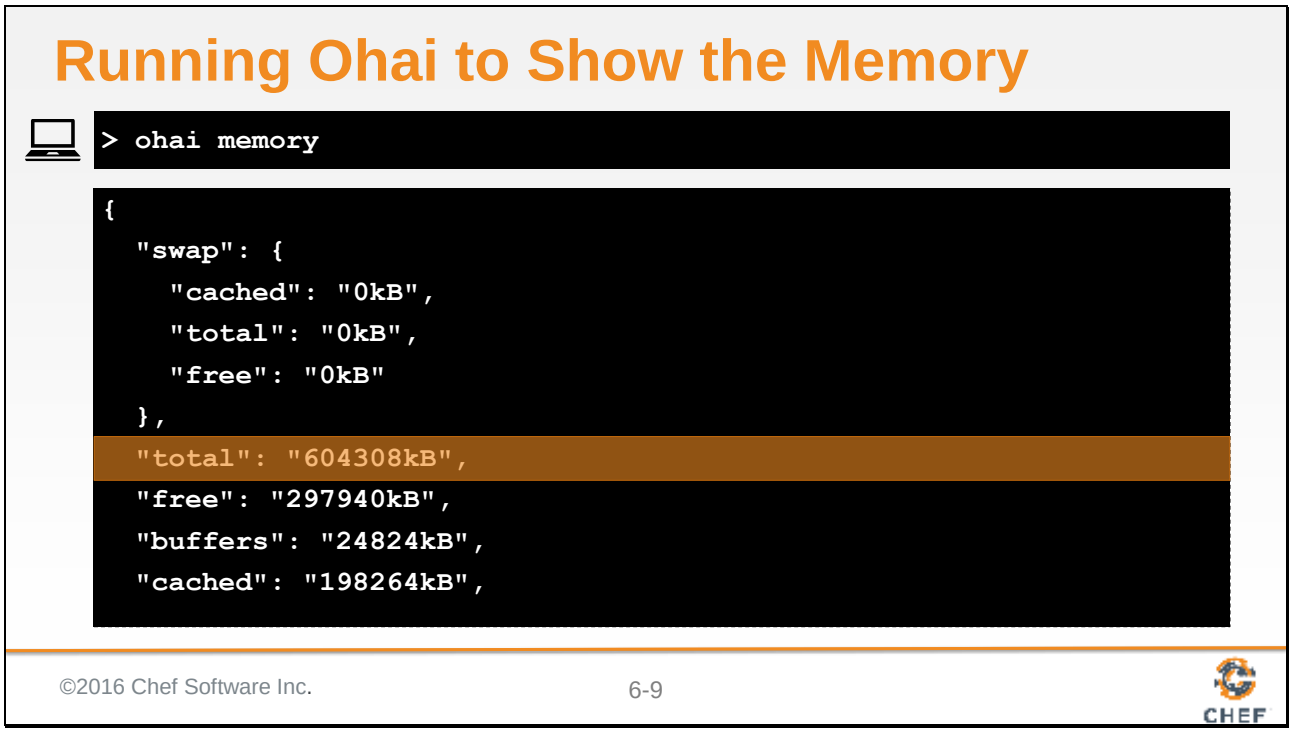
You can also run ohai with a parameter. In this case when we want only the ipaddress from the entire body of information we can provide it as a parameter.

Slide 8



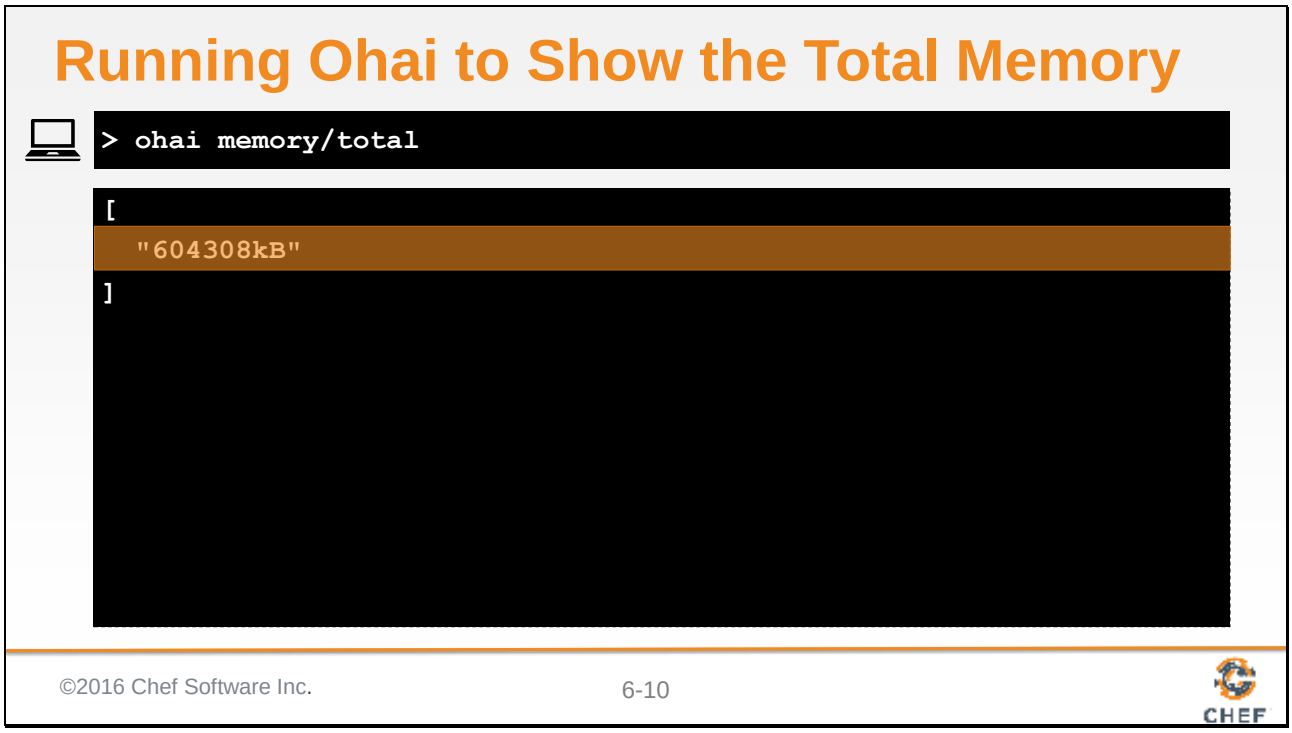
Similar, we can specify the hostname to return only the hostname of the system.

Slide 9

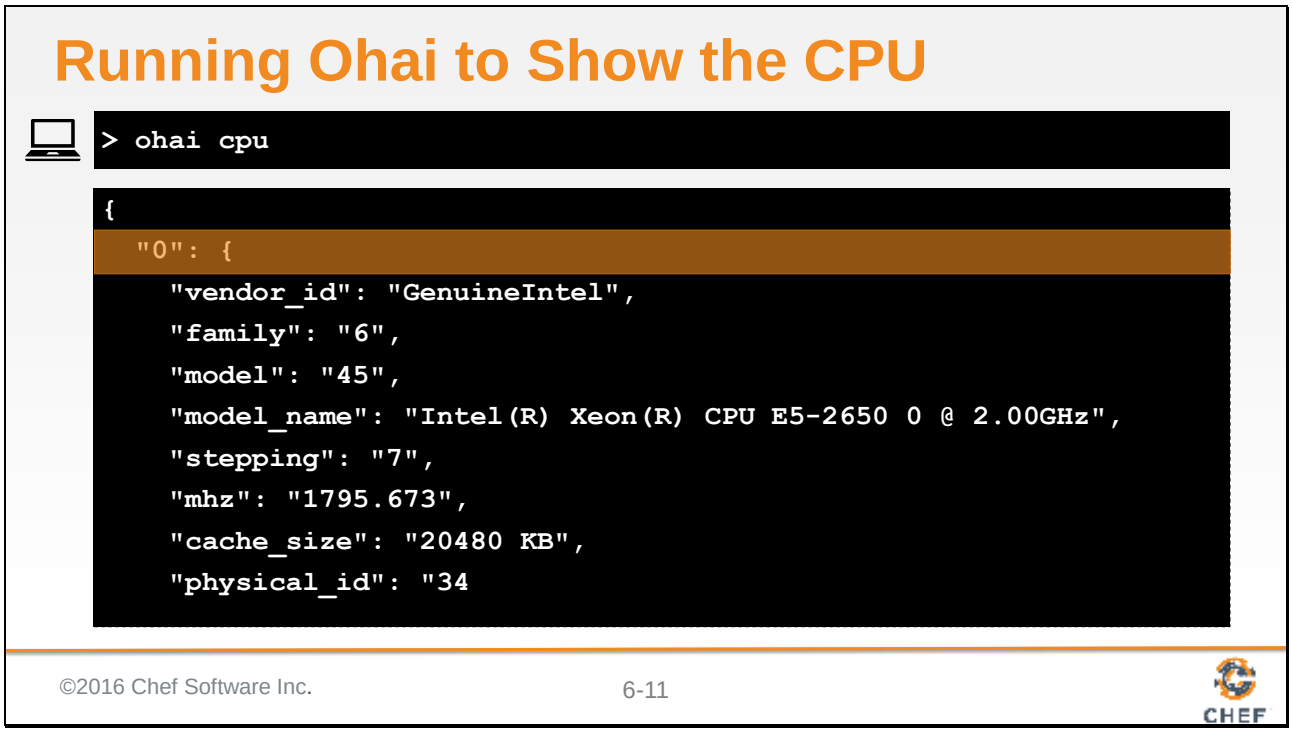


When we ask for the memory of the system we receive a hash that contains a number of keys and values.

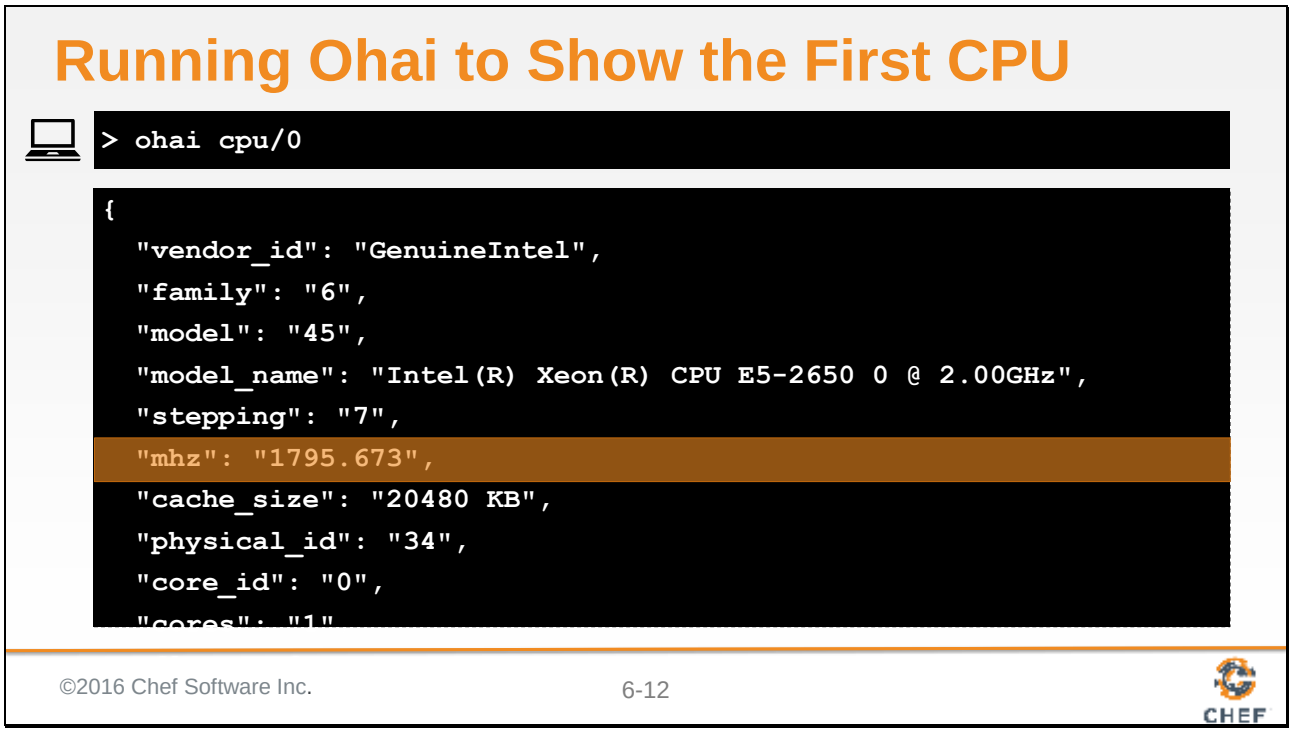
Slide 10



We can grab a single value, like the total memory, by specifying a slash between the top-level key and the next level key underneath it. This command will return the total memory of the system.

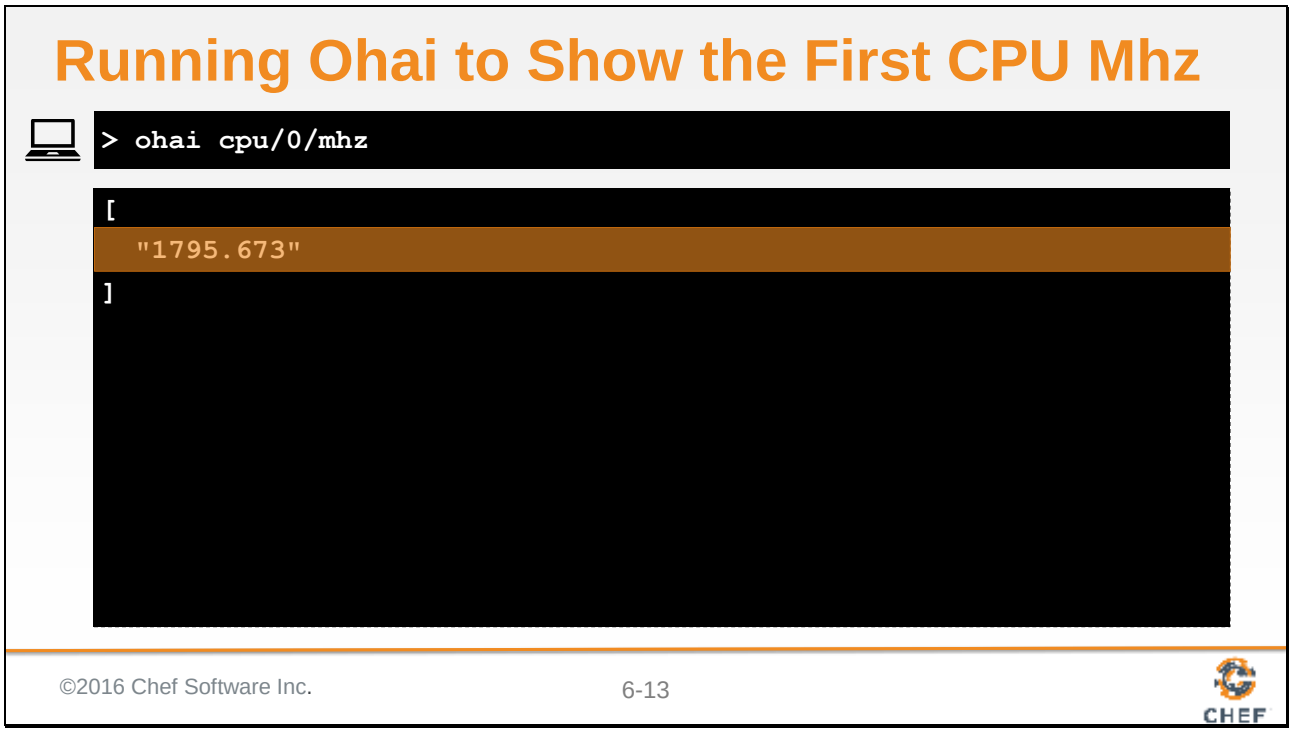


We can return all the details about the cpu. We see that there is one cpu, named '0', that contains more information.



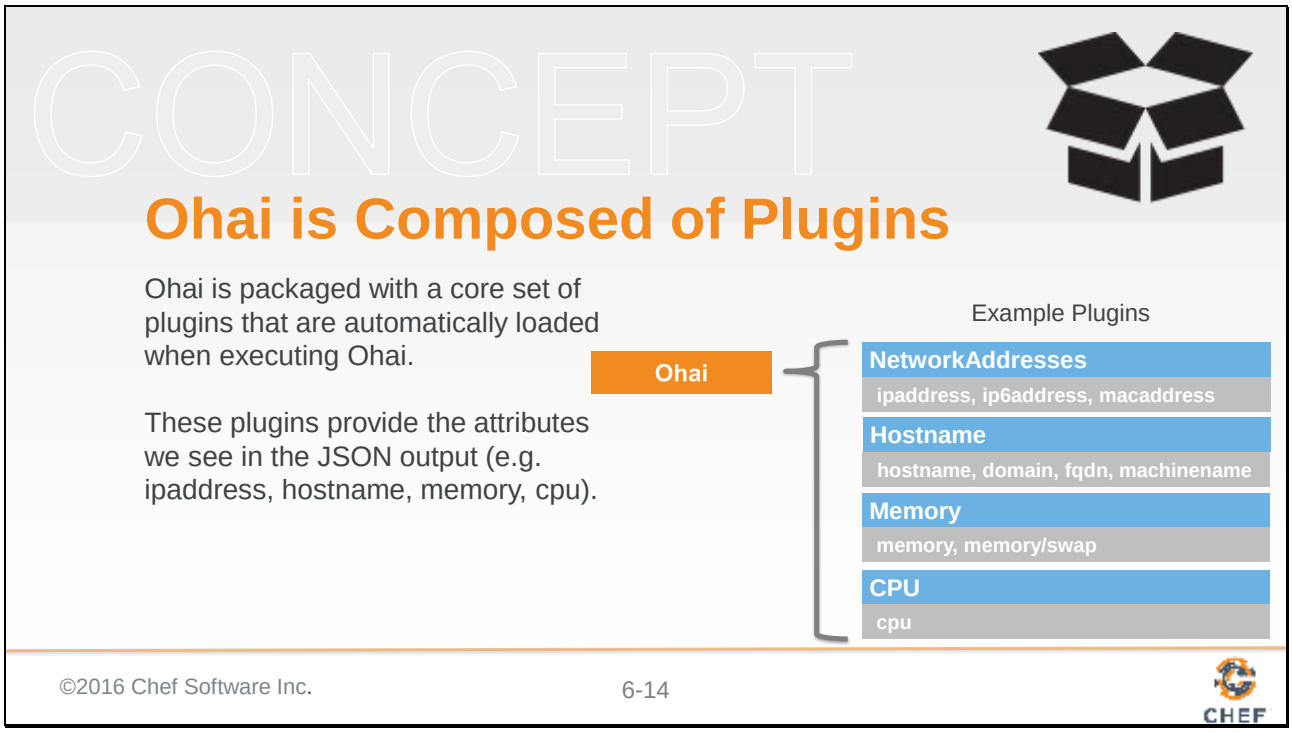
Here we are asking for all the details about the cpu named '0'.

Slide 13



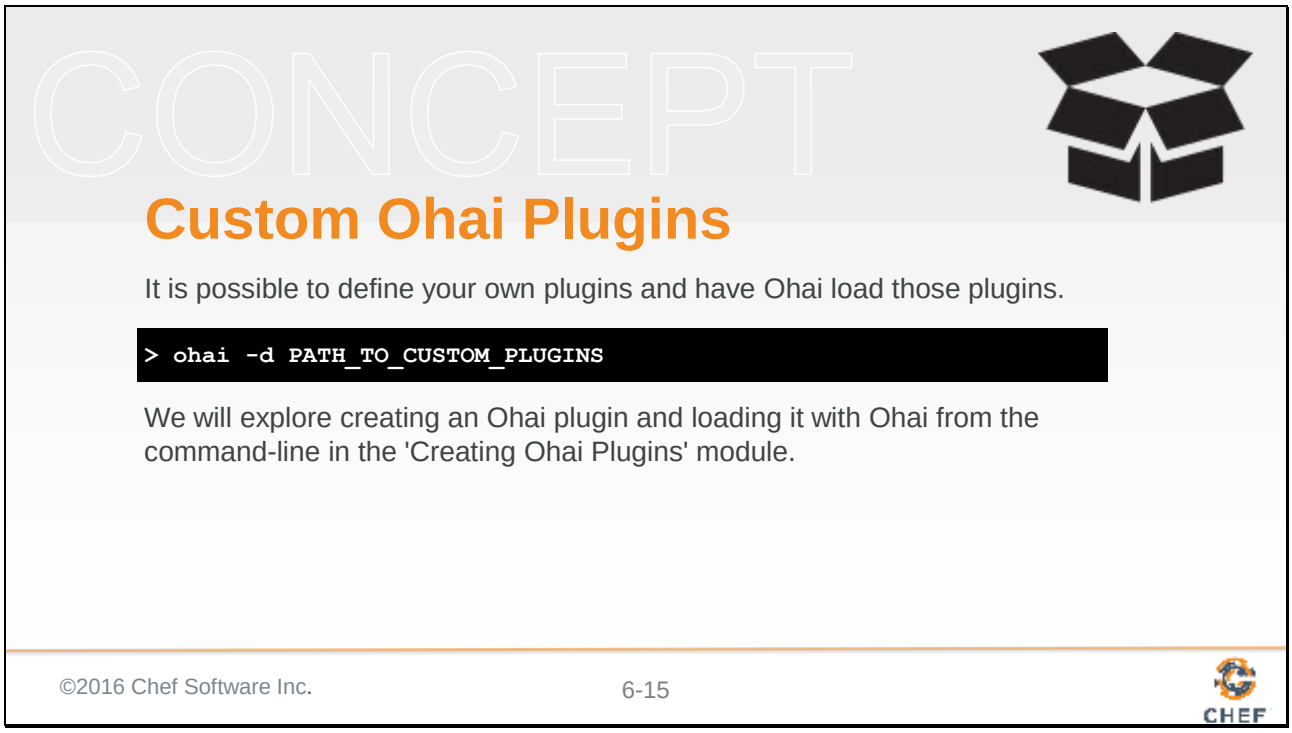
And finally if we wanted to display the Megahertz of that specific cpu we can append an additional key to the parameter.

Slide 14



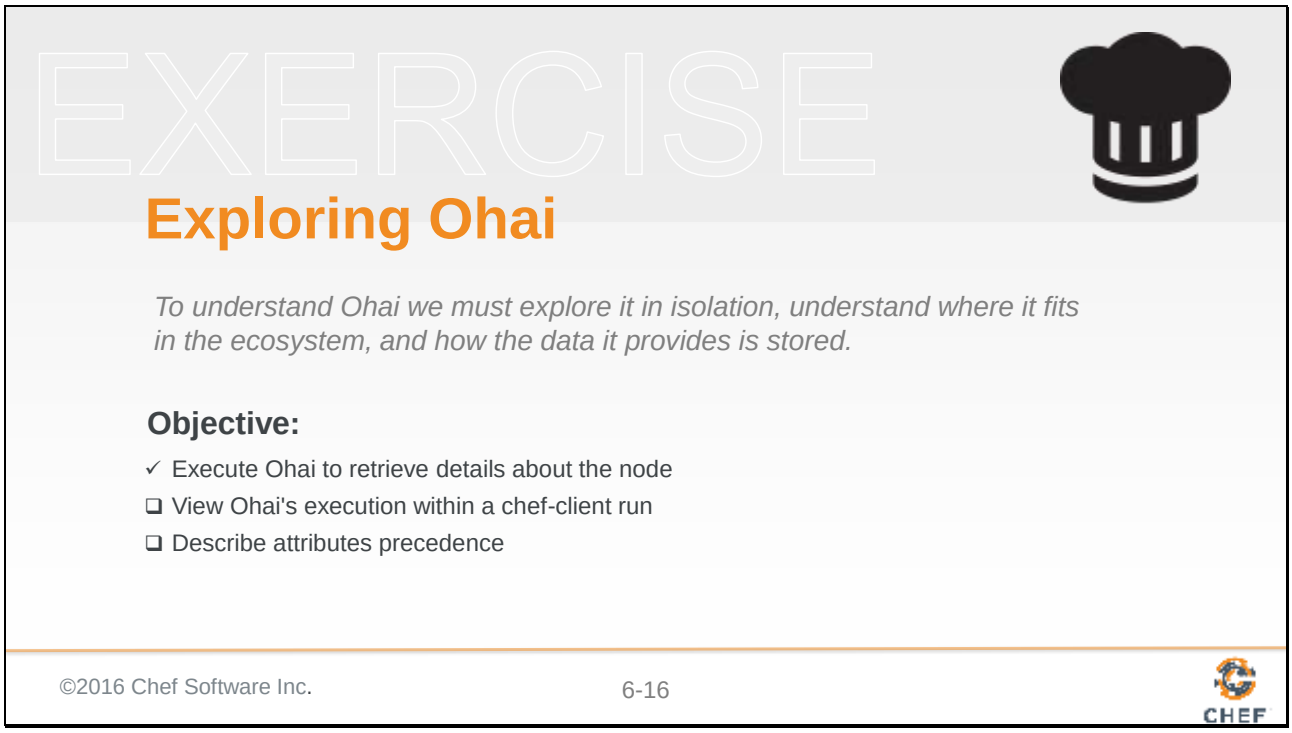
Ohai is composed of plugins that collect these different attributes. When you execute Ohai it will load the core plugins that are packaged with it.

Slide 15



Ohai is composed of plugins that collect these different attributes. When you execute Ohai it will load the core plugins that are packaged with it.

Slide 16



Executing ohai from the terminal gives you an idea about all the data that Ohai can provide for a system. Now it is important to see where this data is captured in the chef-client run.

Slide 17

CONCEPT

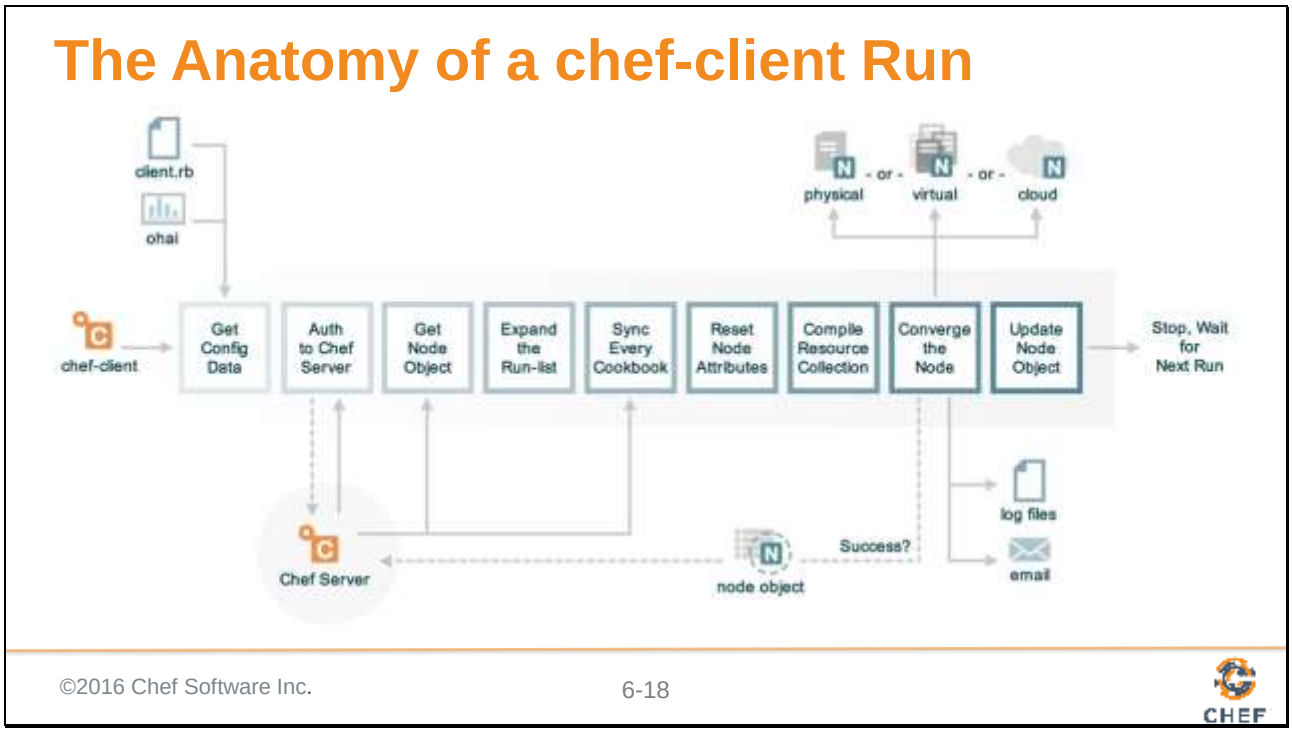
chef-client

chef-client automatically executes ohai and stores the data about the node in an object we can use within the recipes. When the chef-client run completes successfully the details about the node are sent to the Chef Server.

http://docs.chef.io/ohai.html

CHEF

Slide 18



chef-client automatically loads the Ohai libraries and executes them to capture details about the system. These details are stored in the node object which is available in the recipes that we define. At the end of a successful chef-client run this node object is sent to the Chef Server.

Running Ohai in Code

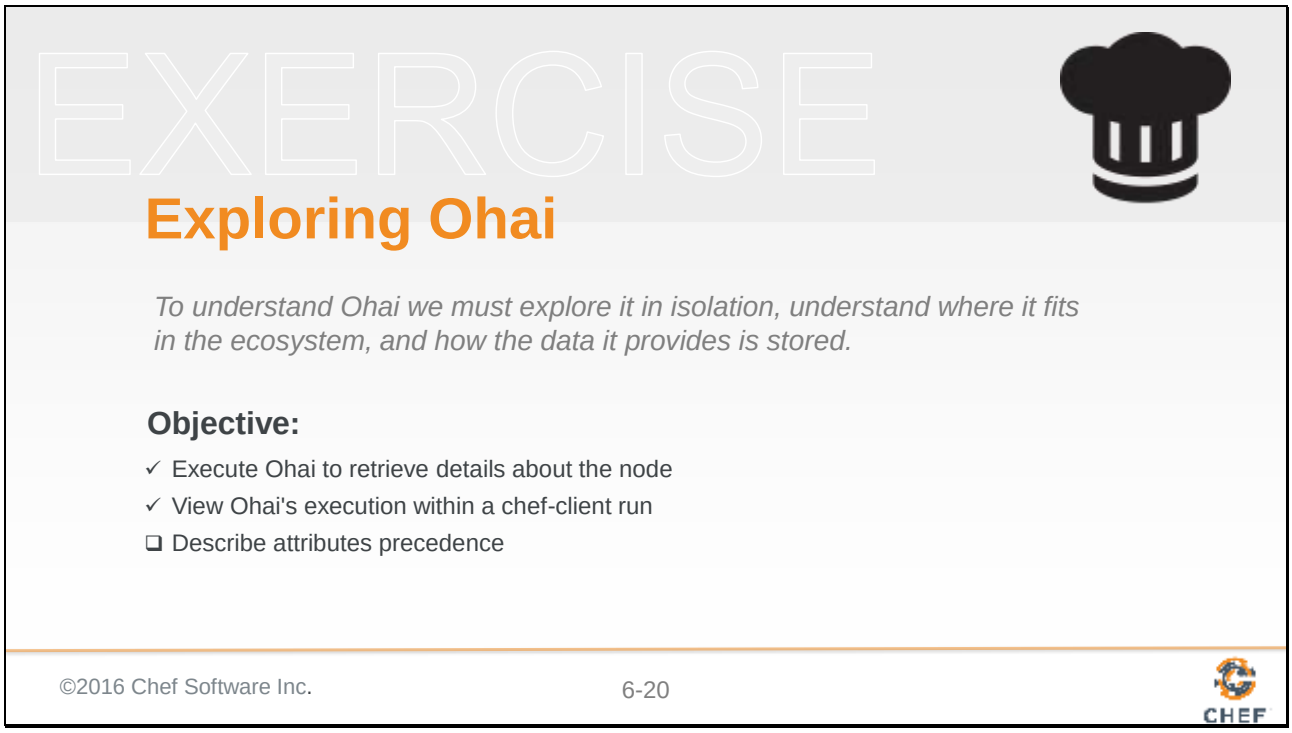
```
> chef exec pry
```

```
[1] pry(main)> require 'ohai'
=> true
[2] pry(main)> ohai = Ohai::System.new
=> #<Ohai::System:0x007fc62fadc490 @plugin_pat...@safe_run=true>>
[3] pry(main)> ohai.all_plugins('ipaddress')
[4] pry(main)> ohai.all_plugins('hostname')
[5] pry(main)> ohai.all_plugins('memory')
[6] pry(main)> ohai.all_plugins('cpu')
[7] pry(main)> ohai.all_plugins
[8] pry(main)> exit
```


chef-client run ohai in code as one of it's first steps. We can examine how that is done with Pry. Pry can be used as a debugger and as a REPL (Read-Evaluate-Print-Loop) tool. We can run this to allow to explore how Ohai is executed by the chef-client application.

First launch the session by running the specified command. Within this interactive session you can load the Ohai gem with the require command, create a new Ohai System object, and then ask the ohai object to load specific plugins or all plugins through the 'all_plugins' method. When you are done you can exit by entering the command 'exit'.

Slide 20



chef-client loads and executes Ohai within Ruby. Ohai returns Ruby object representations of the data that chef-client is able to evaluate and store within the node object. These attributes discovered by Ohai become attributes of the node object and it is important to take a quick moment to discuss how these attributes compare to the other attributes that may be defined in other locations within cookbooks, roles, and environments.

Slide 21

CONCEPT

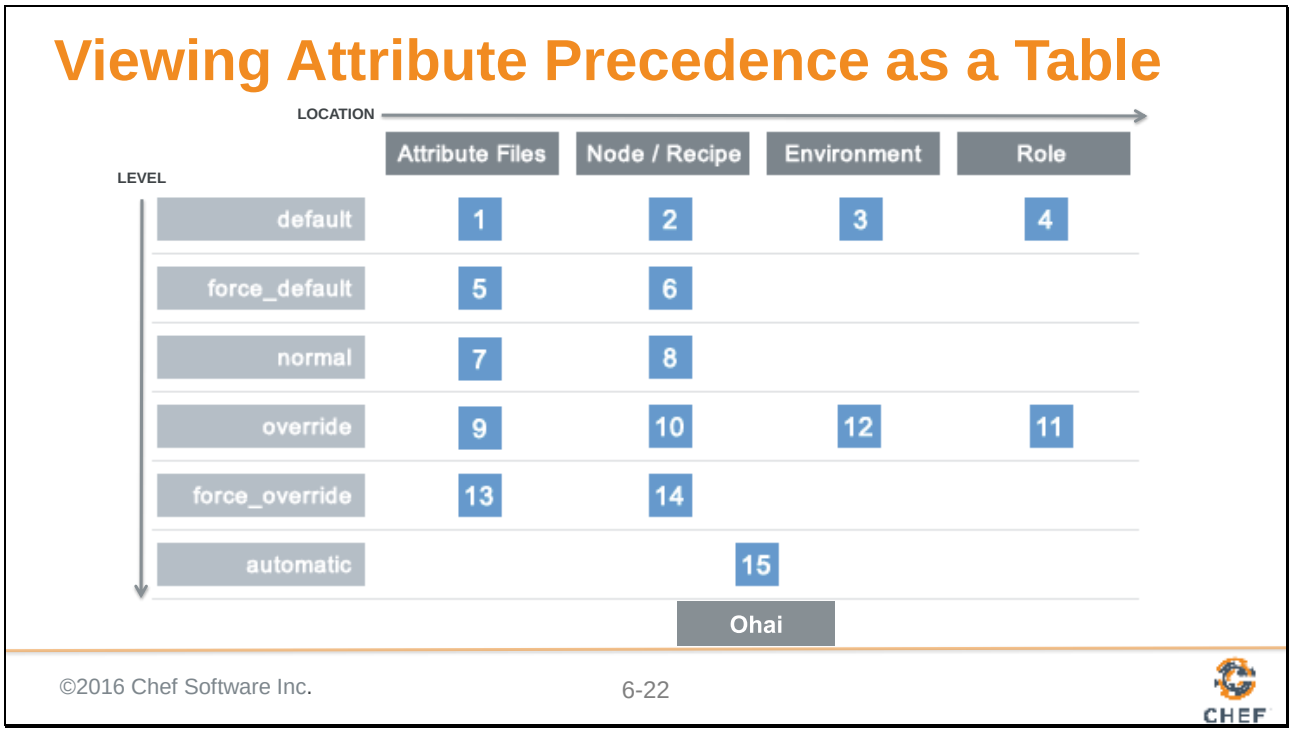
Node Attributes

A node object maintains the attributes collected from Ohai, from the previous node object (as returned by the Chef Server), from the environments, roles, and the cookbooks defined in the run list.

CHEF

Later within the chef-client a node object is created with the attributes collected from Ohai, the values previously stored on the Chef Server, and then the attributes defined in the environments, roles and cookbooks described in the node's run list. The node prioritizes and gives precedence to the attributes collected by Ohai.

Slide 22

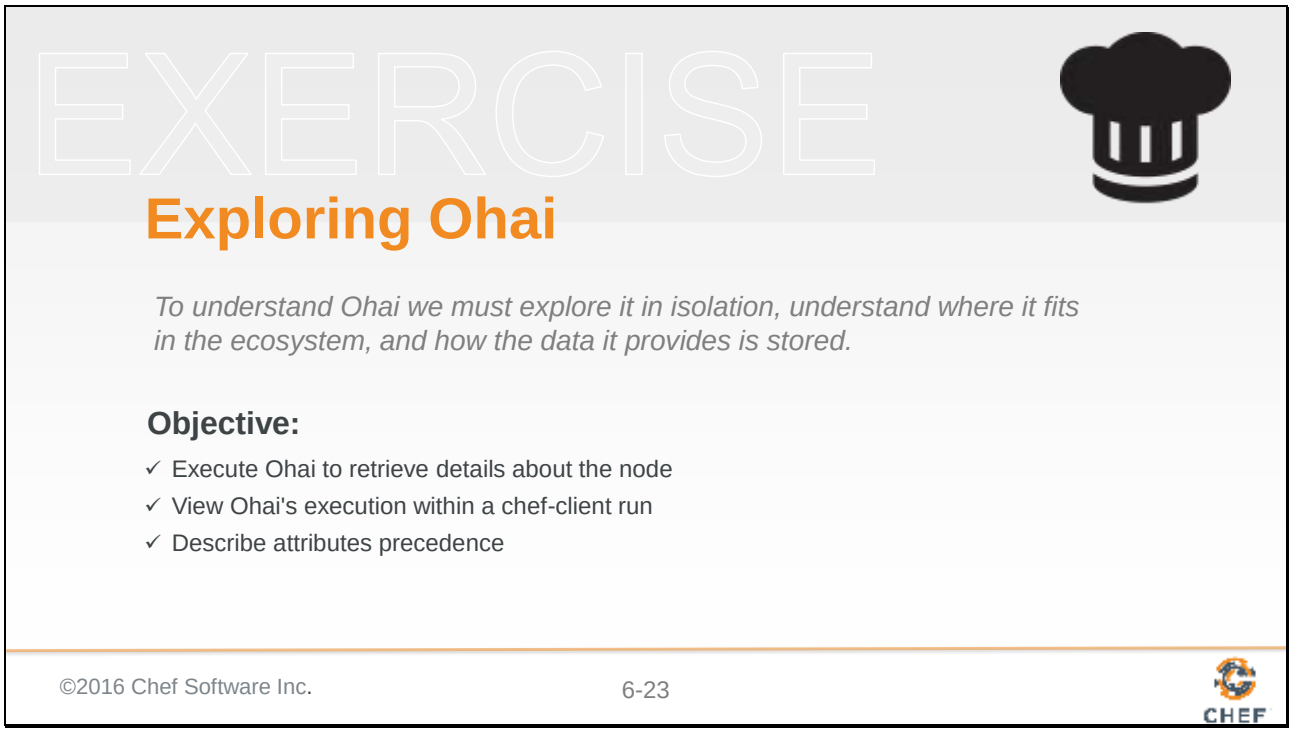


This is a table representation of the various levels of precedence that can be specified with the location in which it can be specified. The lower the value, the lower the precedence. The higher the value, the higher the precedence.

The attributes collected from Ohai are considered automatic attributes granting them the value of 15. This means all data collected through Ohai attributes cannot ever be overridden.

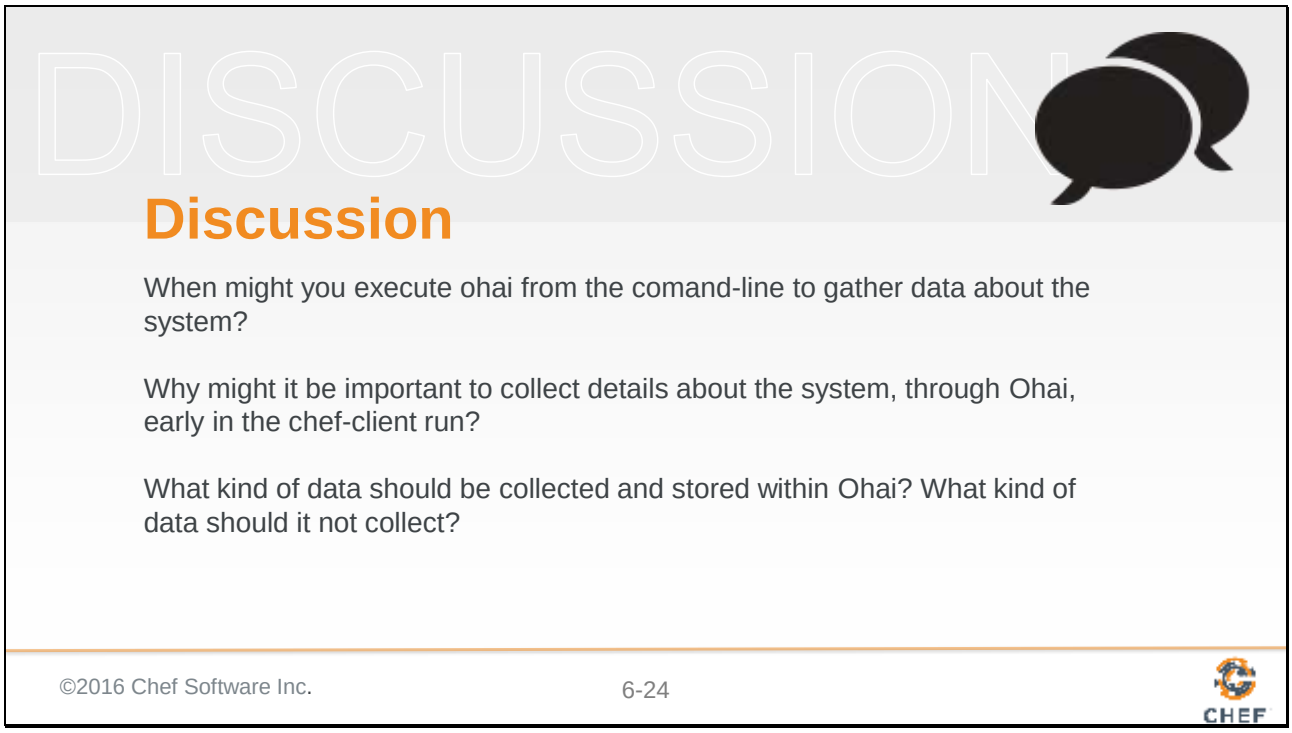
That should make sense based on the data that we have queried so far in this module (e.g. CPU, memory). Never would we want to have an attribute defined in a cookbook or environment override this data collected about our system. This is also important when considering whether you want to create an Ohai plugin. The kind of data that you want to collect should not be data that you will want to override as it is data that describes the system and not data that you want to configure the system.

Slide 23



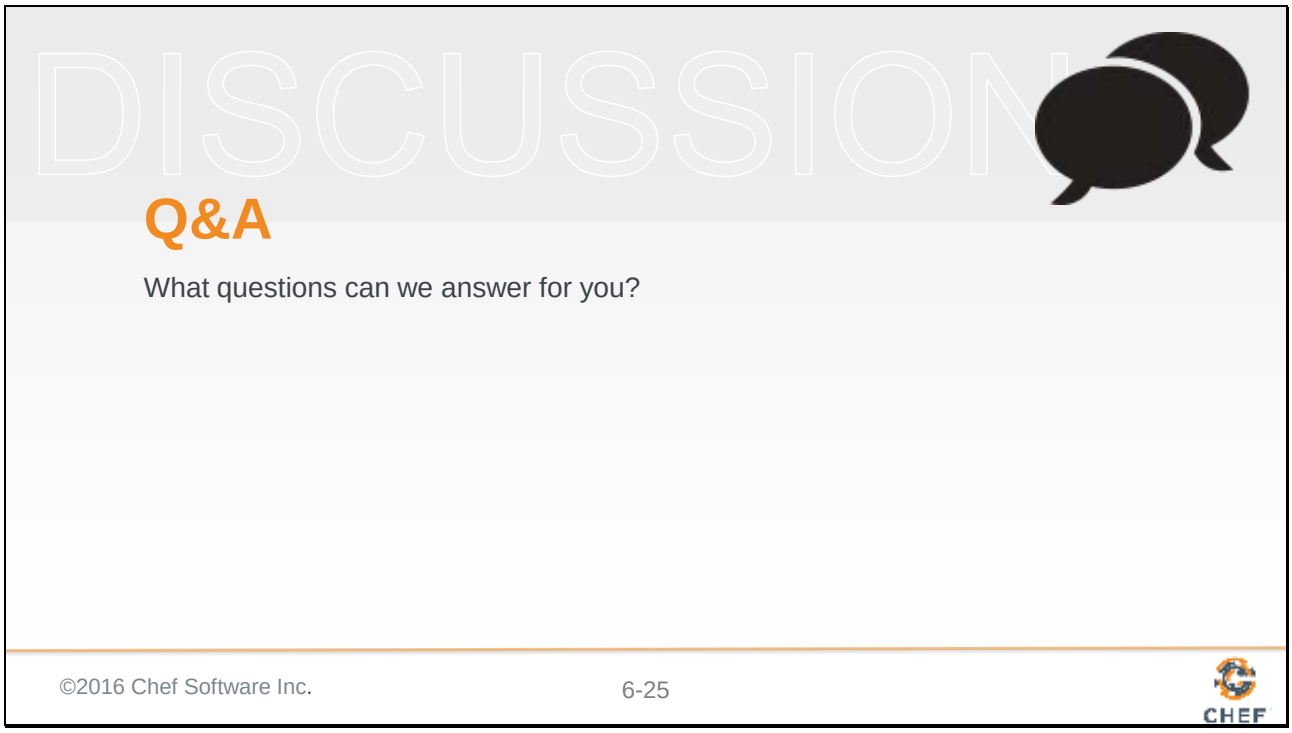
We have seen how to use Ohai as a command-line tool, explored how chef-client uses it, and seen the precedence level at which this data is stored. In the next module we will discuss Ohai's plugin history, its plugin structure, and the DSL (Domain Specific Language) it provides to express these plugins.

Slide 24



Let's finish with a discussion.

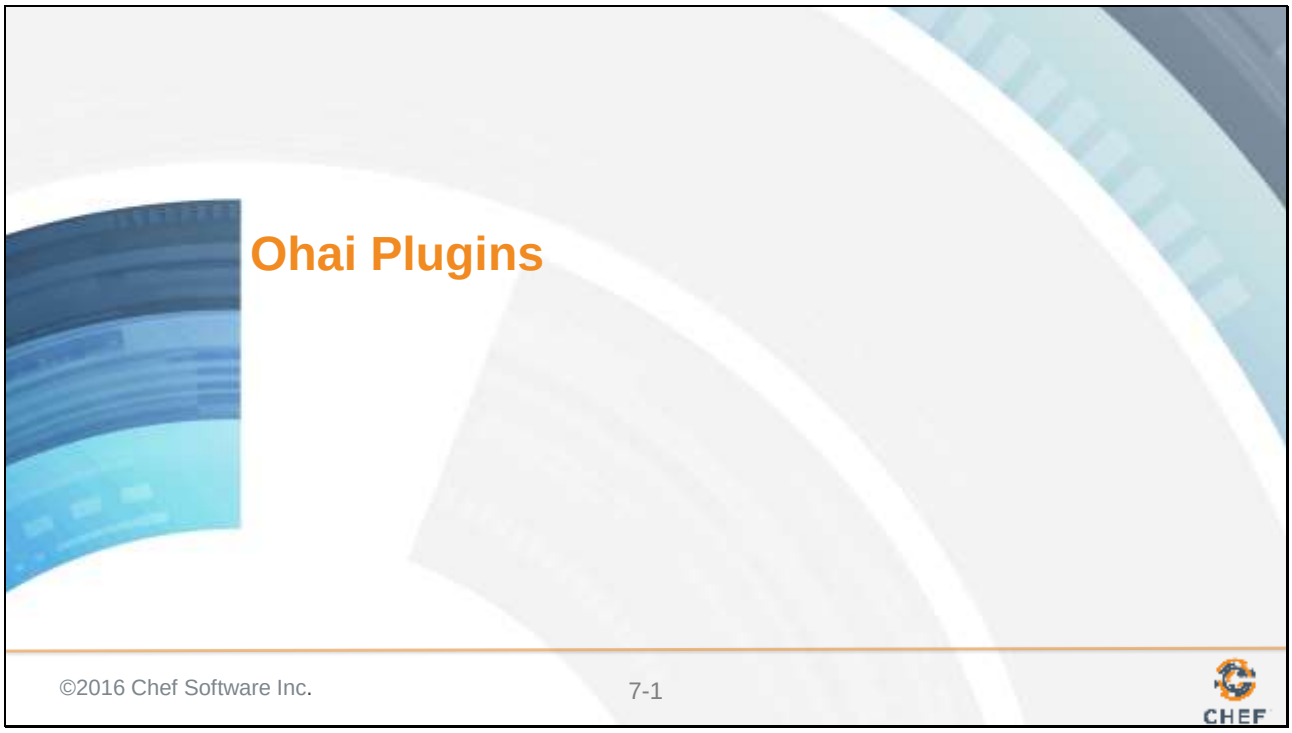
Slide 25



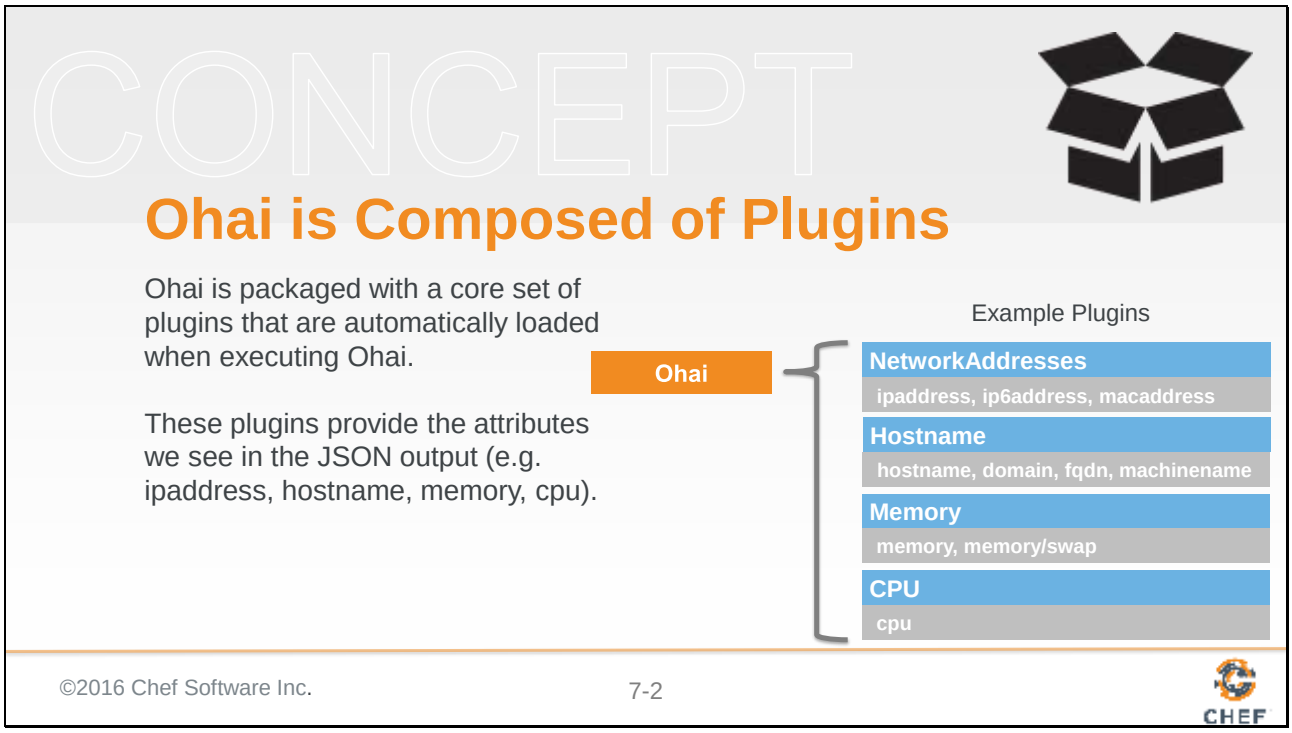
What questions can we answer for you?

Slide 26

7: Ohai Plugins



Slide 2



As we saw in the previous module Ohai provides a large set of attributes that it provides through plugins. All the data that Ohai collects are stored in plugins. Ohai comes packaged with a core set of plugins that capture a lot of common data across many different platforms.

Slide 3

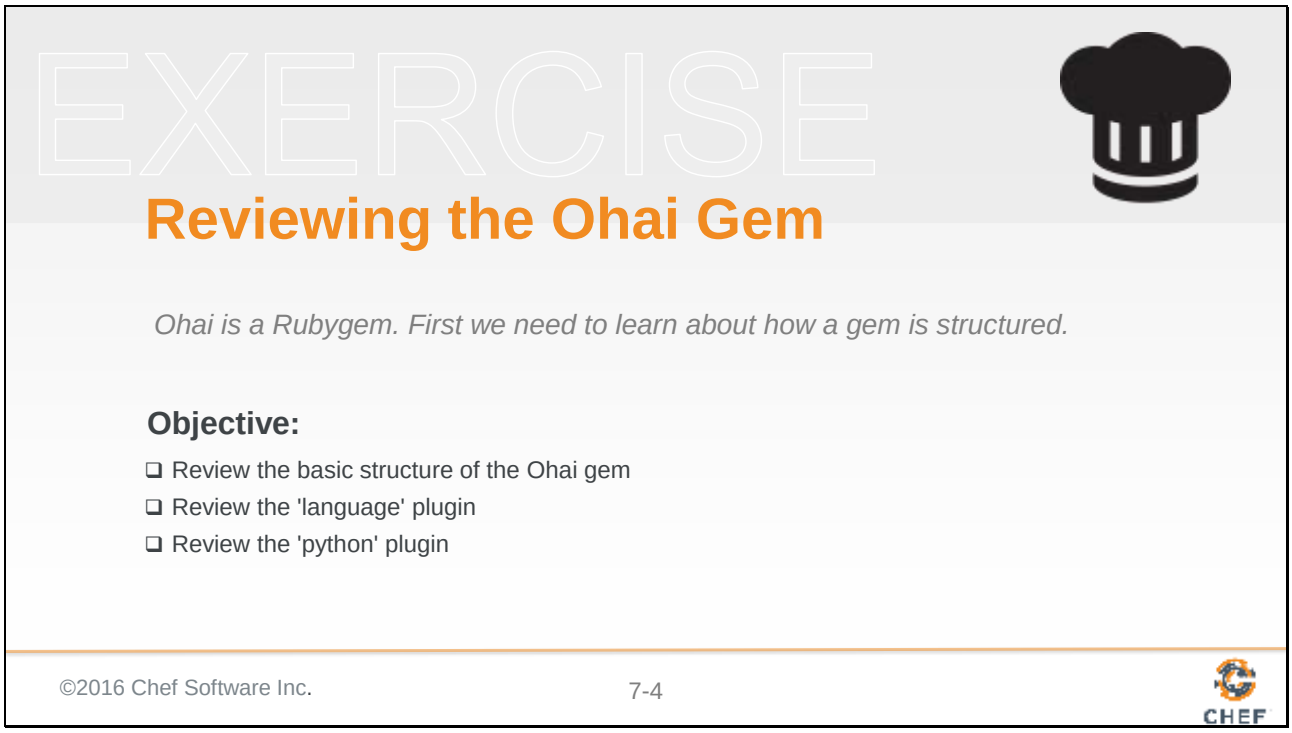
Objectives

After completing this module, you should be able to:

- Find Ohai's core plugins
- Express what a plugin provides, depends on, and how it collects its data

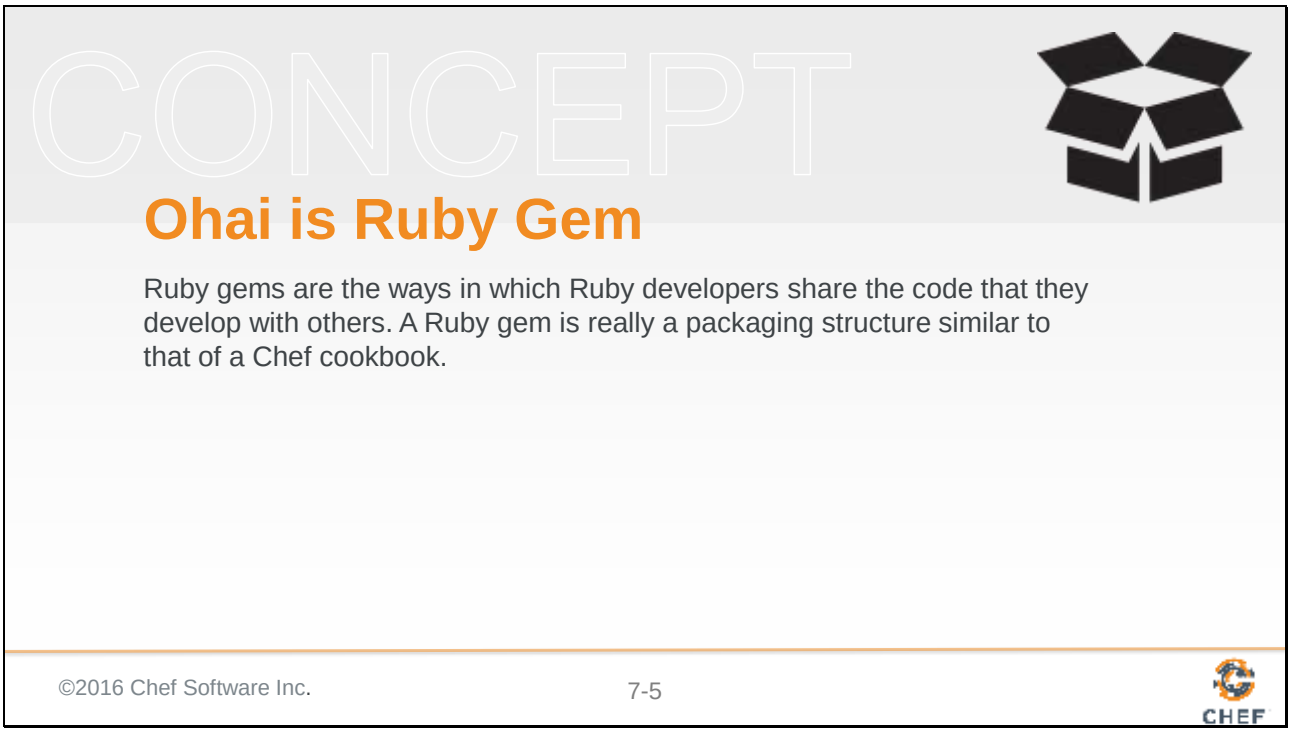
After completing this module you will be able to find the plugins that come packaged core with Chef, express what a plugin provides, depends on, and how it collects its data

Slide 4



To review the core plugins packaged with Ohai we need to spend some time reviewing the source code of the gem as none of the gems are not defined in documentation.

Slide 5



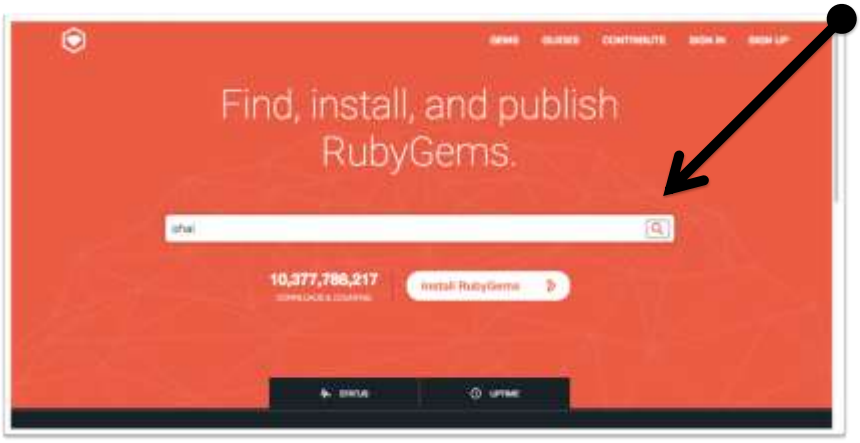
Ohai is a Ruby gem that is packed in the Chef Development Kit (Chef DK). A Ruby gem is a packaging structure that allows for the code to be reused and shared.

Slide 6

Searching for a Gem on Rubygems

Steps

1. Visit https://rubygems.org
2. Within the search field enter: **ohai**
3. Press enter or click the magnified glass at the right-side of the search box.
4. Click the 'Source Code' link
5. Click on 'Clone or download' and then copy the git URL.



CHEF

All Rubygems are stored on rubygems.org. We can come to the site and search for any gem by their name. Search for the Rubygem named "ohai".

Slide 7

Searching for a Gem on Rubygems

Steps

1. Visit https://rubygems.org
2. Within the search field enter: **ohai**
3. Press enter or click the magnified glass at the right-side of the search box.
4. Click the 'Source Code' link
5. Click on 'Clone or download' and then copy the git URL.

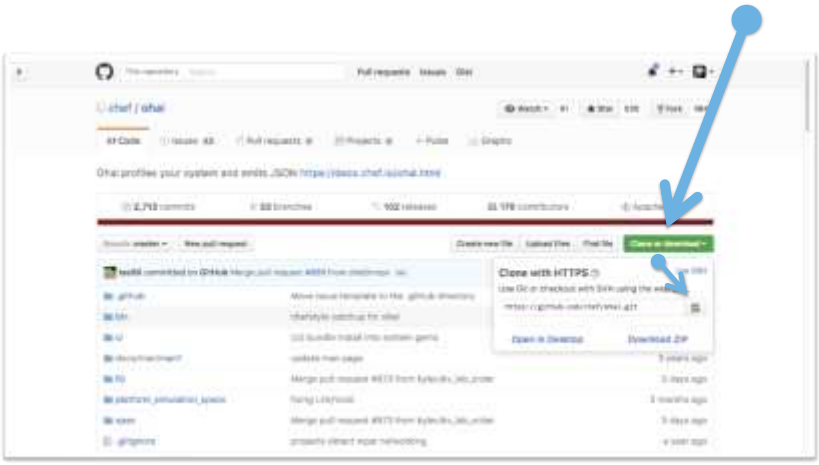
The project page for the gem itself contains important information about the releases, where to find the source, where to file issues, etc. We are interested in viewing the source of the project so we want to click on that link.

Slide 8

Searching for a Gem on Rubygems

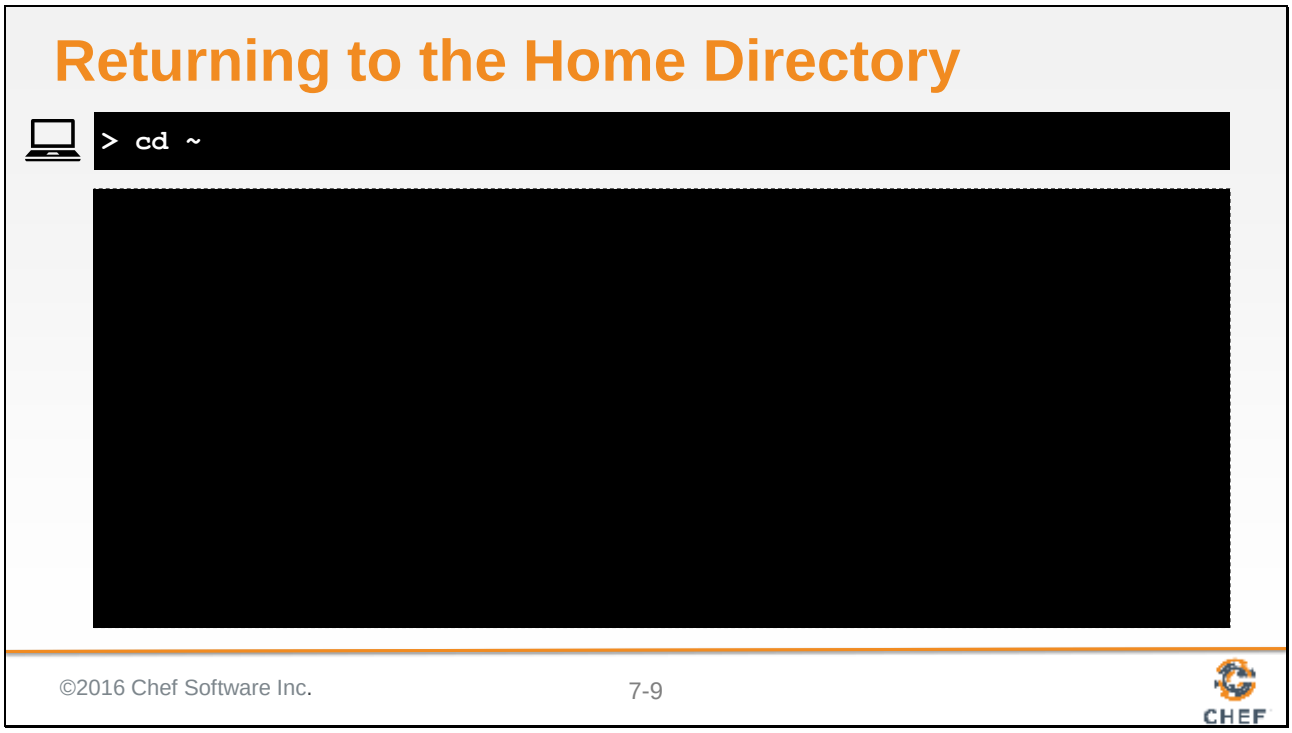
Steps

1. Visit https://rubygems.org
2. Within the search field enter: **ohai**
3. Press enter or click the magnified glass at the right-side of the search box.
4. Click the 'Source Code' link
5. Click on 'Clone or download' and then copy the git URL.

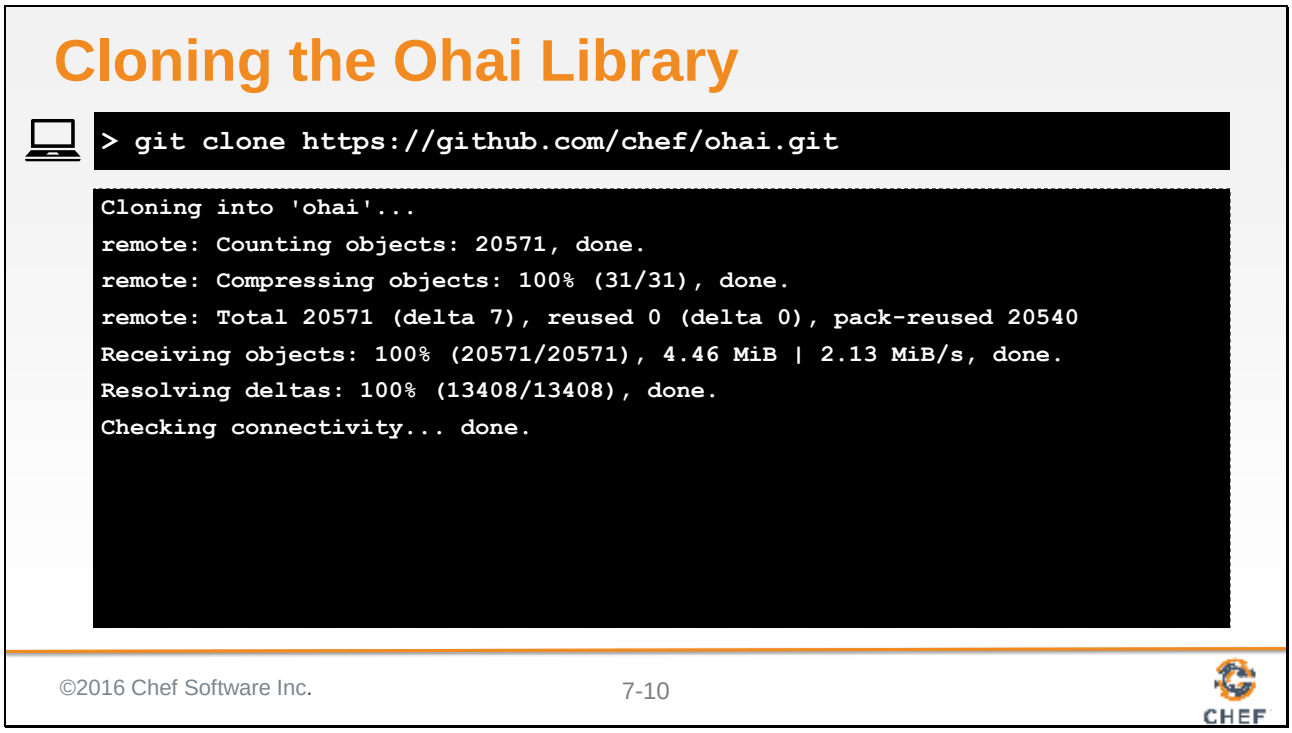


The ohai project is stored as a git repository within the Chef organization on GitHub. We can clone this project to our workstation to give us the ability to review the source code.

Slide 9



We are going to obtain the Ohai library on our local workstation so let's start by returning to the home directory.



Git is installed with the Chef DK so we will use it to clone the Ohai project.

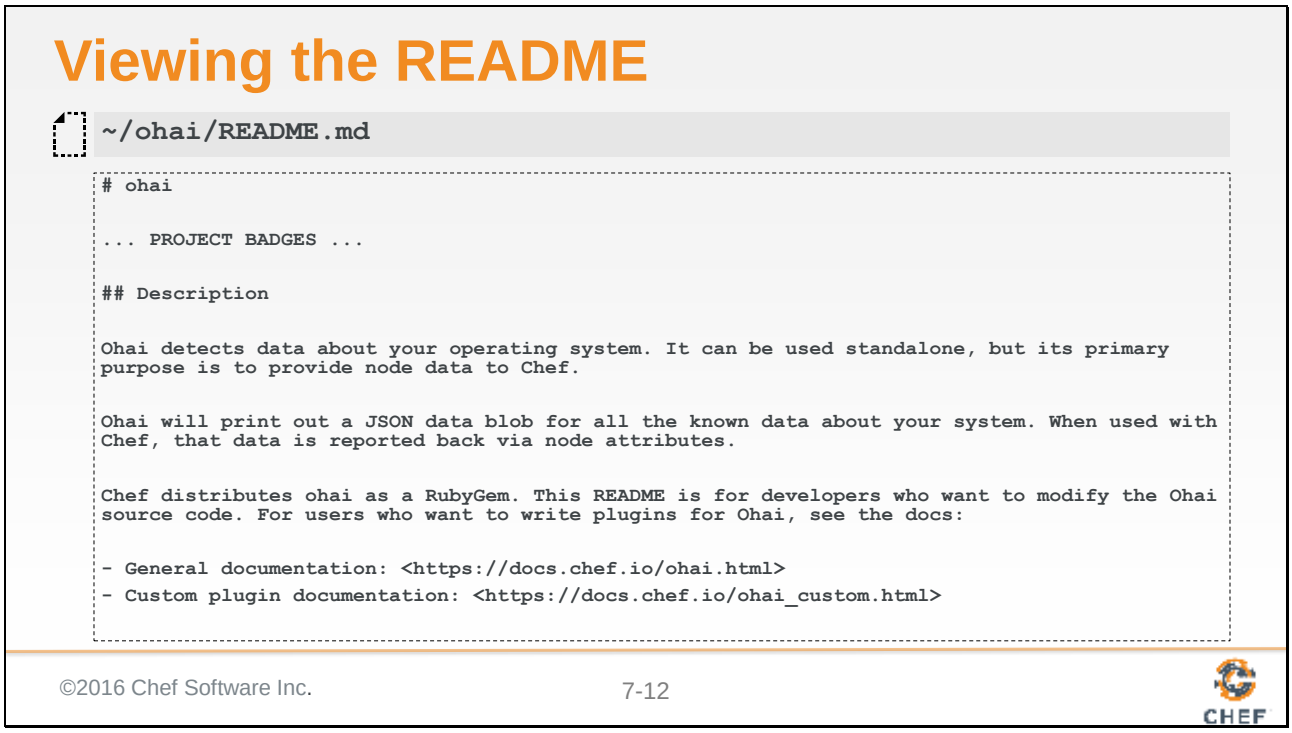
Viewing the Contents of the Project

```
> tree ohai
```

```
ohai
├── CHANGELOG.md
├── DOC_CHANGES.md
├── Gemfile
├── LICENSE
├── NOTICE
├── OHAI_MVPS.md
├── README.md
├── RELEASE_NOTES.md
├── Rakefile
├── appveyor.yml
├── bin
```


CHEF

The gem contains several important items within the top-level directory. We are going to explore the contents of some of the essential files.



The README contains information on how to install, configure and use this gem. This is often the place to start when exploring the gem.

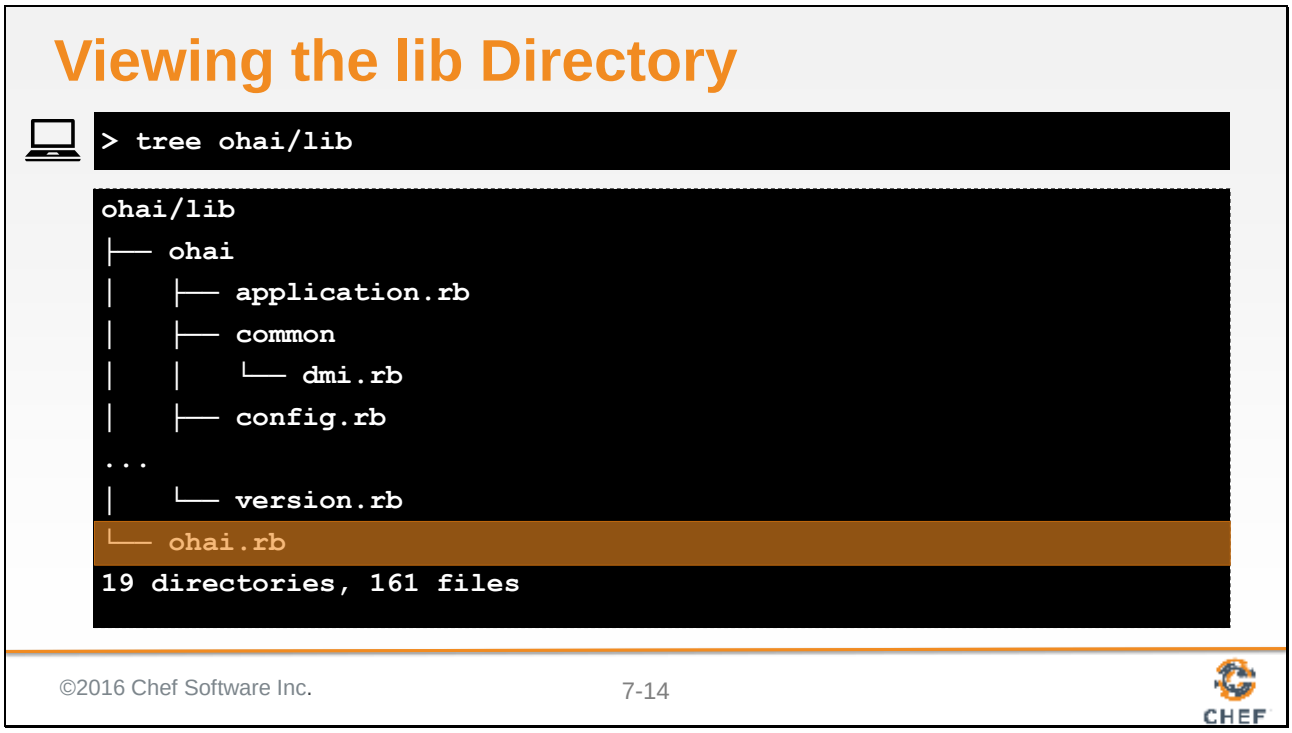
Viewing the Gem Specification

~/ohai/ohai.gemspec

```
$:.unshift File.expand_path("../lib", __FILE__)
require "ohai/version"

Gem::Specification.new do |s|
  s.name = "ohai"
  s.version = Ohai::VERSION
  s.platform = Gem::Platform::RUBY
  s.summary = "Ohai profiles your system and emits JSON"
  s.description = s.summary
  s.license = "Apache-2.0"
  s.author = "Adam Jacob"
  s.email = "adam@chef.io"
  s.homepage = "https://docs.chef.io/ohai.html"

  s.required_ruby_version = ">= 2.1.0"

  s.add_dependency "systemu", "~> 2.6.4"
```


CHEF

The gem specification defines important information about the Rubygem. Within it you will find metadata that describes the owner, licensing, contact information, dependencies, development dependencies, the files to package in the gem, and which one of those are executables.



The lib (or library) directory contains the source code for this gem. Within the root of the directory you will find a single file that shares the same name as the gem.

In the previous module when we typed "require 'ohai'" this was the file that was loaded into memory.

Viewing the ohai.rb file in the lib Directory

~/ohai/lib/ohai.rb

```
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

require "ohai/version"
require "ohai/config"
require "ohai/system"
require "ohai/exception"
```


This file requires more files from within the gem. The paths specified are relative to the 'lib' directory so all of these examples are loading files from within the subdirectory of ohai.

Viewing the plugins directory

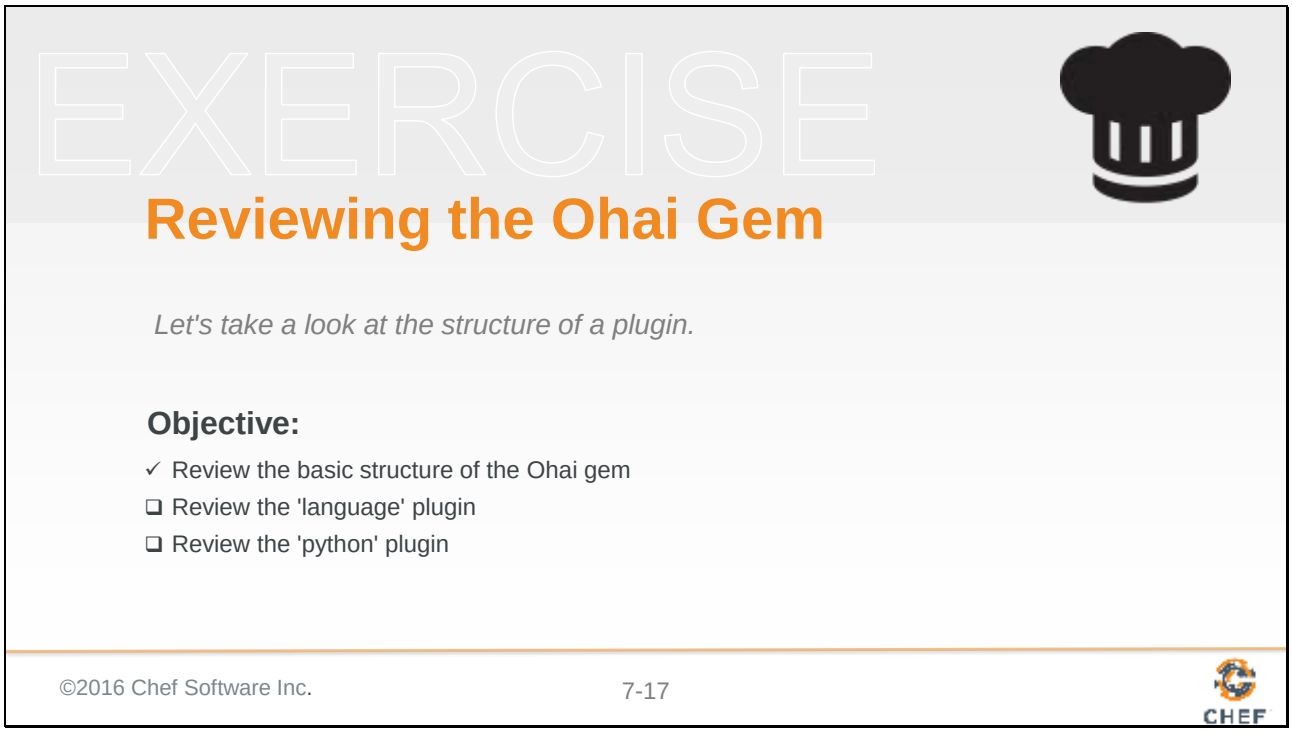
```
> tree ohai/lib/ohai/plugins
```

```
ohai/lib/ohai/plugins
├── aix
│   ├── cpu.rb
│   ├── filesystem.rb
│   ├── kernel.rb
│   ├── memory.rb
│   ├── network.rb
│   ├── os.rb
│   ├── platform.rb
│   ├── uptime.rb
│   └── virtualization.rb
├── azure.rb
├── bsd
│   ├── filesystem.rb
```


CHEF

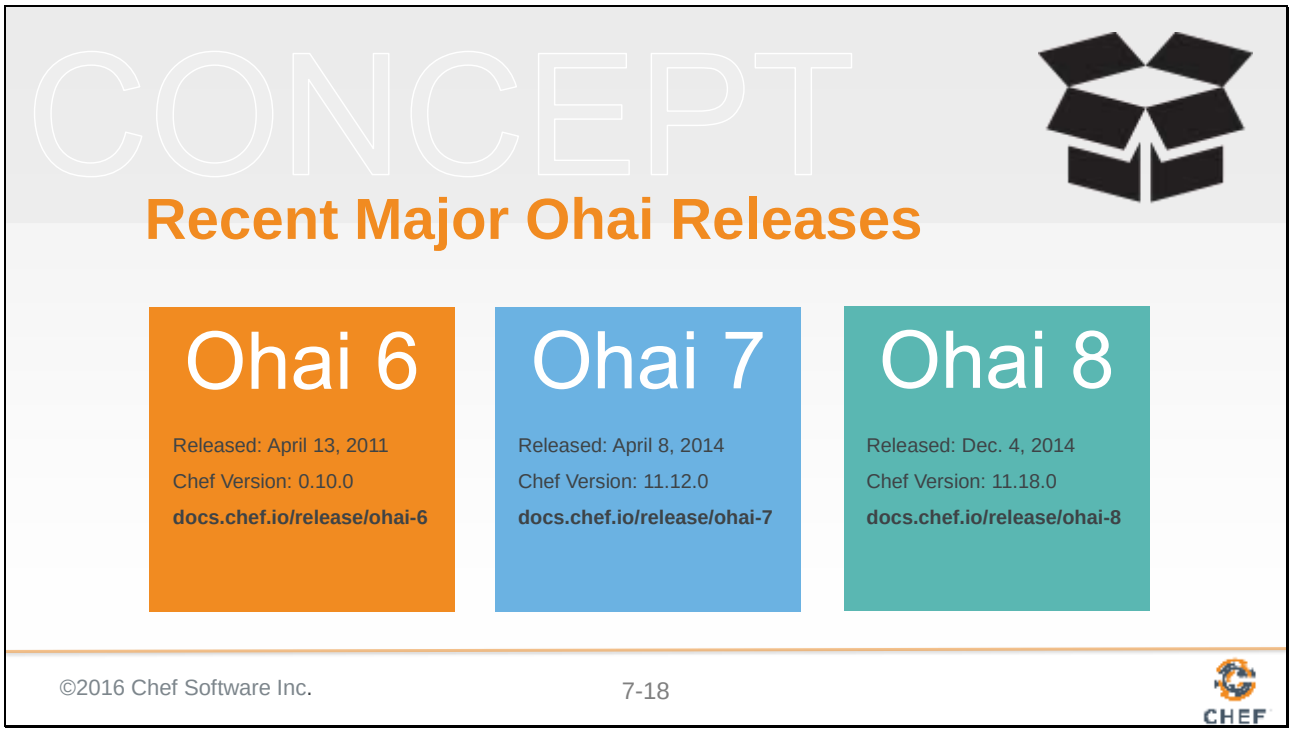
Ohai stores its plugins in a specific subdirectory of this project.

Slide 17



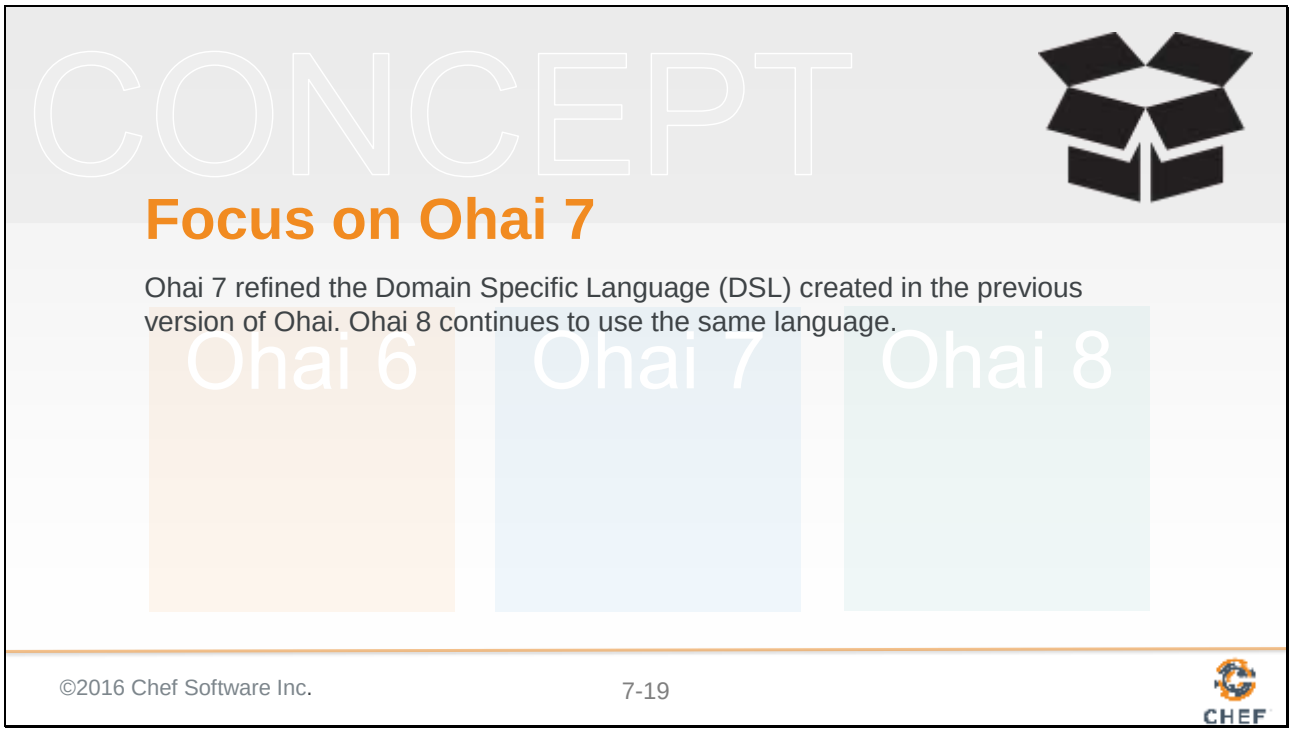
That was a quick introduction to the gem structure to give us an idea about where the plugins are stored. Now it is time to explore the Domain Specific Language (DSL) used to write these plugins.

Slide 18



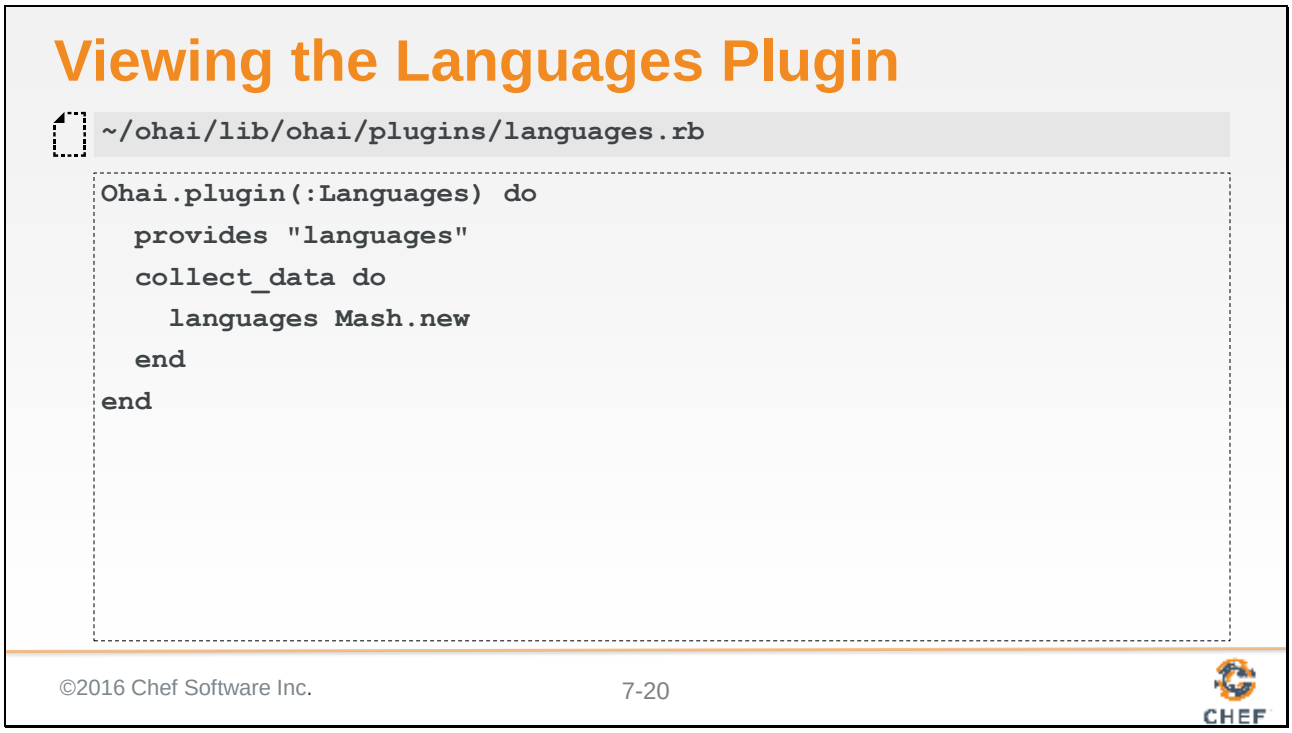
Ohai has seen many notable releases. Depending on the version of Chef you are using within your organization may dictate which version of Ohai is being used.

Slide 19



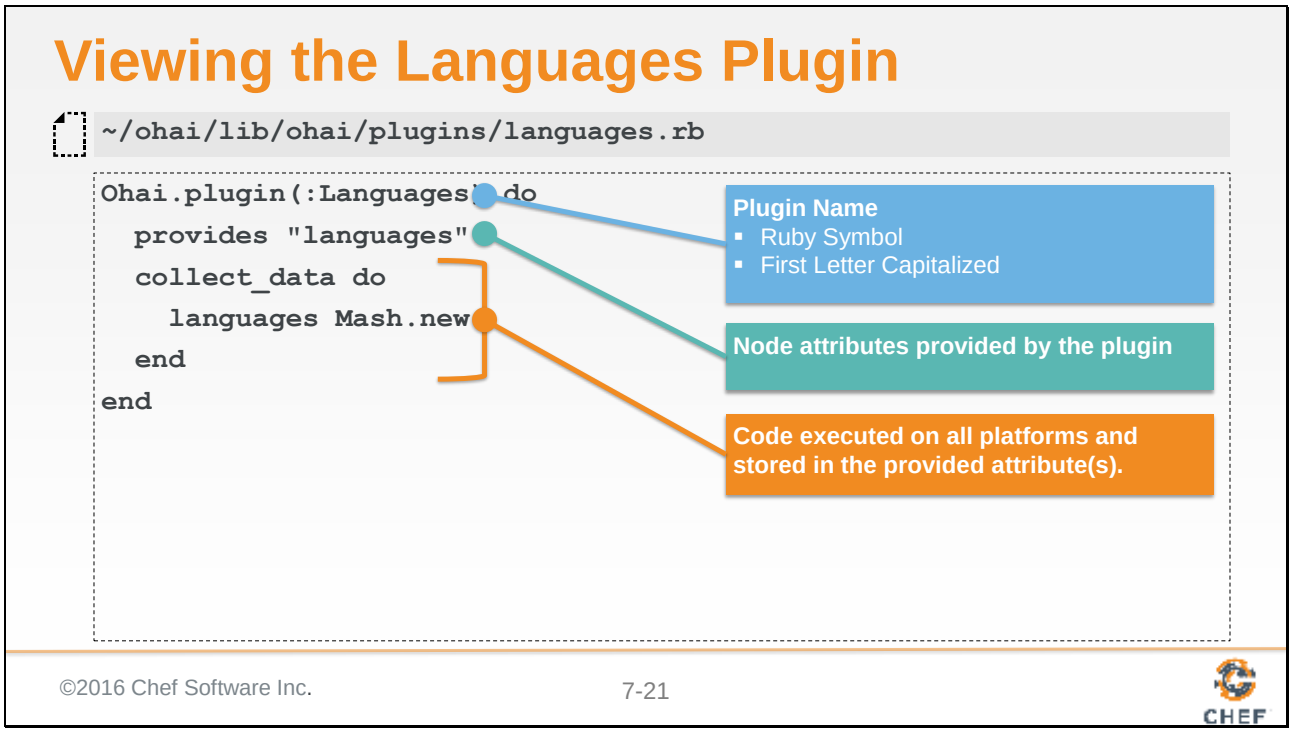
Ohai 6 introduced the ability to express plugins through a DSL. Ohai 7 refined that DSL. Ohai 8 continues to use that same language. The following slides and our exercise in the next module will focus on the DSL defined in Ohai 7.

Slide 20



Let's load the languages plugin and review the basic structure of the plugin.

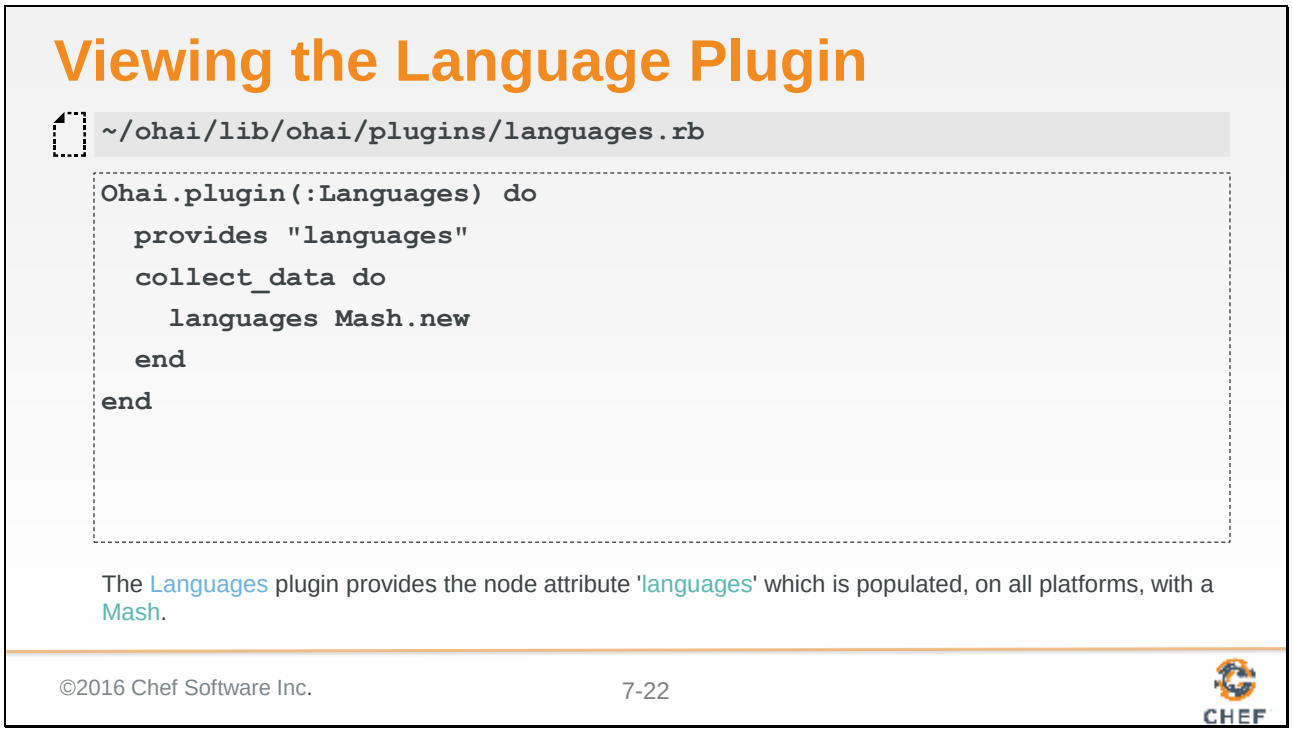
Slide 21



A plugin starts with invoking a method on the Ohai class with a single parameter. That parameter provided is the symbol name of the plugin. All Ohai plugins must have a symbol name with the first letter capitalized.

The remainder of the plugin is defined within the block of the 'plugin' method. The 'provides' method specifies what attribute or attributes the plugin will be added to the node object. The 'collect_data' method defines a block which contains the code that is executed on all platforms. This block of code will often times set the values of the attributes the plugin provides.

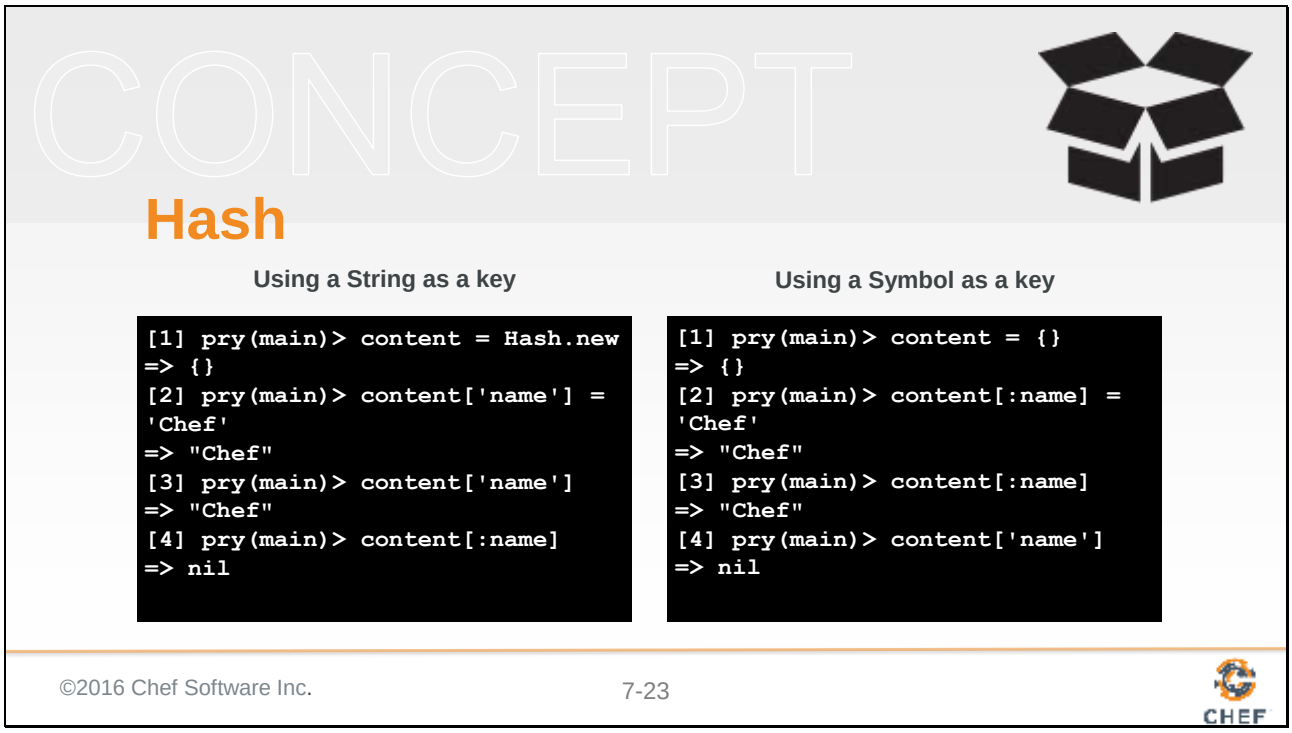
Slide 22



This plugin is named Languages. It provides the languages attribute on the node. This languages attribute is populated with the contents of a new Mash.

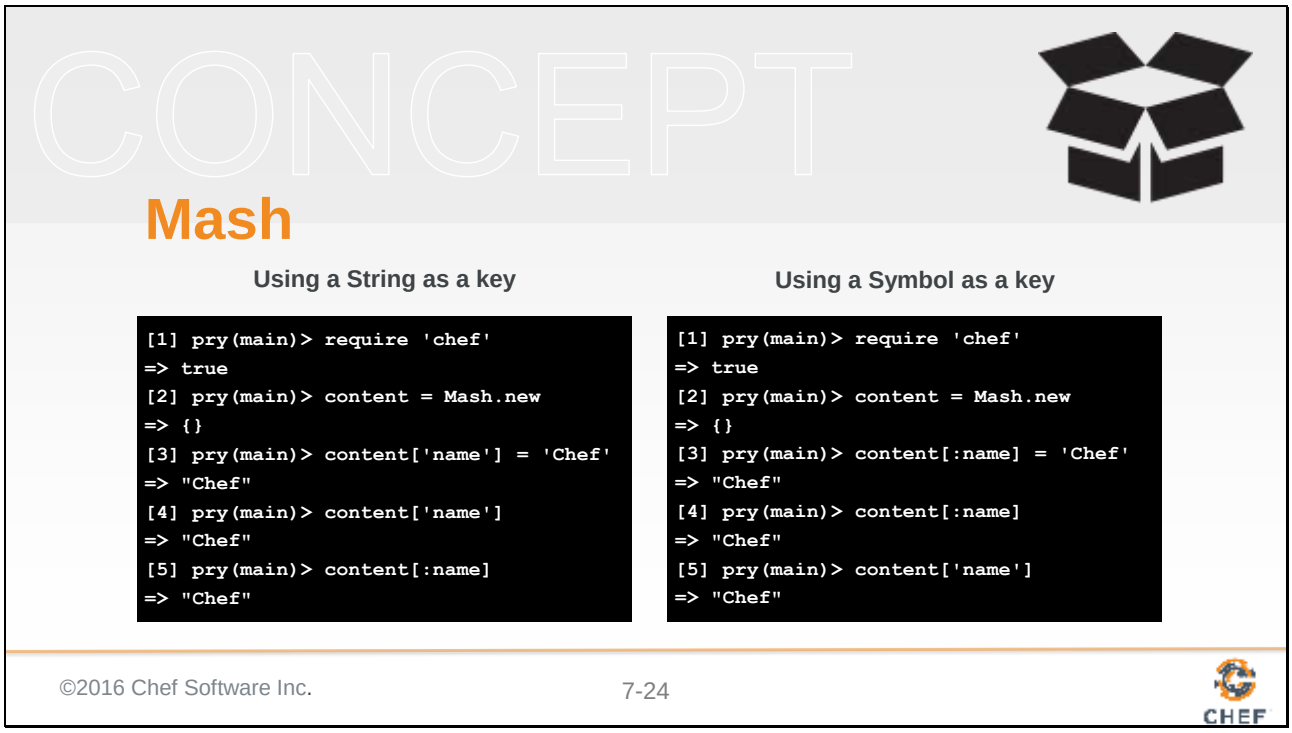
But what is a Mash?

Slide 23



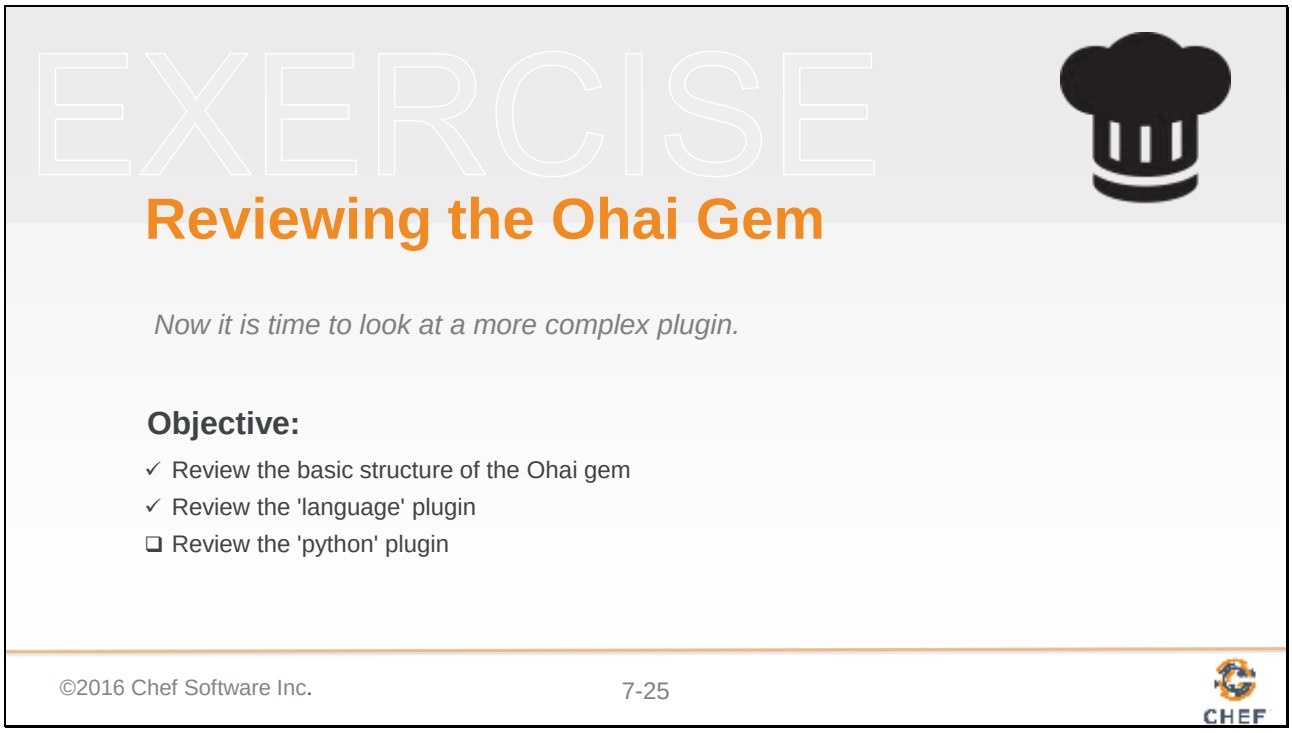
To understand what a Mash is first let's talk about Ruby's Hash. Hashes allow you to store values with a key; often times these keys are Ruby Strings or Ruby Symbols. When you want to retrieve that value you need to provide the same key. So if say you stored data with a Symbol key it is only retrievable with a Symbol key. The same could be said for using a String key.

Slide 24



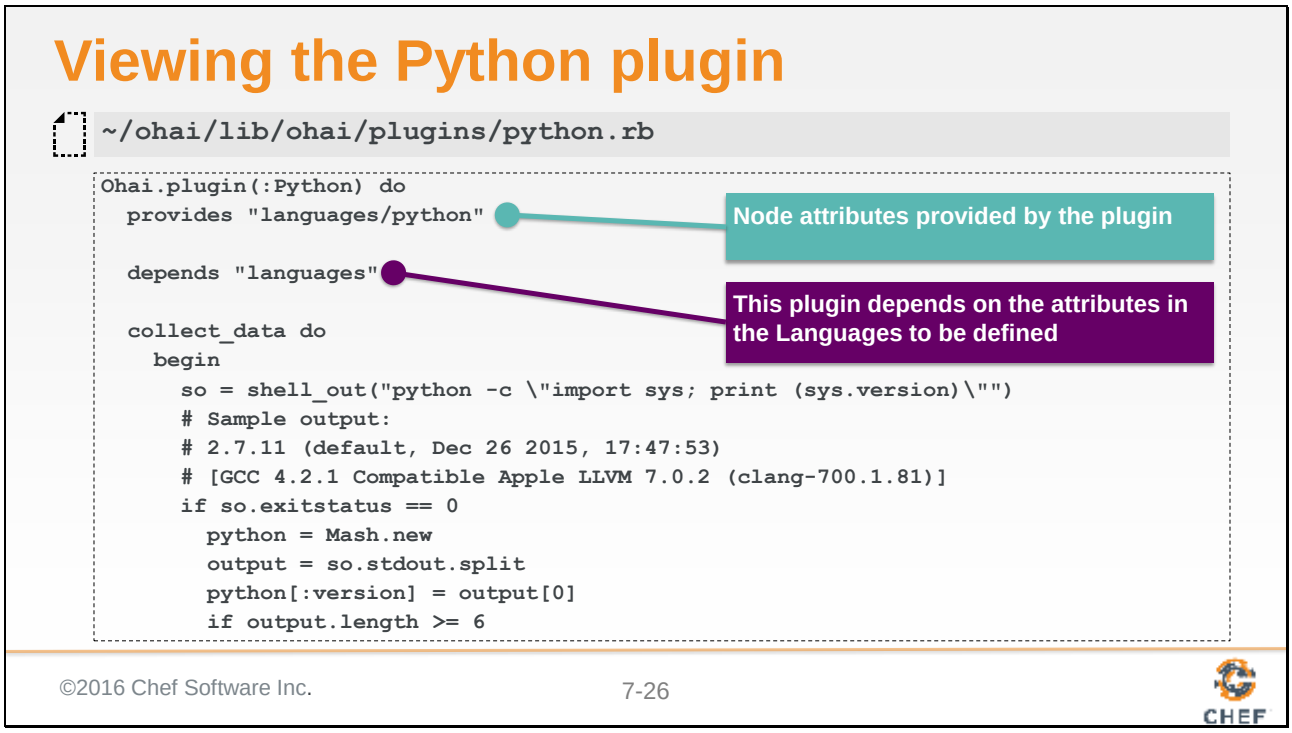
A Mash is similar to a Ruby Hash except that it is indifferent to whether you provide it a String key or Symbol key. Either of those types of keys will return value stored by the other. This more lenient data structure allows for these two keys to be used interchangeably. Allowing us to use whichever key style we prefer without being penalized if we were to guess the key style that differs from other plugins.

Slide 25



The language plugin is small plugin that setups up a data structure for other language plugins to add more information to it. Let's review a specific language plugin to see a more complex implementation.

Slide 26



Here within the Python plugin we see the same structure with a dependency and a significant amount of work being done in the 'collect_data' method block. The attribute provided by this plugin can be found on the node object under the specified path. Remember this is the same path structure you use on the command-line when wanting to traverse the attributes provided.

The dependency described here states that this plugin requires that the node attribute value 'languages' must be defined first before this plugin will execute. Ohai will determine how to execute the plugins based on these dependencies.

Viewing the Python plugin

~/ohai/lib/ohai/plugins/python.rb

```
Ohai.plugin(:Python) do
  provides "languages/python"

  depends "languages"

  collect_data do
    begin
      so = shell_out("python -c \"import sys; print (sys.version)\"")
      # Sample output:
      # 2.7.11 (default, Dec 26 2015, 17:47:53)
      # [GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)]
      if so.exitstatus == 0
        python = Mash.new
        output = so.stdout.split
        python[:version] = output[0]
        if output.length >= 6
```


CHEF

Within the collect_data block we use a helper method named 'shell_out'. This 'shell_out' method accepts a single parameter which is the command to run. This 'shell_out' method will generate an object for which you can ask for the standard output, standard error, and the exit status.

This command is executed and if the status is successful (0 status code) then look at the standard output, split it into multiple lines, extract the version and possibly any build date information, and then store that information into the Mash that was created by the Languages plugin. If a failure occurs at any point catch that error and display a debug message.

You will find that most Ohai plugins will fit the following pattern. Perform a system related call to collect some data, use Ruby to process that data, and then store the data.

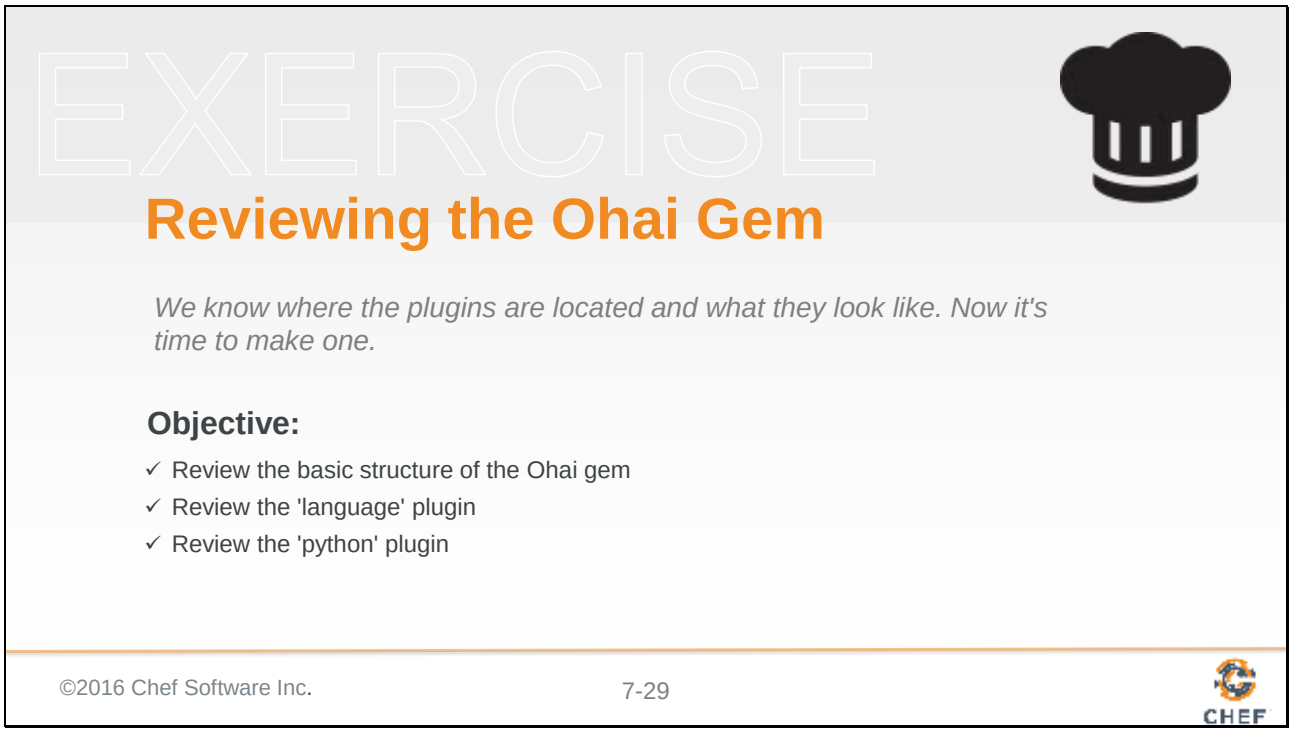
CONCEPT

collect_data for a specific Platform

Plugins can collect data in different ways across different platforms. When defining a collect_data block if you do not provide any arguments it is assumed the default and all platforms unless you define a collect_data block specific for a platform.

CHEF

Slide 29



We were able to view the contents of the gem and examine the contents of a few plugins to give us an understanding of how plugins are structured. Now it is time for use to create our own.

Discussion

How are the structures of a Rubygem and a Cookbook similar to each other?

What are the requirements when specifying the name of a Ohai plugin?

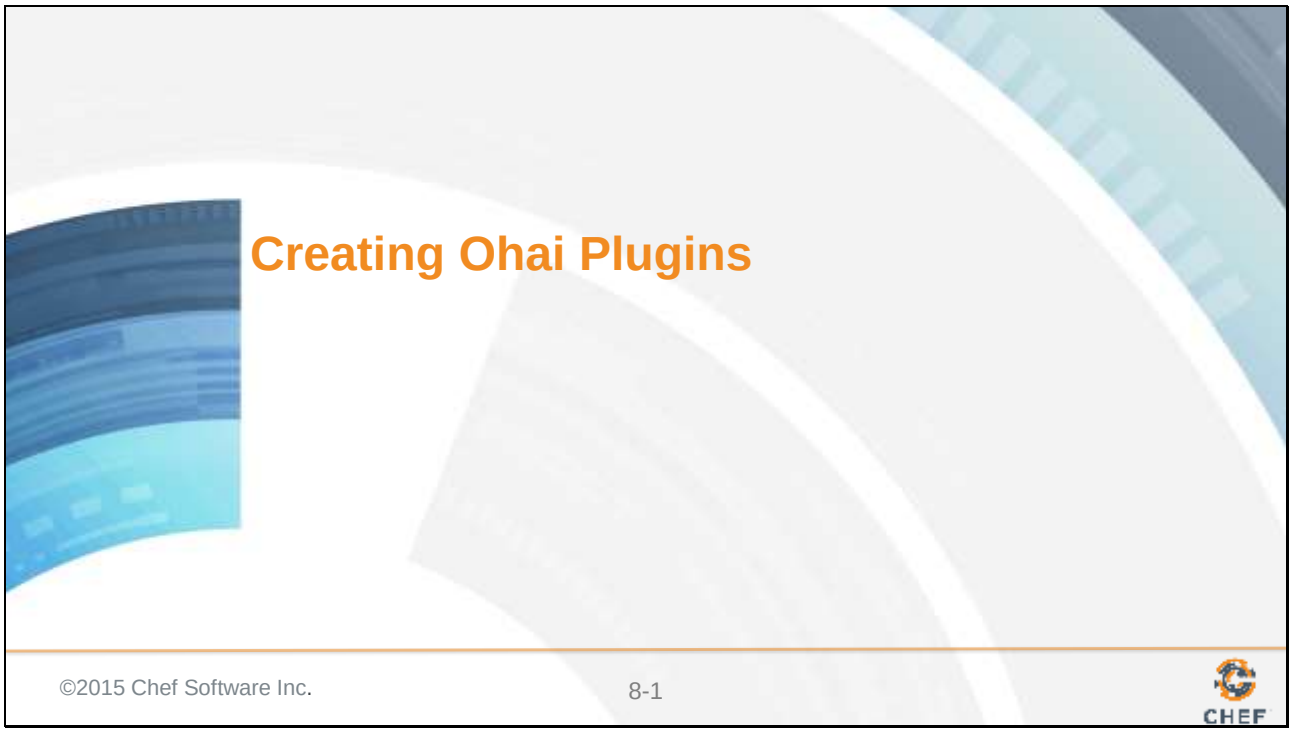
What is the difference between a Ruby Hash and a Mash?

Slide 31

What questions can we answer for you?

Slide 32

8: Creating Ohai Plugins



Slide 2

Objectives

After completing this module, you should be able to:

- Create a tested Ohai plugin

After completing this module you will be able to

Slide 3

EXERCISE

Creating an Ohai Plugin

Being able to learn more about our nodes will make our reporting more powerful and benefit the recipes we write.

Objective:

- ❑ Define expectations for the Ohai plugin
- ❑ Create the Ohai plugin
- ❑ Define expectations for the recipe to deliver the Ohai plugin
- ❑ Create the recipe that delivers the Ohai plugin
- ❑ Define an integration test

CHEF

Slide 4

Installing the chefspec-ohai gem

```
> chef gem install chefspec-ohai
```

```
Successfully installed chefspec-ohai-0.1.0
1 gem installed
```

Adding the Gem to the Spec Helper

~/cookbooks/httpd/spec/spec_helper.rb

```
require 'chefspec'
require 'chefspec/berkshelf'
require 'chefspec/ohai'
```

Slide 6

Creating a Directory for Plugins Tests

```
> mkdir cookbooks/httpd/spec/unit/plugins
```


CHEF

Defining the First Expectation for Plugin

~/cookbooks/httpd/spec/unit/plugins/httpd_modules_spec.rb

```
require 'spec_helper'

describe_ohai_plugin :Apache do
  let(:plugin_file) { 'files/default/httpd_modules.rb' }

  it 'provides apache/modules' do
    expect(plugin).to provides_attribute('apache/modules')
  end
end
```

Executing the Tests to See Failure

```
> chef exec rspec
```

```
F.........

Failures:

 1) Apache provides 'apache/modules'
     Failure/Error: let(:plugin_source) { File.read(plugin_file) }

     Errno::ENOENT:
       No such file or directory @ rb_sysopen -
files/default/httpd_modules.rb
```


CHEF

Slide 9

EXERCISE

Creating an Ohai Plugin

Being able to learn more about our nodes will make our reporting more powerful and benefit the recipes we write.

Objective:

- ✓ Define expectations for the Ohai plugin
- ❑ Create the Ohai plugin
- ❑ Define expectations for the recipe to deliver the Ohai plugin
- ❑ Create the recipe that delivers the Ohai plugin
- ❑ Define an integration test

CHEF

Creating the Plugin as a Cookbook File

```
> chef generate file httpd_modules.rb
```

```
Recipe: code_generator::cookbook_file
  * directory[/Users/franklinwebber/training/source/extending_cookbooks-
repo/cookbooks/httpd/files/default] action create (up to date)
  * template[/Users/franklinwebber/training/source/extending_cookbooks-
repo/cookbooks/httpd/files/default/httpd_modules.rb] action create
    - create new file
/Users/franklinwebber/training/source/extending_cookbooks-
repo/cookbooks/httpd/files/default/httpd_modules.rb
    - update content in file
/Users/franklinwebber/training/source/extending_cookbooks-
repo/cookbooks/httpd/files/default/
```


CHEF

Defining the Plugin that Provides Values

~/cookbooks/httpd/files/default/httpd_modules.rb

```
Ohai.plugin :Apache do
  provides 'apache/modules'
end
```

Executing the Tests to See Success

```
> chef exec rspec
```

```
..........

Finished in 5.85 seconds (files took 2.74 seconds to load)
10 examples, 0 failures
```


CHEF

Defining an Expectation for Attribute Value

~/cookbooks/httpd/spec/unit/plugins/httpd_modules_spec.rb

```
require 'spec_helper'

describe_ohai_plugin :Apache do
  let(:plugin_file) { 'files/default/httpd_modules.rb' }

  it 'provides apache/modules' do
    expect(plugin).to provides_attribute('apache/modules')
  end

  it 'correctly captures output' do
    allow(plugin).to_receive(:shell_out).with('apachectl -t -D
DUMP_MODULES').and_return('OUTPUT')
    expect(plugin_attribute('apache/modules')).to eq("OUTPUT")
  end
end
```


CHEF

Executing the Tests to See Failure

```
> chef exec rspec
```

```
.F.........

Failures:

  1) Apache correctly captures output
     Failure/Error: expect(plugin_attribute('apache/modules')).to
eq("OUTPUT")

     NoMethodError:
       undefined method `[]' for nil:NilClass
```


CHEF

Capturing the Apache Modules Content

~/cookbooks/httpd/files/default/httpd_modules.rb

```
Ohai.plugin :Apache do
  provides 'apache/modules'

  collect_data :default do
    apache Mash.new
    modules_cmd = shell_out('apachectl -t -D DUMP_MODULES')
    apache[:modules] = modules_cmd.stdout
  end
end
```

Executing the Tests to See Success

```
> chef exec rspec
```

```
.[2016-10-05T17:00:43-05:00] WARN: Plugin Definition Error:
<files/default>: collect_data already defined on platform default
..........

Finished in 5.84 seconds (files took 3.01 seconds to load)
11 examples, 0 failures
```


CHEF

Slide 17

EXERCISE

Creating an Ohai Plugin

Being able to learn more about our nodes will make our reporting more powerful and benefit the recipes we write.

Objective:

- ✓ Define expectations for the Ohai plugin
- ✓ Create the Ohai plugin
- ❑ Define expectations for the recipe to deliver the Ohai plugin
- ❑ Create the recipe that delivers the Ohai plugin
- ❑ Define an integration test

CHEF

Adding a Dependency on the Ohai Cookbook

~/cookbooks/httpd/metadata.rb

```
name 'httpd'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'all_rights'
description 'Installs/Configures httpd'
long_description 'Installs/Configures httpd'
version '0.1.0'

# If you upload to Supermarket you should set this so your cookbook
# gets a `View Issues` link
# issues_url 'https://github.com/<insert_org_here>/httpd/issues' if respond_to?(:issues_url)

# If you upload to Supermarket you should set this so your cookbook
# gets a `View Source` link
# source_url 'https://github.com/<insert_org_here>/httpd' if respond_to?(:source_url)
depends 'ohai'
```

Creating the Recipe to Add the Plugin

```
> chef generate recipe ohai_httpd_modules
```

```
Recipe: code_generator::recipe
  *
directory[/Users/franklinwebber/training/source/extending_cookbook
s-repo/cookbooks/httpd/spec/unit/recipes] action create (up to
date)
  *
cookbook_file[/Users/franklinwebber/training/source/extending_cook
books-repo/cookbooks/httpd/spec/spec_helper.rb] action
create_if_missing (up to date)
  *
template[/Users/franklinwebber/training/source/extending_cookbooks
-
```


CHEF

Defining the Expectation Plugin Deployed

~/cookbooks/httpd/spec/unit/recipes/ohai_httpd_modules_spec.rb

```
# ... UPPER PORTION OF THE SPECIFICATION ...
    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

    it 'installs the modules plugin' do
      expect(chef_run).to create_ohai_plugin('httpd_modules')
    end
  end
end
```

Execute the Tests to See Failure

```
> chef exec rspec
```

```
..........F

Failures:

  1) httpd::ohai_httpd_modules When all attributes are default, on
an unspecified platform installs the modules plugin
     Failure/Error: expect(chef_run).to
create_ohai_plugin('httpd_modules')

       expected "ohai_plugin[httpd_modules]" with action :create
to be in Chef run. Other ohai_plugin resources:
```


CHEF

Slide 22

EXERCISE

Creating an Ohai Plugin

Being able to learn more about our nodes will make our reporting more powerful and benefit the recipes we write.

Objective:

- ✓ Define expectations for the Ohai plugin
- ✓ Create the Ohai plugin
- ✓ Define expectations for the recipe to deliver the Ohai plugin
- ❑ Create the recipe that delivers the Ohai plugin
- ❑ Define an integration test

CHEF

Slide 23

Defining the Ohai Plugin in the Recipe

~/cookbooks/httpd/recipes/ohai_httpd_modules.rb

```
#
# Cookbook Name:: httpd
# Recipe:: ohai_httpd_modules
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
ohai_plugin 'httpd_modules'
```

Executing the Tests to See Success

```
> chef exec rspec
```

```
...........

Finished in 5.77 seconds (files took 2.62 seconds to load)
12 examples, 0 failures
```


CHEF

Slide 25

EXERCISE

Creating an Ohai Plugin

Being able to learn more about our nodes will make our reporting more powerful and benefit the recipes we write.

Objective:

- ✓ Define expectations for the Ohai plugin
- ✓ Create the Ohai plugin
- ✓ Define expectations for the recipe to deliver the Ohai plugin
- ✓ Create the recipe that delivers the Ohai plugin
- ❑ Define an integration test

CHEF

Appending the New Recipe to the Suite

~/httpd/.kitchen.yml

```
# ... TOP OF KITCHEN CONFIGURATION ...
suites:
  - name: default
    run_list:
      - recipe[httpd::default]
      - recipe[httpd::ohai_httpd_modules]
    verifier:
      inspec_tests:
        - test/recipes
    attributes:
```

Converging the Updated Default Suite

```
> kitchen converge
```

```
-----> Starting Kitchen (v1.11.1)
-----> Converging <default-centos-67>...
       resolving cookbooks for run list: ["httpd::default",
"httpd::ohai_httpd_modules"]
       Synchronizing Cookbooks:
         - ohai (4.2.2)
         - httpd (0.1.0)
         - compat_resource (12.14.7)
       Installing Cookbook Gems:
       Compiling Cookbooks...
       Recipe: httpd::ohai_httpd_modules
```

Defining an Expectation for the Plugin

~/httpd/test/recipes/ohai_httpd_modules.rb

```
plugin_directory = '/tmp/kitchen/ohai/plugins'

describe command("ohai -d #{plugin_directory} apache") do
  its(:stdout) { should match(/core_module/) }
end
```

Verifying the New Expectation

```
> kitchen verify
```

```
-----> Starting Kitchen (v1.11.1)
-----> Verifying <ohai-plugin-centos-67>...
       Use `/home/chef/httpd/test/recipes/default` for testing

Target:  ssh://kitchen@localhost:32768

  ✔  Command ohai -d /tmp/kitchen/ohai/plugins apache stdout
should match /core_module \(static\)/

Summary: 1 successful, 0 failures, 0 skipped
```

Slide 30

EXERCISE

Creating an Ohai Plugin

Being able to learn more about our nodes will make our reporting more powerful and benefit the recipes we write.

Objective:

- ✓ Define expectations for the Ohai plugin
- ✓ Create the Ohai plugin
- ✓ Define expectations for the recipe to deliver the Ohai plugin
- ✓ Create the recipe that delivers the Ohai plugin
- ✓ Define an integration test

CHEF

Slide 31

DISCUSSION

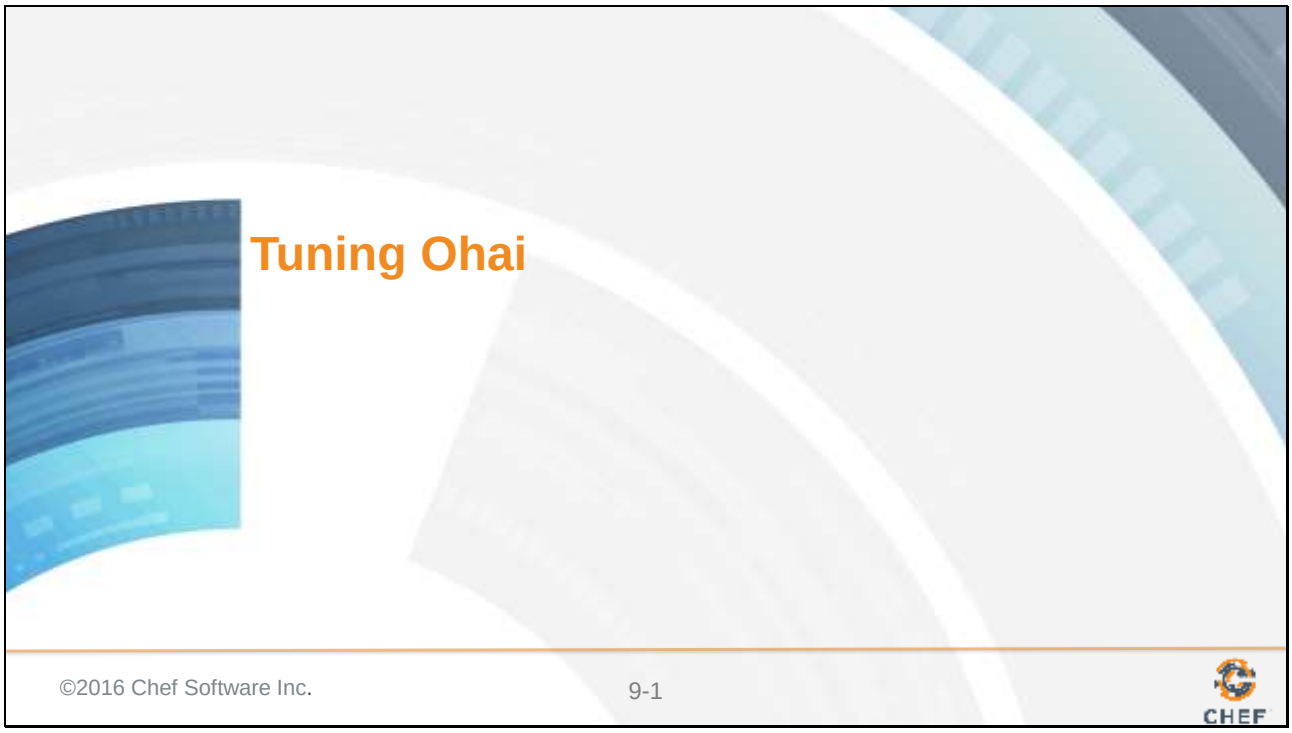
Q&A

What questions can we answer for you?

CHEF

Slide 32

9: Tuning Ohai



Slide 2

Objectives

After completing this module, you should be able to:

- Describe how you configure the node to automatically load ohai plugins
- Describe how to enable ohai hints
- Describe how to remove plugins that you do not want executed
- Describe where you can fine more information about better Ohai performance

After completing this module you will be able to

Slide 3

EXERCISE

Run Ohai Smoother

There are a few more things to learn about Ohai that could help increase performance.

Objective:

- ❑ Configure the node to automatically load ohai plugins
- ❑ Enable ohai hints
- ❑ Remove plugins that you do not want executed
- ❑ Choose only the plugins you want executed

CHEF

Slide 4

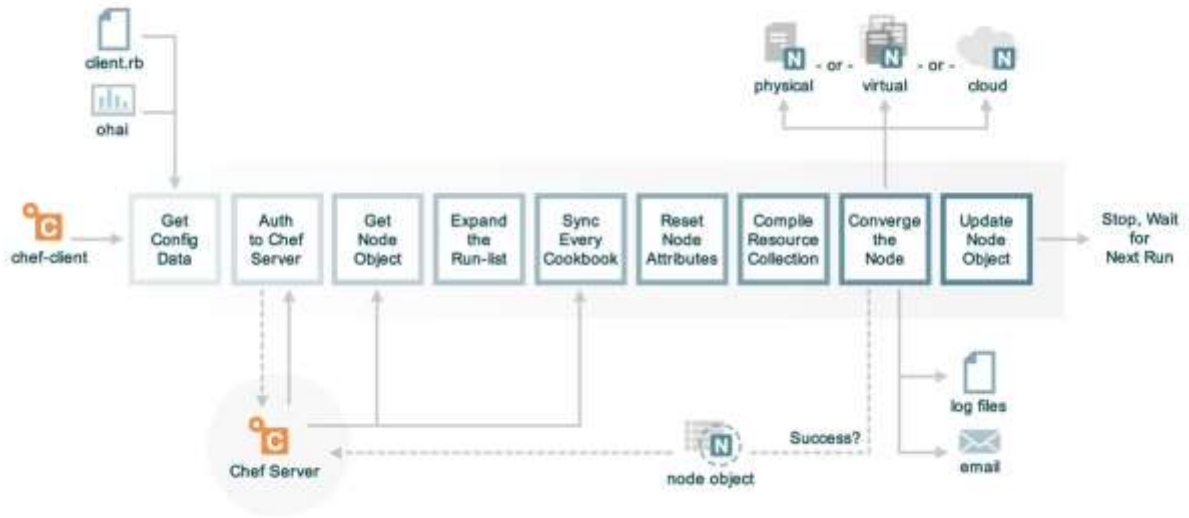
CONCEPT

Node Configuration File

All nodes have a configuration file that contain details about node, overrides, and other data.

Slide 5

The Anatomy of a chef-client Run



CHEF

Slide 6

CONCEPT

chef-client cookbook

The chef-client cookbook contains a number of useful recipes:

- service (default)
  - Run chef-client as a service, converging at a periodic interval
- delete_validation
  - Remove the organization validation key [SECURITY ISSUE]
- config
  - Define node configuration

https://supermarket.chef.io/cookbooks/chef-client

CHEF

Viewing the chef-client config Recipe

~/cookbooks/chef-client/recipes/config.rb

```
# ... OTHER RESOURCES ...
template "#{node["chef_client"]["conf_dir"]}/client.rb" do
  source 'client.rb.erb'
  owner d_owner
  group d_group
  mode 00644
  variables(
    :chef_config => node['chef_client']['config'],
    :chef_requires => chef_requires,
    :ohai_disabled_plugins  => node['ohai']['disabled_plugins'],
    :start_handlers => node['chef_client']['config']['start_handlers'],
    :report_handlers => node['chef_client']['config']['report_handlers'],
    :exception_handlers => node['chef_client']['config']['exception_handlers']
  )
  notifies :create, 'ruby_block[reload_client_config]', :immediately
end
```

Slide 8

~/cookbooks/chef-client/templates/default/client.rb.erb

```
<% if node.attribute?("ohai") && node["ohai"].attribute?("plugin_path") -%>

Ohai::Config[:plugin_path] << "<%= node["ohai"]["plugin_path"] %>"
<% end -%>
<% unless @ohai_disabled_plugins.empty? -%>
Ohai::Config[:disabled_plugins] = [<%= @ohai_disabled_plugins.map { |k| k...

<% end -%>
```

Slide 9

PROBLEM

The Work

- Add the chef-client cookbook to your cookbook collection
- Provide attributes in a wrapper cookbook, role, or environment for:

```
node['ohai']['plugin_path']
```

- Add chef-client cookbook to every node's run list

CHEF

Slide 10

EXERCISE

Run Ohai Smoother

There are a few more things to learn about Ohai that could help increase performance.

Objective:

- ✓ Configure the node to automatically load ohai plugins
- ❑ Enable ohai hints
- ❑ Remove plugins that you do not want executed
- ❑ Choose only the plugins you want executed

CHEF

Slide 11

CONCEPT

Not all Node Plugins are Executed

Some of the plugins will not automatically run against your node. This is particularly true of the cloud provider plugins. As the node does not often know the environment in which is being run.

CONCEPT

Ohai Hints

For those plugins that are not executed you can leave them a hint file that will ensure they are executed.

CHEF

Slide 13

PROBLEM

The Work

- Add the ohai cookbook to your cookbook collection
- Define a recipe that uses the ohai cookbook's ohai_hint resource
- Add this recipe to every node's run list

CHEF

Slide 14

EXERCISE

Run Ohai Smoother

There are a few more things to learn about Ohai that could help increase performance.

Objective:

- ✓ Configure the node to automatically load ohai plugins
- ✓ Enable ohai hints
- ❑ Remove plugins that you do not want executed
- ❑ Choose only the plugins you want executed

CHEF

Slide 15

~/cookbooks/chef-client/templates/default/client.rb.erb

```
<% if node.attribute?("ohai") && node["ohai"].attribute?("plugin_path") -%>

Ohai::Config[:plugin_path] << "<%= node["ohai"]["plugin_path"] %>"
<% end -%>
<% unless @ohai_disabled_plugins.empty? -%>
Ohai::Config[:disabled_plugins] = [<%= @ohai_disabled_plugins.map { |k| k...

<% end -%>
```

Viewing the chef-client config Recipe

~/cookbooks/chef-client/recipes/config.rb

```
# ... OTHER RESOURCES ...
template "#{node["chef_client"]["conf_dir"]}/client.rb" do
  source 'client.rb.erb'
  owner d_owner
  group d_group
  mode 00644
  variables(
    :chef_config => node['chef_client']['config'],
    :chef_requires => chef_requires,
    :ohai_disabled_plugins  => node['ohai']['disabled_plugins'],
    :start_handlers => node['chef_client']['config']['start_handlers'],
    :report_handlers => node['chef_client']['config']['report_handlers'],
    :exception_handlers => node['chef_client']['config']['exception_handlers']
  )
  notifies :create, 'ruby_block[reload_client_config]', :immediately
end
```


CHEF

Slide 17

PROBLEM

The Work

- Add the chef-client cookbook to your cookbook collection
- Select attributes you want to remove
- Find the name of the plugin that provides those attributes
- Provide attributes in a wrapper cookbook, role, or environment for:

```
node['ohai']['disabled_plugins']
```

- Add chef-client cookbook to every node's run list

CHEF

Slide 18

EXERCISE

Run Ohai Smoother

There are a few more things to learn about Ohai that could help increase performance.

Objective:

- ✓ Configure the node to automatically load ohai plugins
- ✓ Enable ohai hints
- ✓ Remove plugins that you do not want executed
- ❑ Choose only the plugins you want executed

CHEF

CONCEPT

Whitelist Node Attributes

When it seems like you are disabling far too many plugins and you want to consider attacking it from the other side:

http://ckbk.it/whitelist-node-attrs

CHEF

Slide 20

EXERCISE

Run Ohai Smoother

There are a few more things to learn about Ohai that could help increase performance.

Objective:

- ✓ Configure the node to automatically load ohai plugins
- ✓ Enable ohai hints
- ✓ Remove plugins that you do not want executed
- ✓ Choose only the plugins you want executed

CHEF

Slide 21

DISCUSSION

Q&A

What questions can we answer for you?

CHEF

Slide 22

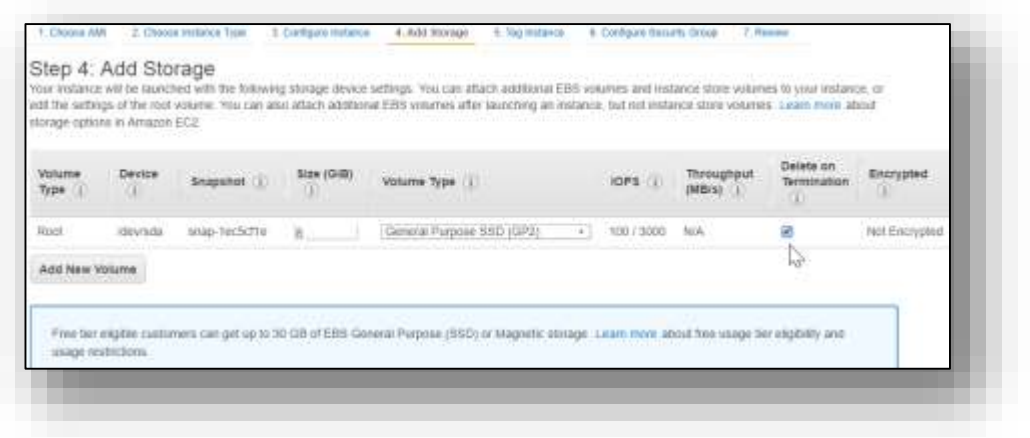
Appendix Z: Course-wide Instructor Notes

1. Training Lab System Setup

1. Open the AWS site from here: https://aws.amazon.com/
   - Login Credentials for Chef instructors: training-aws@chef.io
   - Password: Contact Chef Training Services if you don't know it or how to obtain it. training@chef.io
   - Partner credentials should be provided by Chef directly to partners.
2. Click the first link in column **EC2 *Virtual Servers in the Cloud***
3. From the navigation pane on the left, select **Images/AMIs**. The "Step 1" page displays with a list of available AMIs.
4. Select **Extending Cookbooks – CentOS 6.7 – 1.0.0** from the list of options.
5. Click **Launch**. The "Step 2" page displays.
6. Select the first **Micro Instance** from the list provided and click **Next: Configure Instance Details** at the bottom of the screen. The "Step 3" page displays.
7. Enter the **Number of Instances**.

   Note: You will need 1 instance for each student enrolled in the class - and three for yourself.

8. Click **Next: Add Storage** at the bottom of the page. The "Step 4" page displays. Ensure that '**Delete on Termination**' is selected.



9. Click **Next: Tag Instance** at the bottom of the page. The "Step 5" page displays.
10. Enter a **Value**.

    Note: A recommended naming convention for the instances: [TRAINER'S INITIALS] - [CLASS NAME] - [CLASS DATE]

11. Click **Next: Configure Security Group**. The "Step 6" page displays.
12. Click the **Select an existing security group** radio button. A list of security groups displays.

13. Select **all-open**.
14. Click **Review and Launch** at the bottom of the screen. The "Step 7" page displays.
15. After you review the instances, click **Launch**. The "Select a key pair" window displays.
16. Confirm that this is set to **Proceed without a key pair** and click the acknowledgement check box.
17. Click **Launch Instances**. The "Launch Status" page displays.
18. Click **View Instances**. The instances list displays.
19. From here, copy all of the instances and create a gist file to share with the class.
20. Use goo.gl to shorten the URL to the gist file.

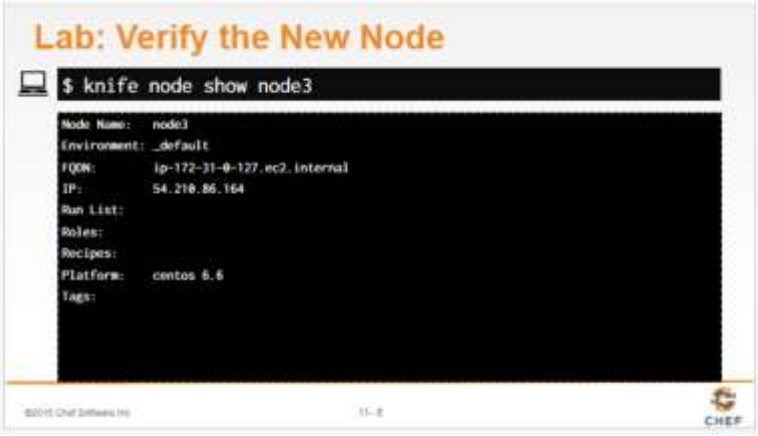
Note: The login credentials and password for the AMIs used in class are chef/chef. You'll need to tell the students that at the appropriate time.

2. How to Use Lab Slides

Regarding the "Lab" exercises (not the Group Exercises), you should encourage students to use the high-level hammer/wrench "Lab" slide steps first, and then resort to the subsequent detailed step slides if the students need the details to complete the lab. You can still use the subsequent detailed step slides as a vehicle to review each lab. For example:

This is a high-level hammer/wrench "Lab" instruction slide. Encourage students to complete the lab using this high level hammer/wrench "Lab" slide first.

If some students can't complete the lab based on the above slide, they are free to follow the subsequent detailed step slides, such as these:



You can also use the above detailed slides as a vehicle for reviewing the labs.