CHEF™

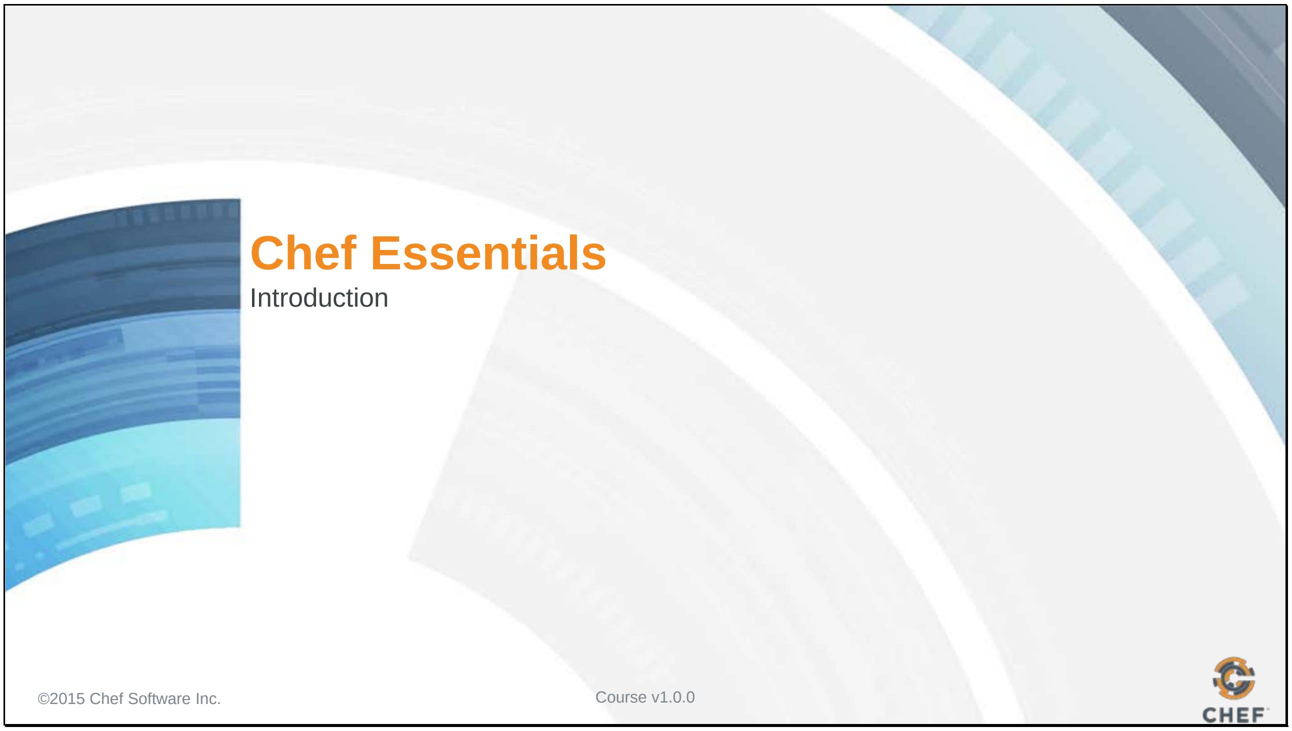
Chef Training Services

Chef Essentials

Participant Guide

Course Rev 1.0.0

1: Introduction



This Chef Essentials course provides a basic understanding of Chef's core components, basic architecture, commonly used tools, and basic troubleshooting methods.

This should provide you with enough knowledge to start using Chef to automate common infrastructure tasks and express solutions to common infrastructure problems.

Slide 2

Introduce Yourselves

Name

Current job role

Previous job roles/background

Experience with Chef and/or config management

Favorite Text Editor

Slide 3

Expectations

You will leave this class with a basic understanding of Chef's core components, architecture, commonly used tools, and basic troubleshooting methods

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.

 1- 3 CHEF

Chef is not, in itself, a solution to your infrastructure problems. Chef is an automation framework. You bring the domain expertise about your own business and its problems. Chef provides a platform for modeling solutions to those problems. Our job in this class is to work together to teach you how to express solutions to your unique problems with Chef.

Together we get unicorns and rainbows, but we can't have one without the other.

Slide 4

Course Objectives

After completing this course, you should be able to:

- Use Chef Resources to define the state of your system
- Write and use Chef recipes and cookbooks
- Automate testing of cookbooks
- Manage multiple nodes with Chef Server
- Create Organizations
- Bootstrap nodes
- Assign Roles to nodes
- Deploy nodes to environments

Slide 5

Agenda

Day 1

Getting a Workstation
Using Resources
Building Cookbooks
Testing with Test Kitchen
Details About a System
Desired State and Data
Local Workstation Installation

Day 2

Connecting to Chef Server
Community Cookbooks
Managing Multiple Nodes
Roles
Search
Environments

Slide 6

Chef

Chef can automate how you build, deploy, and manage your infrastructure.

Chef can integrate with cloud-based platforms such as Rackspace and Amazon Elastic Compute Cloud to automatically provision and configure new machines.

Chef can automate how you build, deploy, and manage your infrastructure. Your infrastructure becomes as versionable, testable, and repeatable as application code enabling you to automate the process of configuring, deploying and scaling servers and applications

Slide 7

Chef

Chef is a large set of tools that are able to be used on multiple platforms and in numerous configurations.

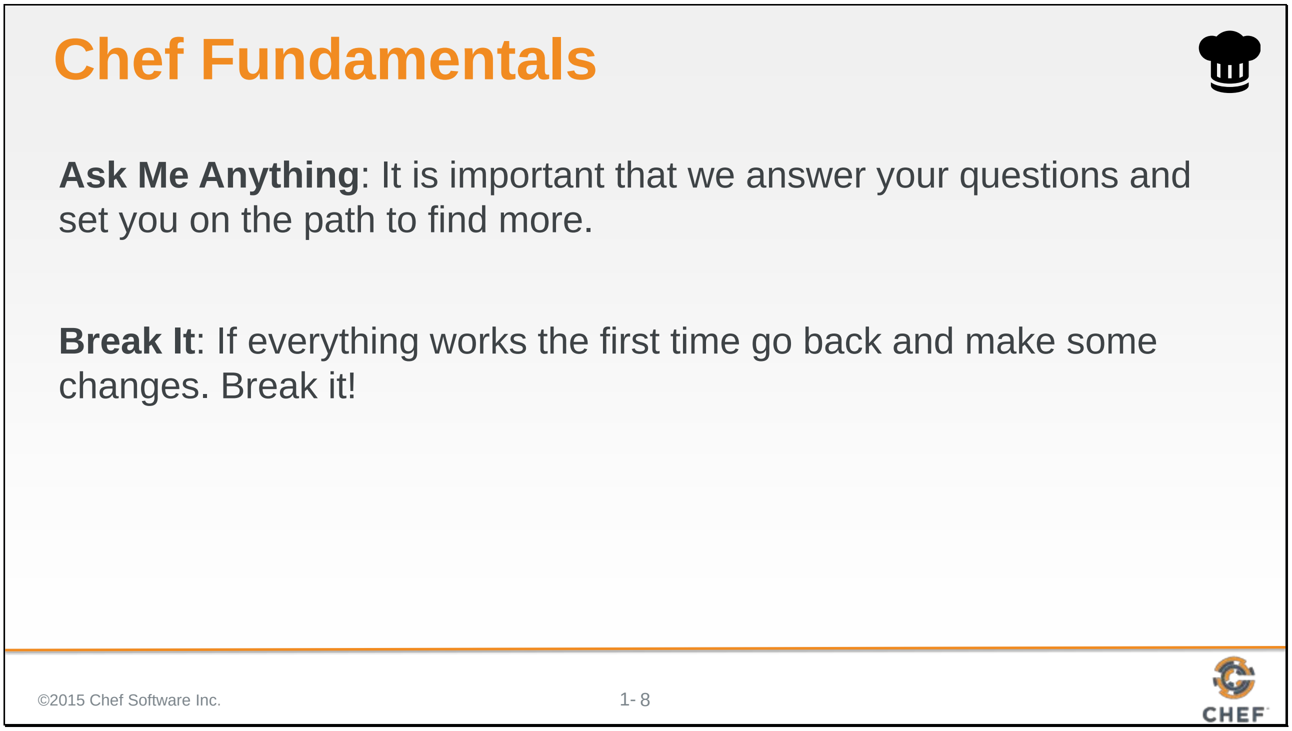
Learning Chef is like learning a language. You will reach fluency very fast but it will take practice until you become comfortable.

A great way to learn Chef is to use Chef

Chef is a large set of tools that are able to be used on multiple platforms and in numerous configurations. We will have time to only explore some of its most fundamental pieces.

Learning Chef is like learning a language. You will reach fluency very fast but it will take practice until you become comfortable.

Slide 8



Ask Me Anything: All of us are coming here with *unique* experiences and from *unique* teams that are using Chef in *unique* ways. It is important that we answer your questions and set you on the path to find more.

Break It: If everything works the first time go back and make some changes. Break it! It's rare that you have a safe space like this to explore. Sometimes its more important to know what something looks like when it does not work than when it does work.

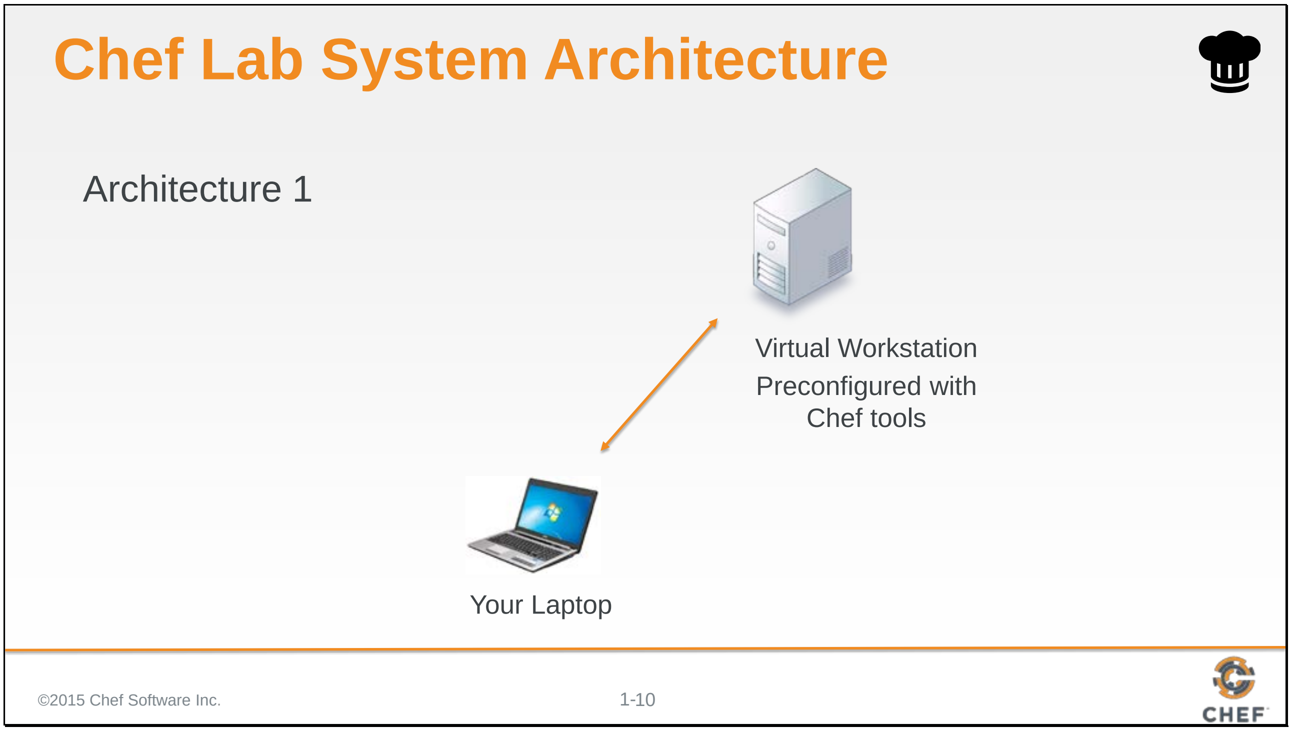
Slide 9

Chef Lab System Architecture

In this course you will use two different architectures:

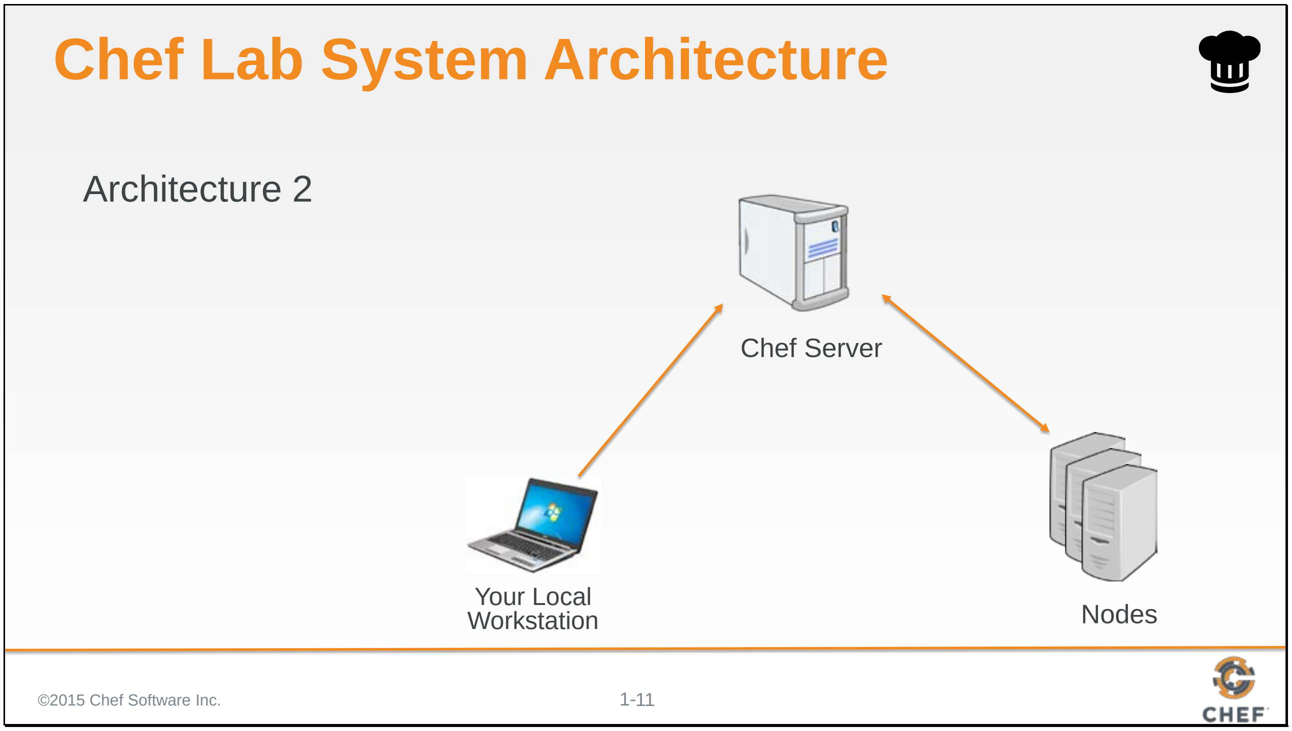
1. Initially, you'll use a virtual workstation so you can start using Chef right away.
2. Later, you'll use a common production type of architecture that includes a Chef Server.

Slide 10



This is the architecture you'll start using in a few minutes. To ensure the smoothest setup experience, you'll be using a virtual workstation with all the necessary tools installed so you can start using Chef right away.

Slide 11

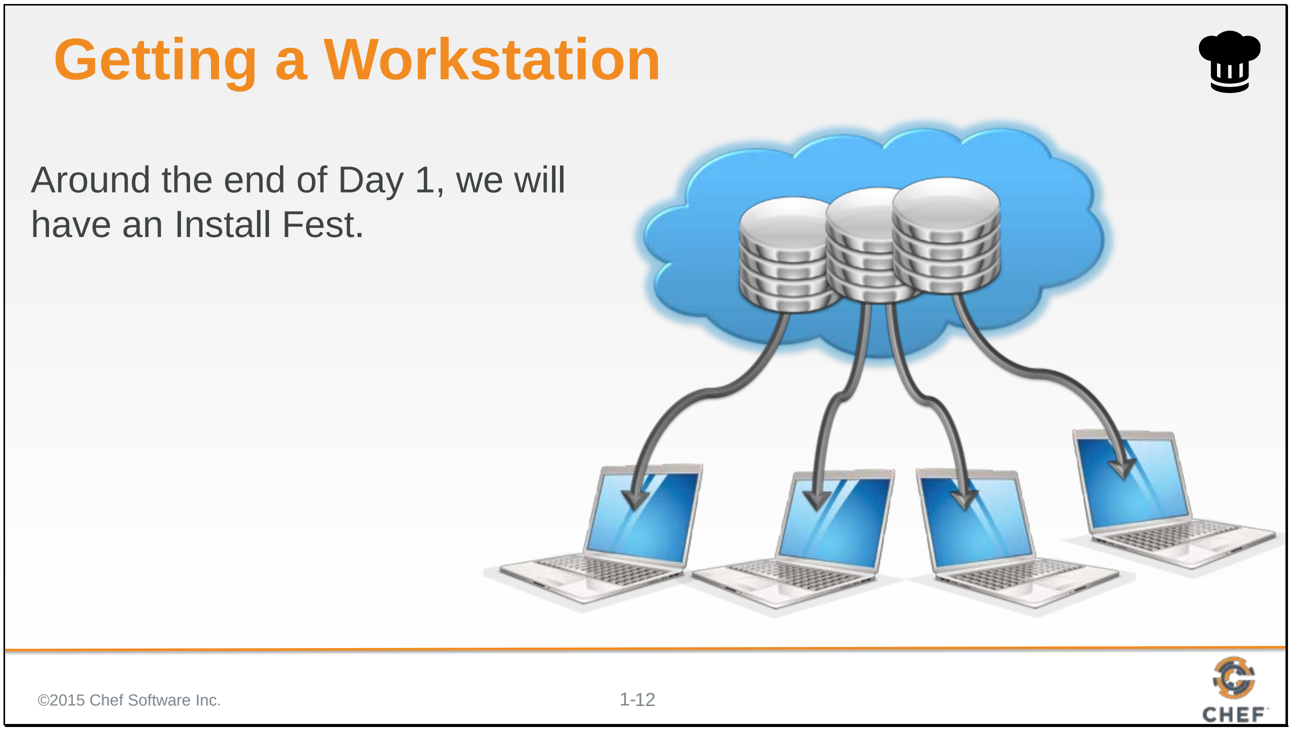


This is the architecture you'll be using later in this course. When using this architecture, the Chef tools will be installed on your laptop and you'll perform your configurations locally before pushing them to the Chef server and ultimately to the nodes you will be managing.

In this way, when you complete this course you will have a code repository on your laptop that can be used and modified to solve real business problems.

We'll discuss the items in this architecture in more detail later in this class.

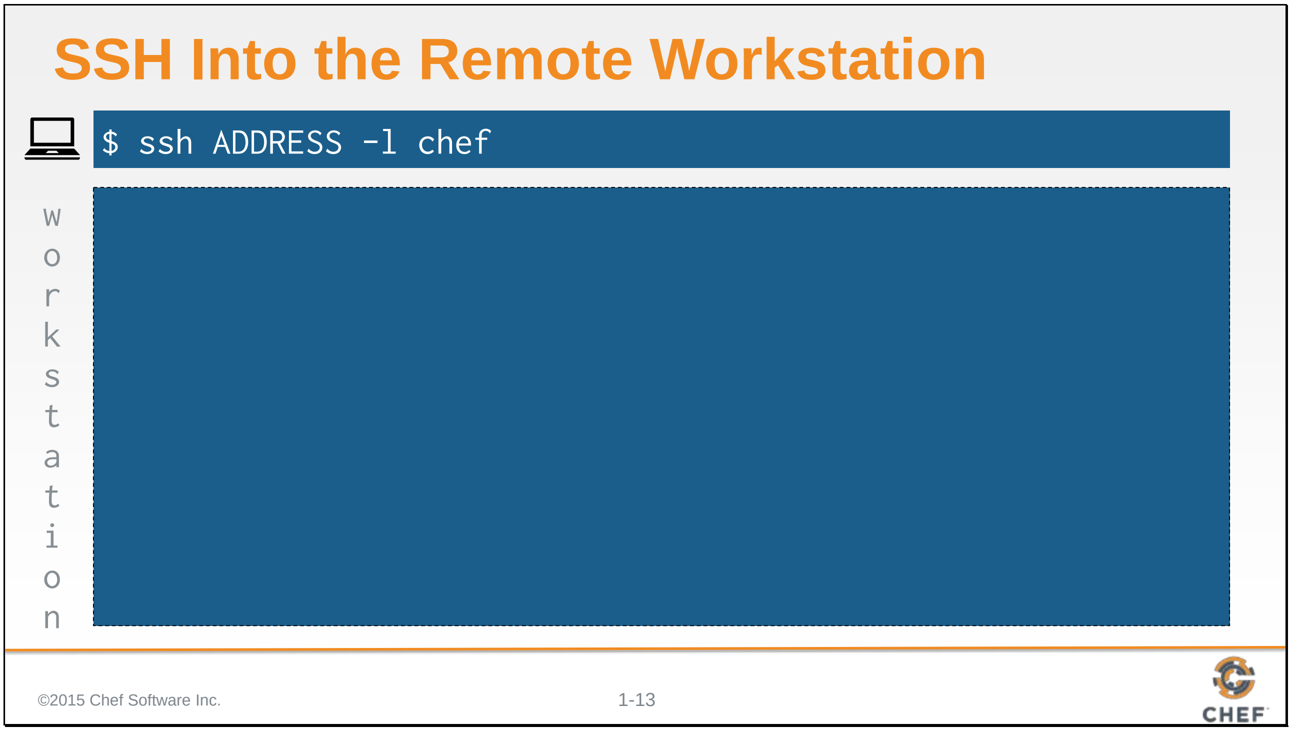
Slide 12



Around the end of Day 1, we will have an Install Fest.

During that time we will install all the necessary tools on your workstation (your laptop) and troubleshoot any installation issues you may experience.

Slide 13



You should use an ssh client like PuTTY to connect to the remote workstation that we assign to you. You'll need to ssh into your assigned workstation in order to issue Chef commands.

You can also use the ssh client to configure Chef recipes.

Slide 14

Getting a Workstation

The chef user has been granted password-less sudoers access

The following software is installed on the remote workstation:

- Chef DK
- Docker
- kitchen-docker gem

Slide 15

Hands-on Legend

- GE or Group Exercise: All participants and the instructor do this task together with the instructor often leading the way and explaining things as we proceed.
- Lab: You perform this task on your own.

In this course, various slides and pages will be tagged with either Group Exercise (or GE), or Lab. This slide defines those tags.

Slide 16

CHEF™

2: Chef Resources

Slide 2

Objectives

After completing this module, you should be able to:

- Use Chef to install packages on your virtual workstation
- Use the chef-apply command
- Create a basic Chef recipe file
- Define Chef Resources

In this module you will learn how to install packages on a virtual workstation, use the 'chef-apply' command, create a basic Chef recipe file and define Chef Resources.

Slide 3

Choose an Editor

You'll need to choose an editor to edit files:

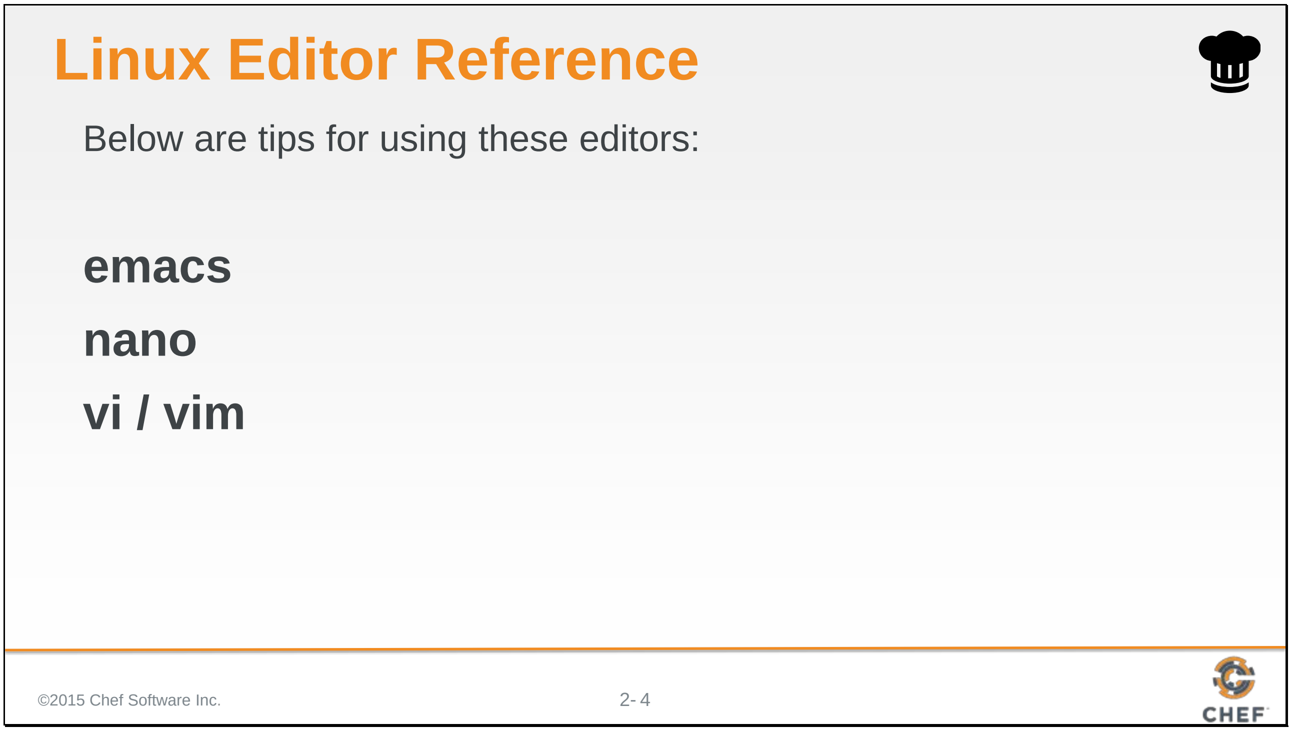
emacs

nano

vi / vim

During this course we are going to need our workstations to have an editor installed. There are at least three command-line editors that we can choose from on the Linux workstation: Emacs, Nano, or Vim.

Slide 4



Emacs: (Emacs is fairly straightforward for editing files.)

OPEN FILE $ emacs FILENAME
WRITE FILE ctrl+x, ctrl+w
EXIT ctrl+x, ctrl+c

Nano: (Nano is usually touted as the easiest editor to get started with editing through the command-line.)

OPEN FILE $ nano FILENAME
WRITE (When exiting) ctrl+x, y, ENTER
EXIT ctrl+x

VIM: (Vim, like vi, is more complex because of its different modes.)

OPEN FILE $ vim FILENAME
START EDITING i
WRITE FILE ESC, :w
EXIT ESC, :q
EXIT (don't write) ESC, :q!

Slide 5

GE: How About Nano?

```
$ which nano
```

```
/usr/bin/which: no nano in
(/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/chef/bin)
```

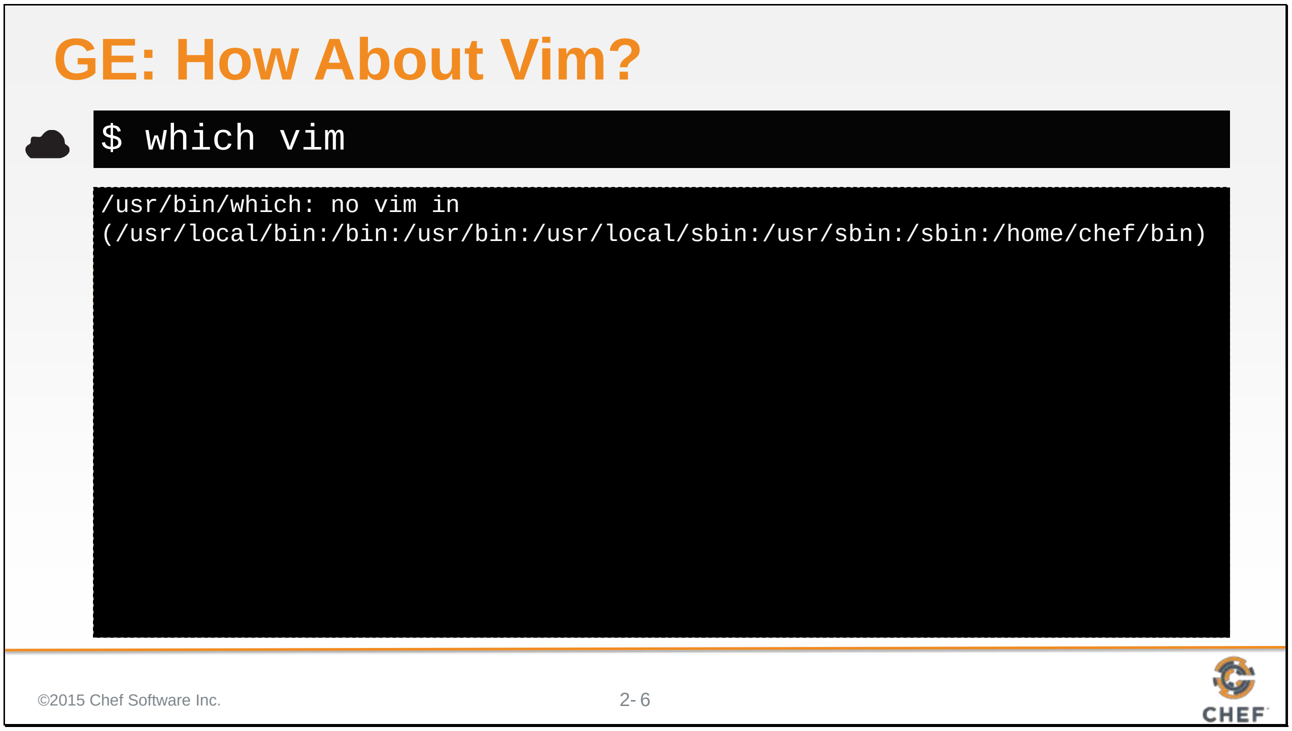

CHEF

Now that you've picked your editor, you need to find out if it is already installed.

Use the `which` command to ask the Operating System (OS) if it knows if there is an executable for our text editor in our path.

Is nano installed? No, it doesn't look like it.

Slide 6



Is vim installed? No, it doesn't look like it either.

Slide 7

GE: How About Emacs?

```
$ which emacs
```

```
/usr/bin/which: no emacs in
(/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/chef/bin)
```


Is emacs installed? Seems like it isn't either.

It seems your workstation doesn't have any of the preferred command-line editors installed. So that means there is a little more configuration left for you to do.

Slide 8

Learning Chef

One of the best ways to learn a technology is to apply the technology in every situation that it can be applied.

A number of chef tools are installed on the system so lets put them to use.

But before you figure out the Linux distribution and start installing packages through the distribution's specific package manager, this seems like a perfect opportunity to experiment with how to solve configuration problems with Chef.

One of the best ways to learn a technology is to apply the technology in every situation that it can be applied. A number of chef tools are installed on the system so lets put them to use.

Slide 9

What is chef-apply?

chef-apply is a command-line application that allows us to work with resources and recipes files.

The first tool we will explore is `chef-apply`. It is a command-line application that allows us to work with resources and recipes files.

Slide 10

What Can chef-apply Do?

```
$ sudo chef-apply --help
```

```
Usage: chef-apply [RECIPE_FILE] [-e RECIPE_TEXT] [-s]
        --[no-]color                 Use colored output, defaults to enabled
    -e, --execute RECIPE_TEXT        Execute resources supplied in a string
    -j JSON_ATTRIBS,                 Load attributes from a JSON file or URL
        --json-attributes
    -l, --log_level LEVEL            Set the log level (debug, info, warn, error,
fatal)
        --minimal-ohai               Only run the bare minimum ohai plugins chef
need ...
    -s, --stdin                      Execute resources read from STDIN
    -v, --version                    Show chef version
    -W, --why-run                    Enable whyrun mode
    -h, --help                       Show this message
```


Run the chef-apply application on the workstation with the "--help" flag to learn more about it.

Reading the output you may be left with more questions. Like what is recipe file? What is recipe text? What are resources?

Let us start answering those questions by looking at Chef's documentation.

Slide 11

First, let's look at Chef's documentation about resources. Visit the docs page on resources and read the first three paragraphs.

Afterwards, let us look at a few examples of resources.

Slide 12

Example: Package

```
package "httpd"
```

The package named "httpd" is installed.

https://docs.chef.io/resource_package.html

Here is an example of the package resource. The package named 'httpd' is installed.

Slide 13

Example: Service

```
service "ntp" do
  action [ :enable, :start ]
end
```

The service named "ntp" is enabled (start on reboot) and started.

https://docs.chef.io/resource_service.html

In this example, the service named 'ntp' is enabled and started.

Slide 14

Example: File

```
file "/etc/motd" do
  content "This company is the property ..."
end
```

The file name "/etc/motd" is created with content "This company is the property ..."

https://docs.chef.io/resource_file.html

In this example, the file named `'/etc/motd'` is created with content "This company is the property...".

Slide 15

Example: File

```
file "/etc/php.ini.default" do
  action :delete
end
```

The file name "/etc/php.ini.default" is deleted.

https://docs.chef.io/resource_file.html

In this example, the file named `'/etc/php.ini.default'` is deleted.

Slide 16

Using the –e Execute Option

```
$ sudo chef-apply --help
```

```
Usage: chef-apply [RECIPE_FILE] [-e RECIPE_TEXT] [-s]
        --[no-]color                 Use colored output, defaults to enabled
    -e, --execute RECIPE_TEXT        Execute resources supplied in a string
    -j JSON_ATTRIBS,                 Load attributes from a JSON file or URL
        --json-attributes
    -l, --log_level LEVEL            Set the log level (debug, info, warn, error,
fatal)
        --minimal-ohai               Only run the bare minimum ohai plugins chef
need ...
    -s, --stdin                      Execute resources read from STDIN
    -v, --version                    Show chef version
    -W, --why-run                    Enable whyrun mode
    -h, --help                       Show this message
```


Let's return to the `chef-apply` command. It looks like you can supply a resource or resources, in a string or text, with the -e flag.

Editors are software and software is delivered to our system through packages. So it seems like you could use the package resource to install our preferred editor.

Slide 17

Group Exercise: Install nano, emacs or vim

```
$ sudo chef-apply -e "package 'nano'"
```

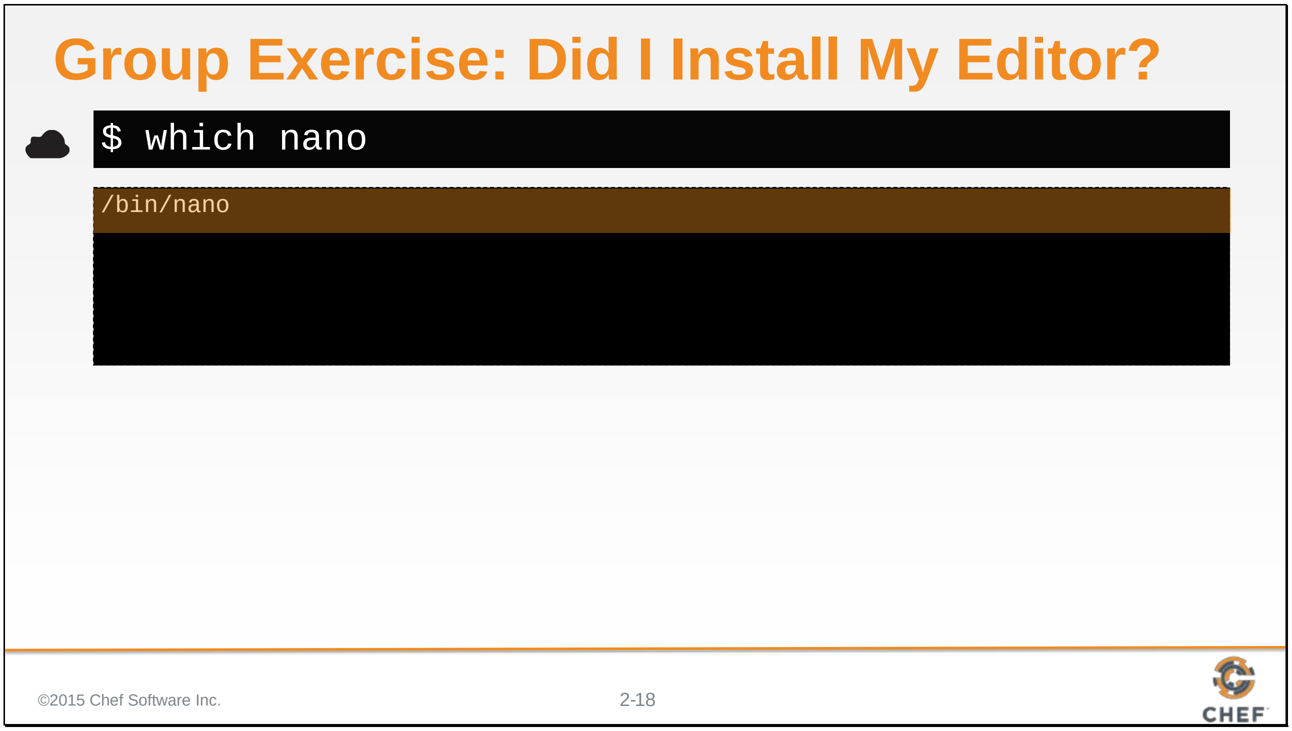
```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * yum_package[nano] action install
    - install version 2.0.9-7.el6 of package nano
```


CHEF

Install the editor package of your choice. In this example we are choosing to install the nano package which installs the nano editor.

You are invited to change the value here to install the editor of your choice.

Slide 18



Verify that the editor is installed by again using the `which` command followed by either nano, emacs or vim.

The 'which' command reports where it was able to find the executable.

Slide 19

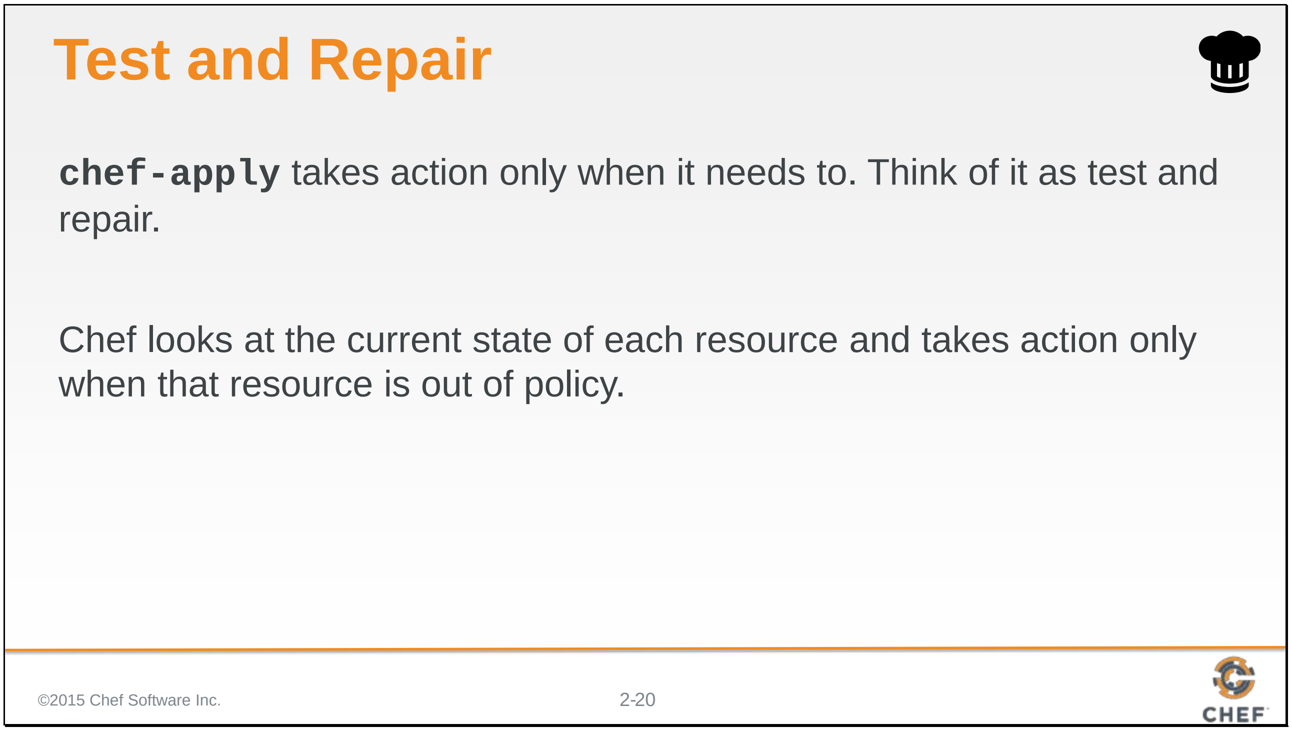
Group Exercise: Test and Repair

1. What would happen if you ran the installation command again?
2. What would happen if the package were to become uninstalled?

What would happen if you ran the installation command again? Before you execute the command think about what will happen. Think about what you would want to happen. Look at the output from the previous execution. Then take a guess. Write it down or type out what you think will happen. Then execute the command again.

What would happen if the package were to become uninstalled? What would the output be if you ran installation command again? Was there a situation where the package was already uninstalled and we executed this resource text?

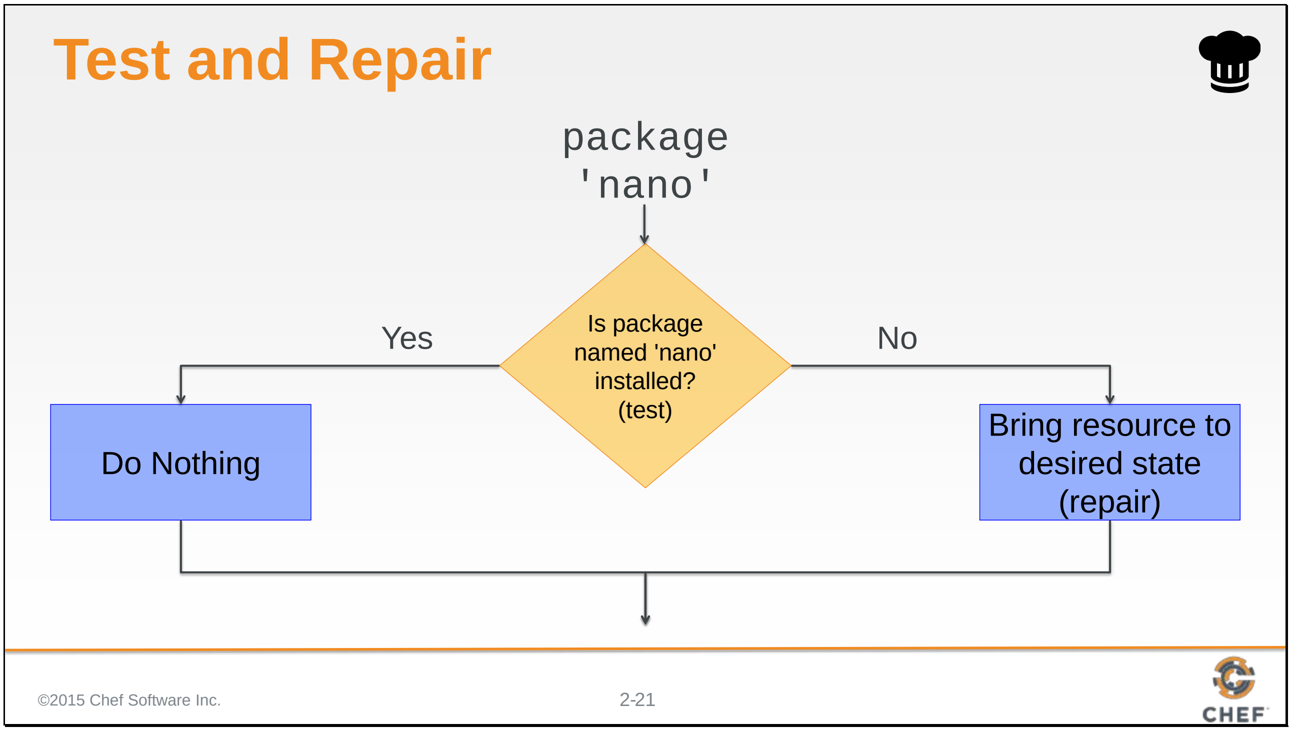
Slide 20



Hopefully it is clear from running the `chef-apply` command a few times that the resource we defined only takes action when it needs to take action.

We call this test and repair. Test and repair means the resource first tested the system before it takes action.

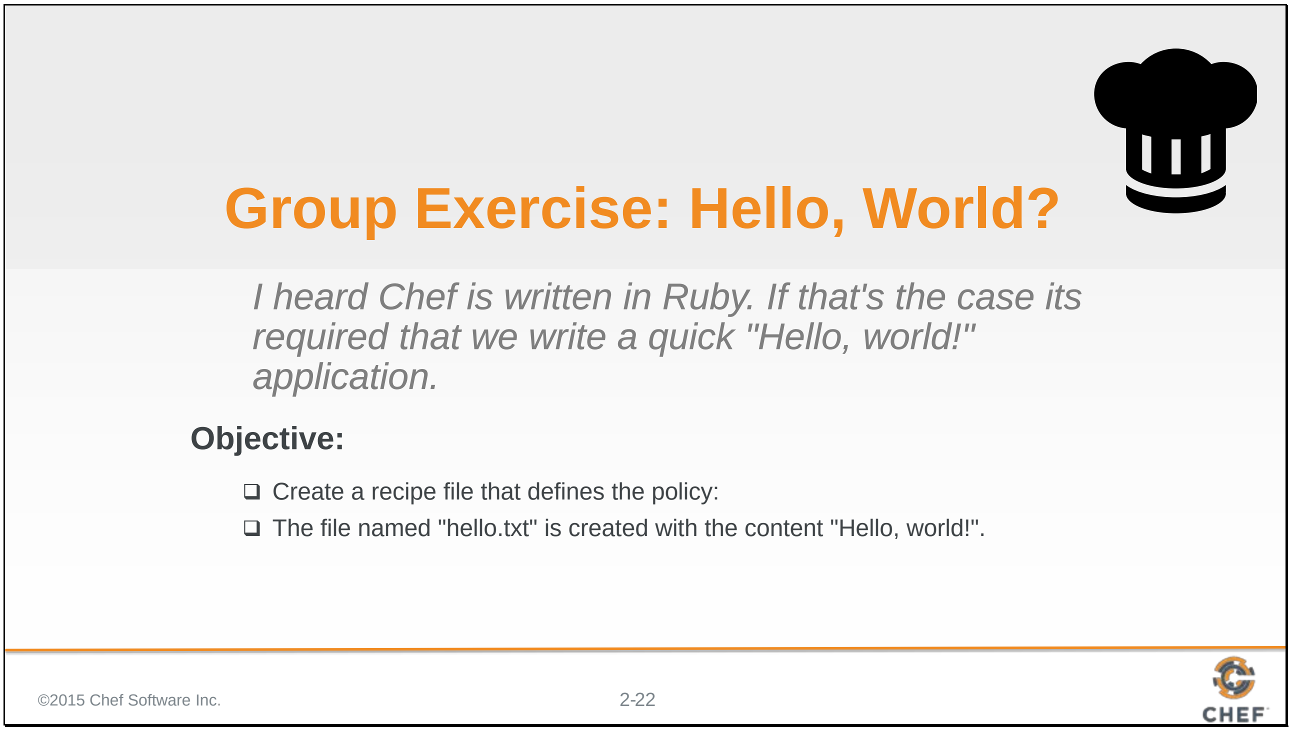
Slide 21



If the package is already installed, then the resource does not need to take action.

If the package is not installed, then the resource NEEDS to take action to install that package.

Slide 22

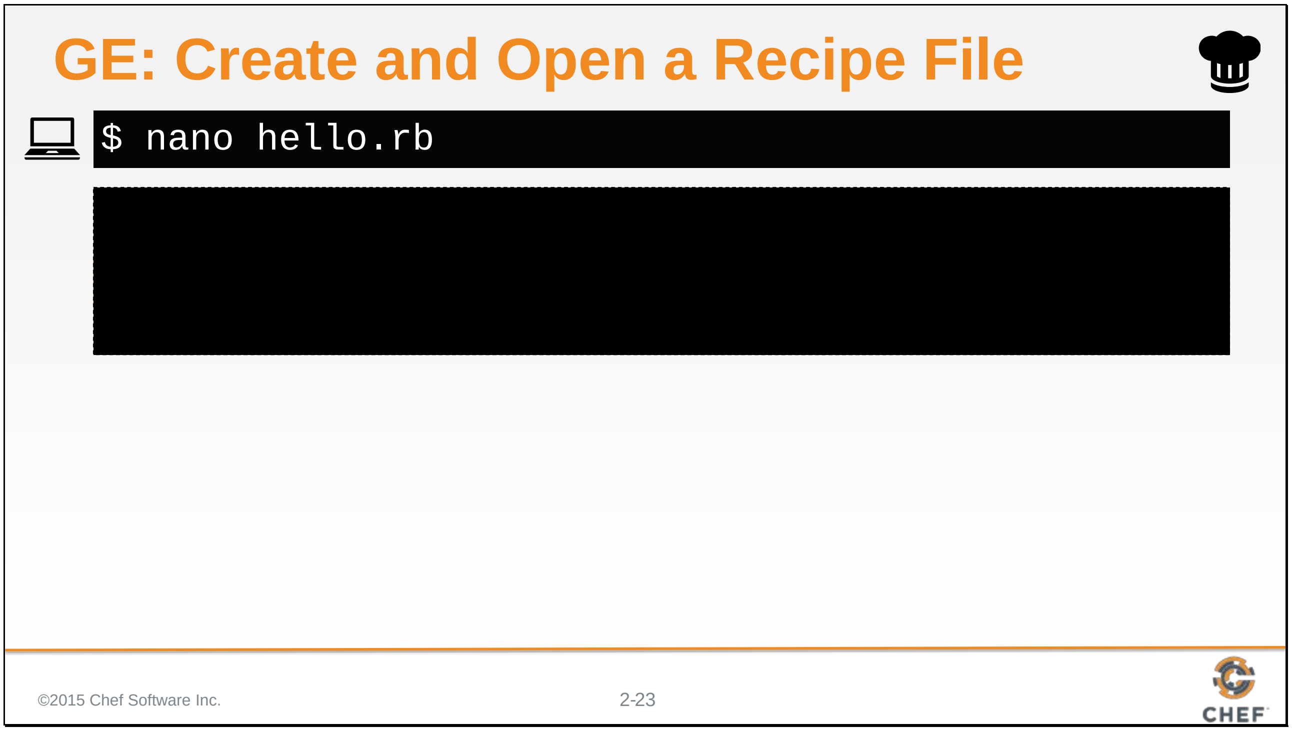


Great! You installed an editor using `chef-apply` but we missed a very important step.

Chef is written in Ruby. Ruby is a programming language and it is required that the first program you write in a programming language is 'Hello World'.

So let's walk through creating a recipe file that creates a file named 'hello.txt' with the contents 'Hello world!'.

Slide 23



Using your editor open the file named 'hello.rb'. 'hello.rb' is a recipe file. It has the extension '.rb' because it is a ruby file.

Slide 24

GE: Create a Recipe File Named hello.rb

~/hello.rb

```
file "hello.txt" do
  content "Hello, world!"
end
```

The file named "hello.txt" is created with the content "Hello, world!"

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

- Add the resource definition displayed above. We are defining a resource with the type called 'file' and named 'hello.txt'. We also are stating what the contents of that file should contain 'Hello, World!'.
- Save the file and return to the terminal and the `chef-apply` command.

Slide 25

GE: Can chef-apply Run a Recipe File?

```
$ sudo chef-apply --help
```

```
Usage: chef-apply [RECIPE_FILE] [-e RECIPE_TEXT] [-s]
        --[no-]color                 Use colored output, defaults to enabled
    -e, --execute RECIPE_TEXT        Execute resources supplied in a string
    -j JSON_ATTRIBS,                 Load attributes from a JSON file or URL
        --json-attributes
    -l, --log_level LEVEL            Set the log level (debug, info, warn, error,
fatal)
        --minimal-ohai               Only run the bare minimum ohai plugins chef
need ...
    -s, --stdin                      Execute resources read from STDIN
    -v, --version                    Show chef version
    -W, --why-run                    Enable whyrun mode
    -h, --help                       Show this message
```


If you were to use '--help' flag again, it looks like you can provide a recipe file directly to the `chef-apply` command.

Slide 26

GE: Apply a Recipe File

```
$ sudo chef-apply hello.rb
```

```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * file[hello.txt] action create
    - create new file hello.txt
    - update content in file hello.txt from none to 315f5b
    --- hello.txt       2015-09-14 22:38:29.386137524 +0000
    +++ ./.hello.txt20150914-1284-1w934it       2015-09-14 22:38:29.386137524
+0000
    @@ -1 +1,2 @@
    +Hello, world!
```


Type the specified command to apply the recipe file. You should see that a file named 'hello.txt' was created and the contents updated to include your 'Hello, World!' text.

Slide 27

Lets look at the contents of the 'hello.txt' file to prove that it was created and the contents of file is what we wrote in the recipe. The result of the command should show you the contents 'Hello, world!'.

Slide 28

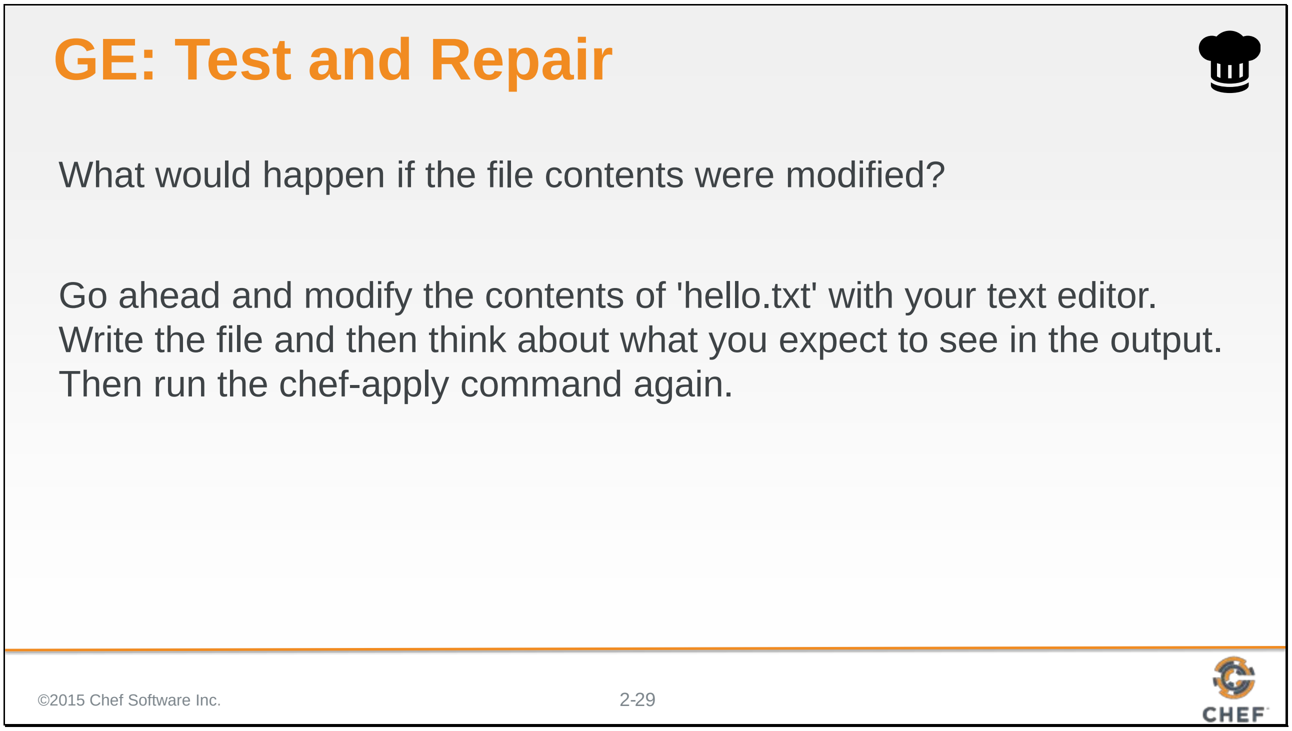
GE: Test and Repair

What would happen if you ran the command again?

What happens when I run the command again?

Again, before you run the command -- think about it. What are your expectations now from the last time you ran it? What will the output look like?

Slide 29



- Modify the contents of 'hello.txt'. Save the file with the new contents.
- Then think about what will happen if you applied this recipe file again.
- Then use `chef-apply` to apply the recipe file again.

Slide 30

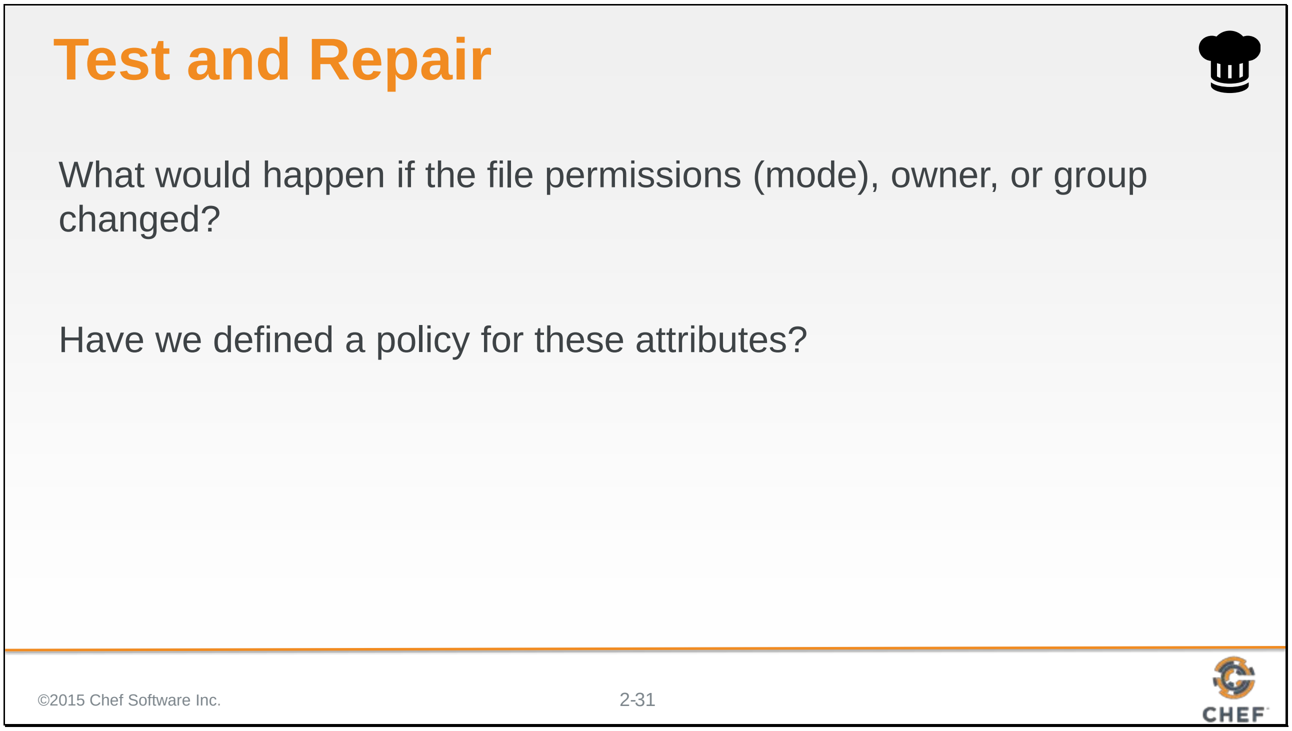
Test and Repair

What would happen if the file were removed?

And, of course, what would happen if the file was removed?

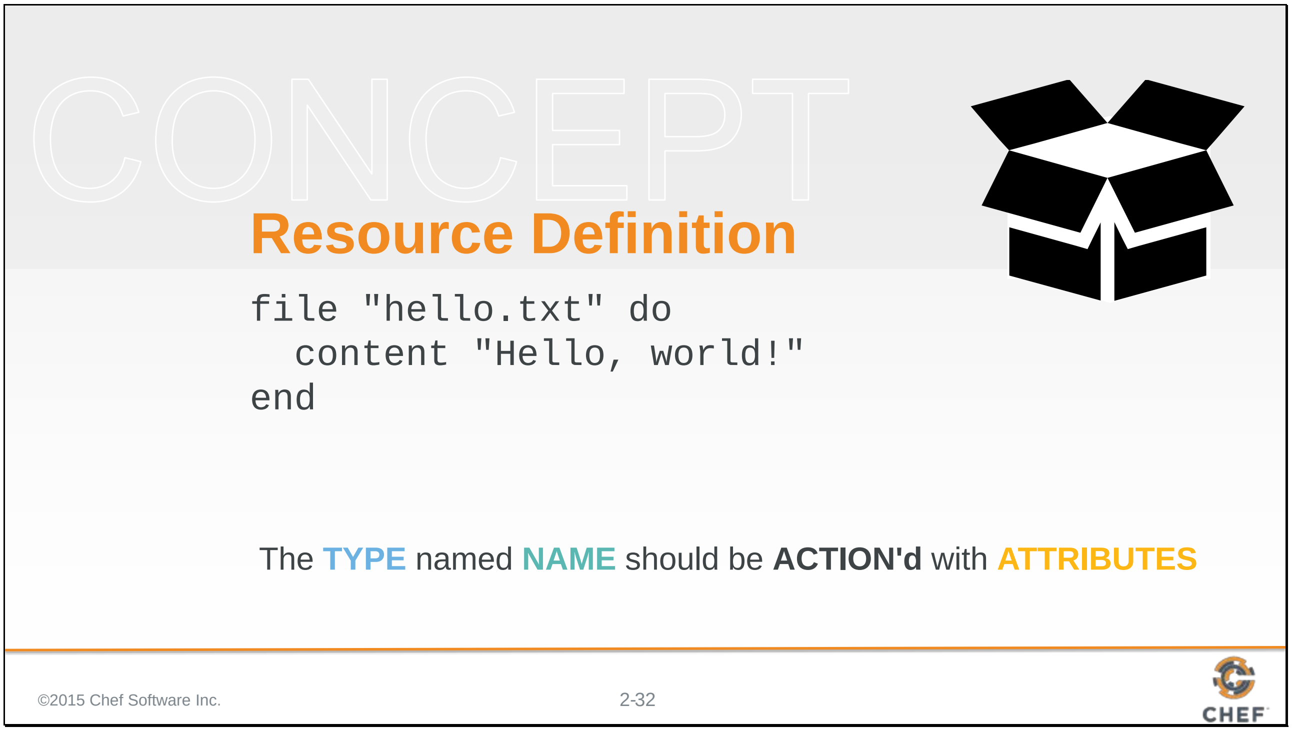
At this point you hopefully you are starting to understand the concept of test and repair.

Slide 31



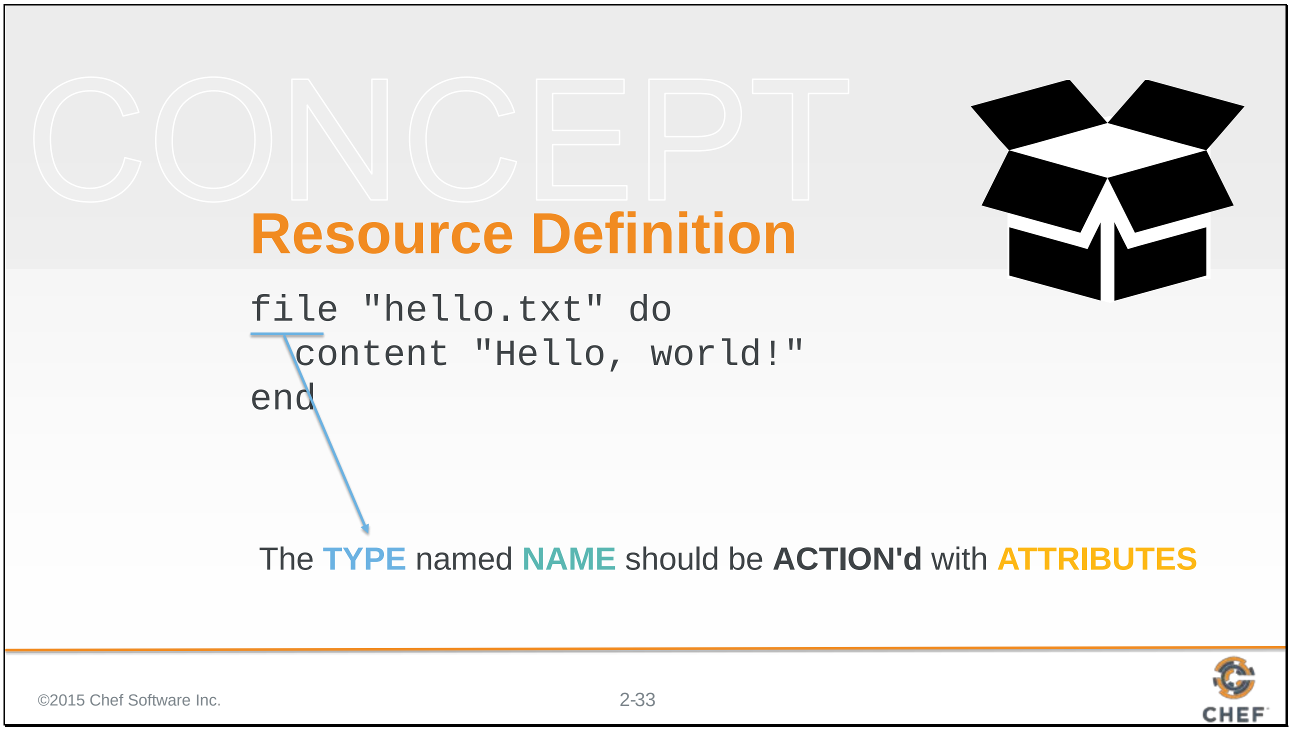
What would happen if the file permissions, owner or group of the file changed? In the resource that we defined have we specified the values that we desired in our policy.

Slide 32



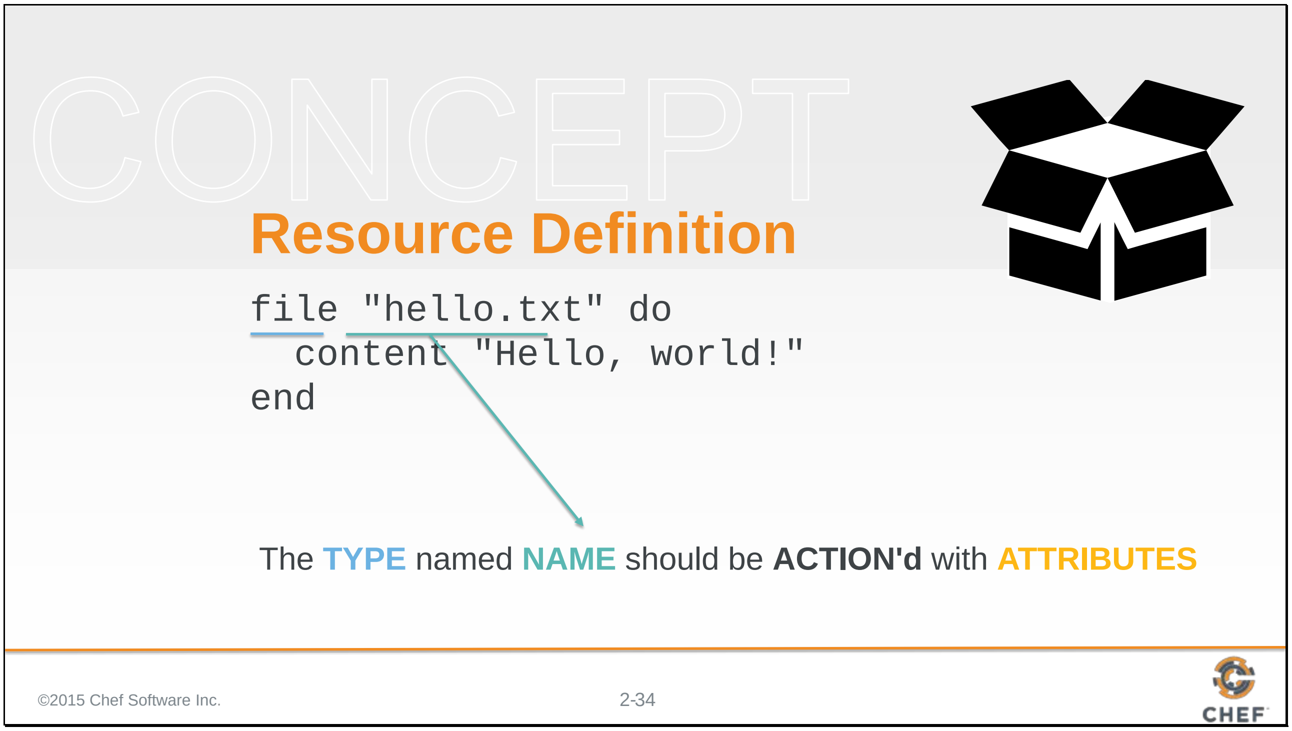
Let's take a moment and talk about the structure of a resource definition. We'll break down the resource that we defined in our recipe file.

Slide 33



The first element of the resource definition is the resource type. In this instance the type is 'file'. Earlier we used 'package'. We showed you an example of 'service'.

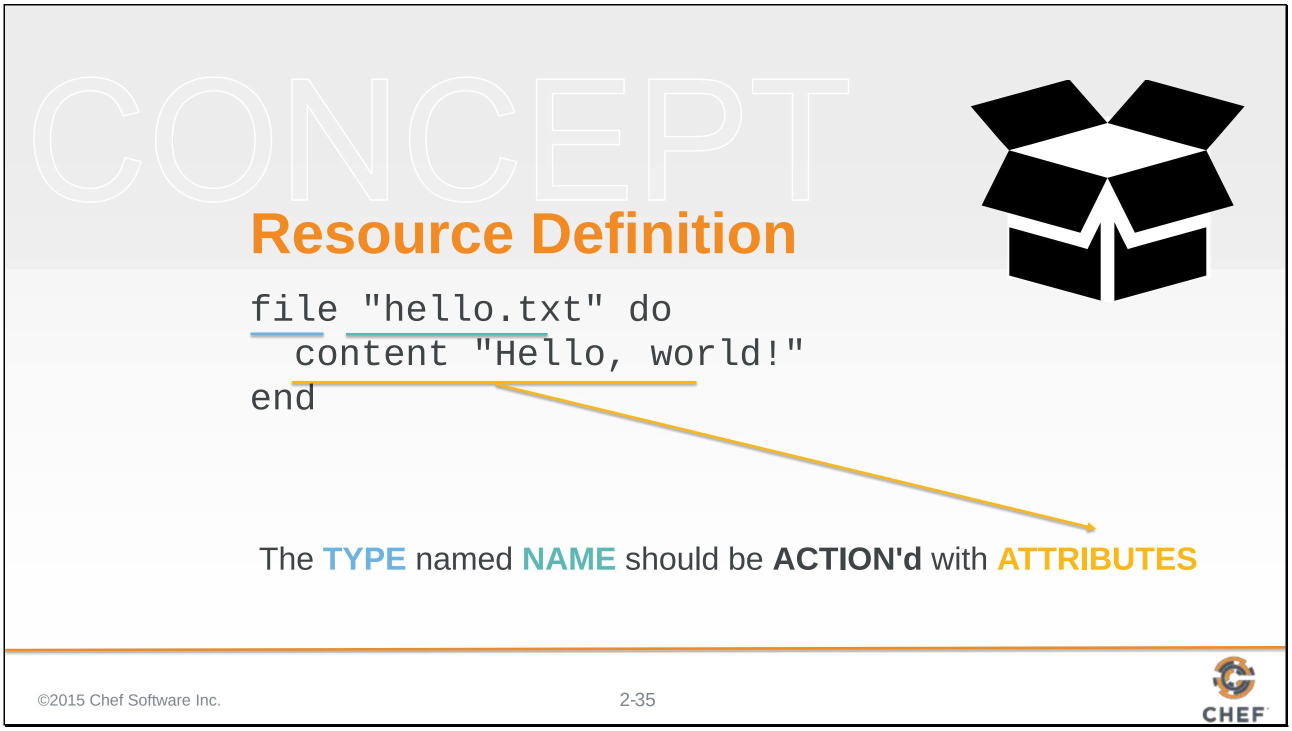
Slide 34



The second element is the name of the resource. This is also the first parameter being passed to the resource.

In this instance the resource name is also the relative file path to the file we want created. We could have specified a fully-qualified file path to ensure the file was written to the exact same location and not dependent on our current working directory.

Slide 35

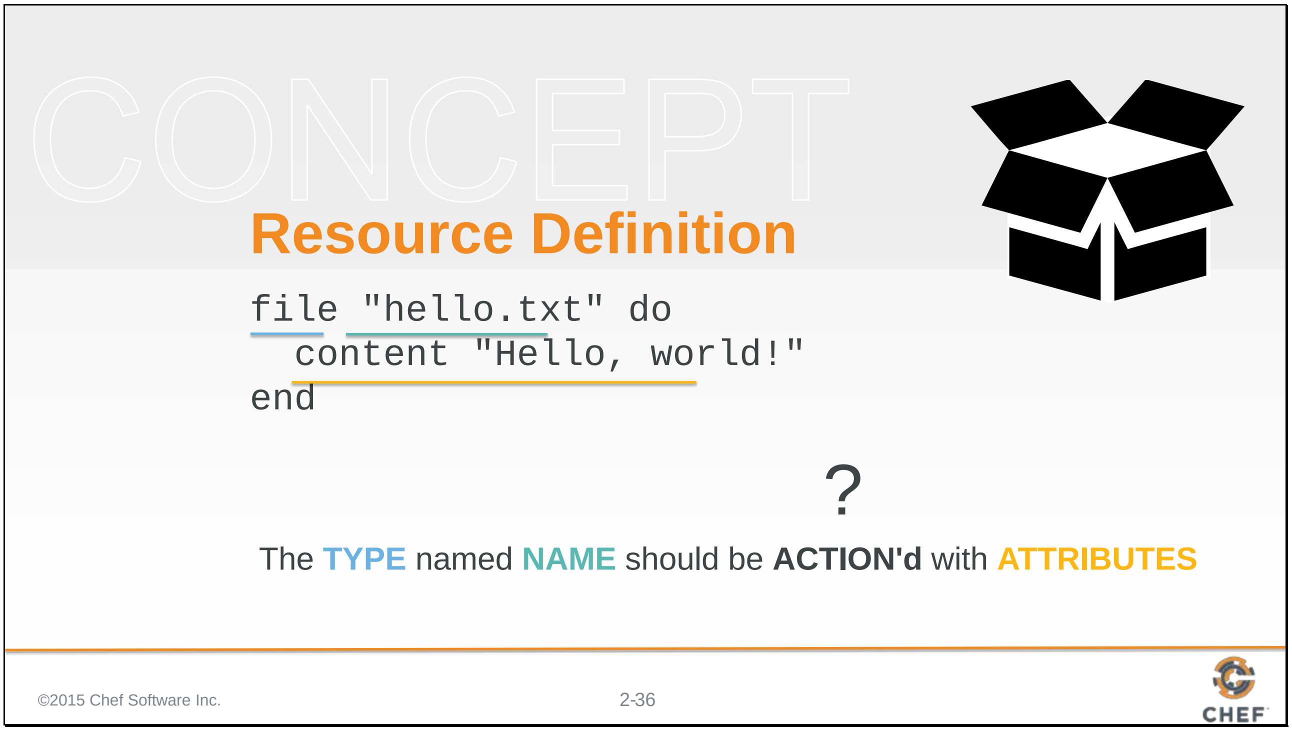


The `do` and `end` keywords here define the beginning of a ruby block. The ruby block and all the contents of it are the second attributes to our resource.

The contents of this block contains attributes (and other things) that help describe the state of the resource. In this instance, the source attribute here specifies the contents of the file.

Attributes are laid out with the name of the attributes followed by a space and then the value for the attribute.

Slide 36



The interesting part is that there is no action defined. And if you think back to the previous examples that we showed you, not all of the resources have defined actions.

So what action is the resource taking? How do you know?

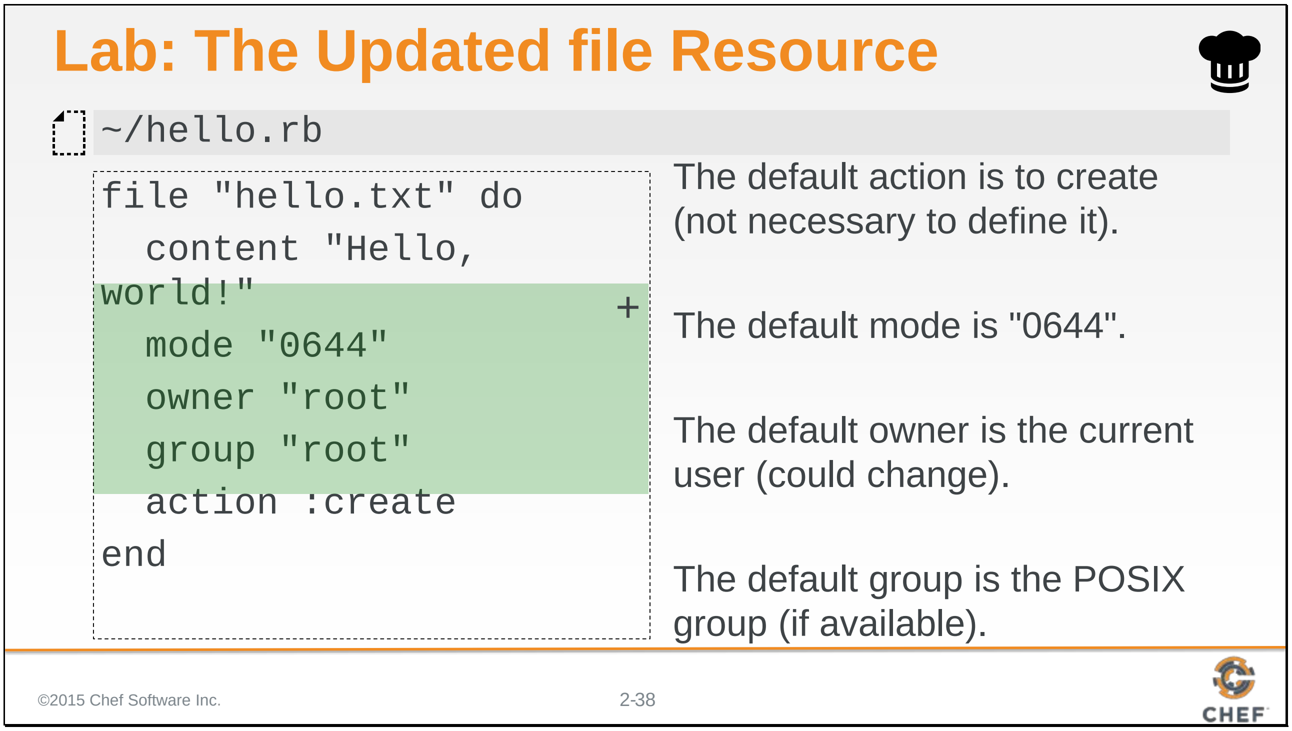
Slide 37

Could you find that information in the documentation for the file resource?

- Read through the file Resource documentation.
- Find the list of actions and then see if you can find the default one.
- Find the list of attributes and find the default values for mode, owner, and group.

The reason for doing this is that we want you to return to the file resource in the the recipe file and add the action, if necessary, and attributes for mode, owner and group.

Slide 38



The file resources default action is to create the file. So if that is the policy we want our system to adhere to then we don't need to specify it. It doesn't hurt if you do, but you will often find when it comes to default values for actions we tend to save ourselves the keystrokes and forgo expressing them.

The file resource in the recipe may or may not need to specify the three attributes: mode; owner; and group.

The mode default value is "0644". That value could change depending on the Operating System we are currently running.

The default owner is the current user. That value could change depending on who applies this policy.

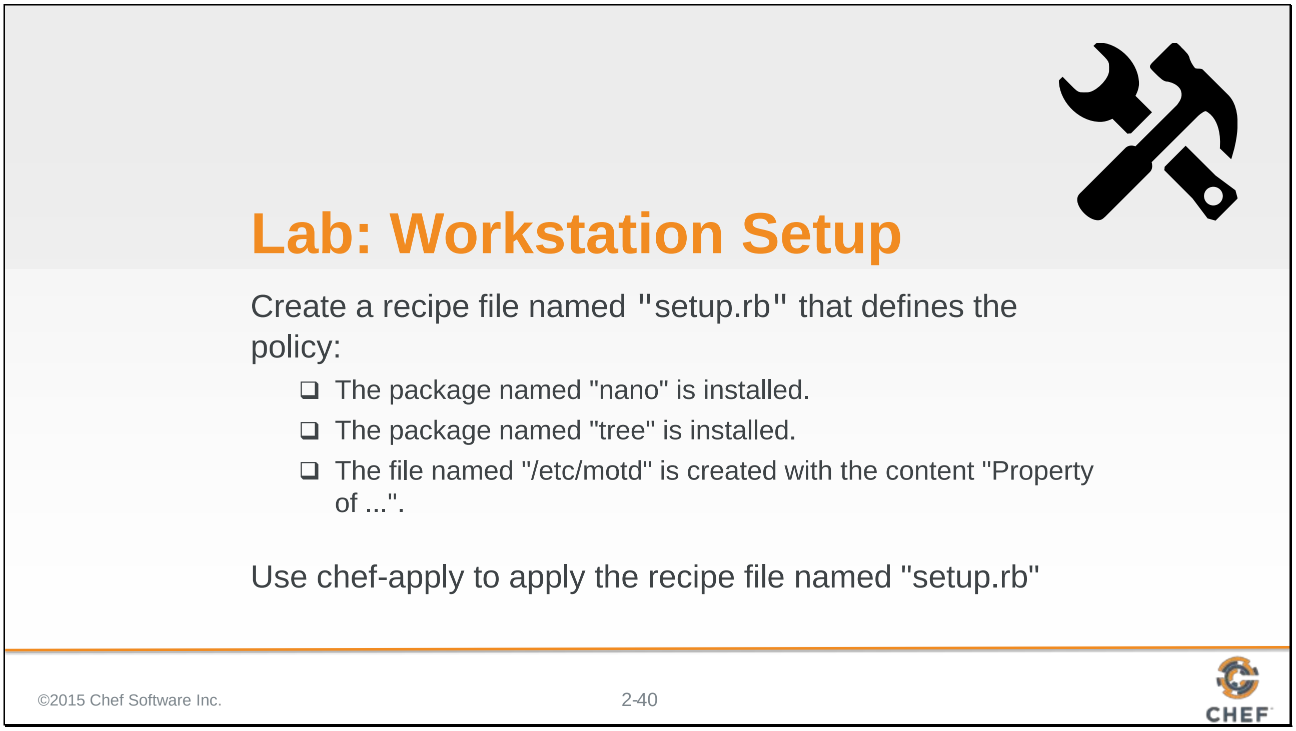
The default group is the POSIX group. In this instance this will be root. This could change depending on the system.

Slide 39

Questions

What questions can we answer for you?

Slide 40



Now that you've practiced:

- Installing an application with the package resource
- Creating a recipe file
- Creating a file with the file resource

Create a recipe that defines the following resource as its policy. When you are done defining the policy apply the policy to the system.

Slide 41

Lab: Workstation Setup Recipe File

~/setup.rb

```
package "nano"
package "vim"
package "emacs"

package "tree"

file "/etc/motd" do
  content "Property of ..."
end
```

The package named "nano" is installed.

The package named "tree" is installed.

The file named "/etc/motd" is created with the content "Property of ...".

Here is a version of the recipe file that installs all the editors, our tree package, and creates the message-of -the-day file.

Slide 42

Lab: Apply the Setup Recipe

```
$ sudo chef-apply setup.rb
```

```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * apt_package[vim] action install (up to date)
  * apt_package[tree] action install
    - install version 1.6.0-1 of package tree
  * file[/etc/motd] action create
    - create new file /etc/motd
    - update content in file /etc/motd from none to d100eb
    --- /etc/motd   2015-05-11 23:17:00.869570000 +0000
    +++ /etc/.motd20150511-1762-trppu1  2015-05-11 23:17:00.865570000 +0000
    @@ -1 +1,2 @@
    +Property of ...
```


This is how you apply the created recipe.

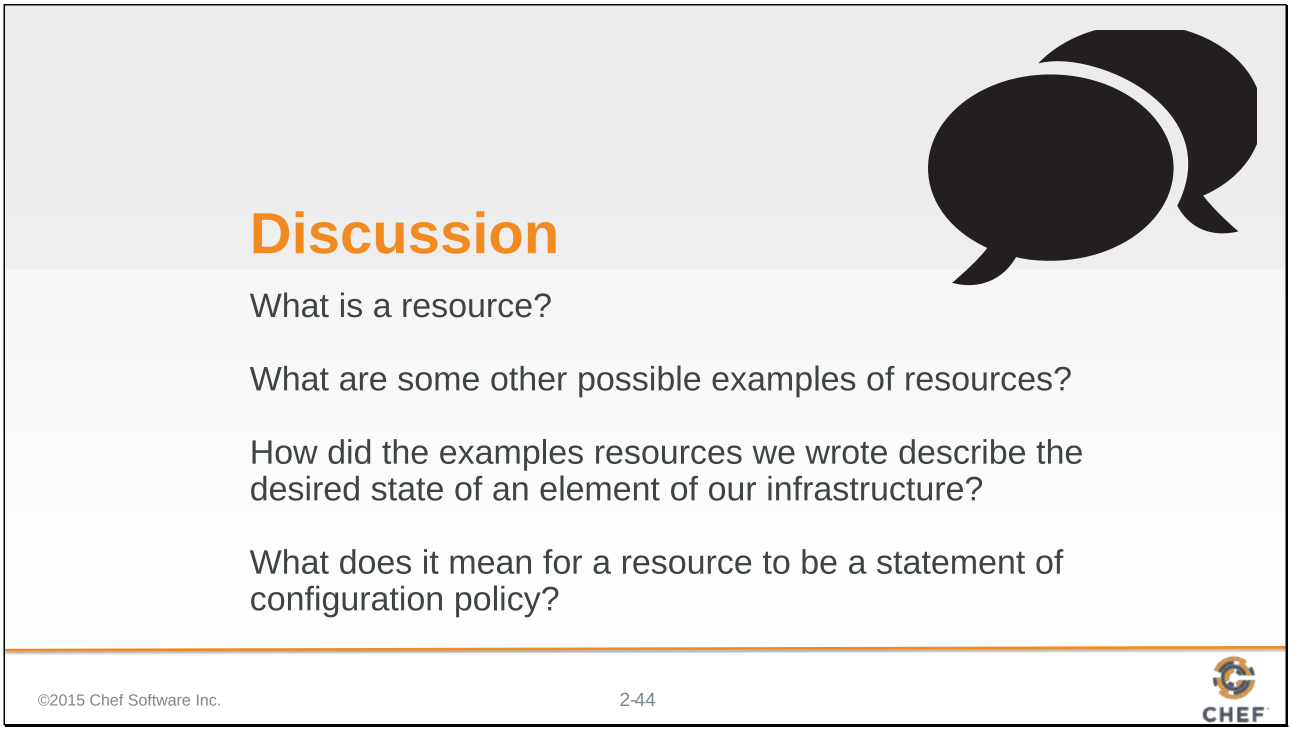
Slide 43

Let's finish this Resources module with a discussion.

Write down or type out a few words for each of these questions. Talk about your answers with each other.

Remember that the answer "I don't know! That's why I'm here!" is a great answer.

Slide 44

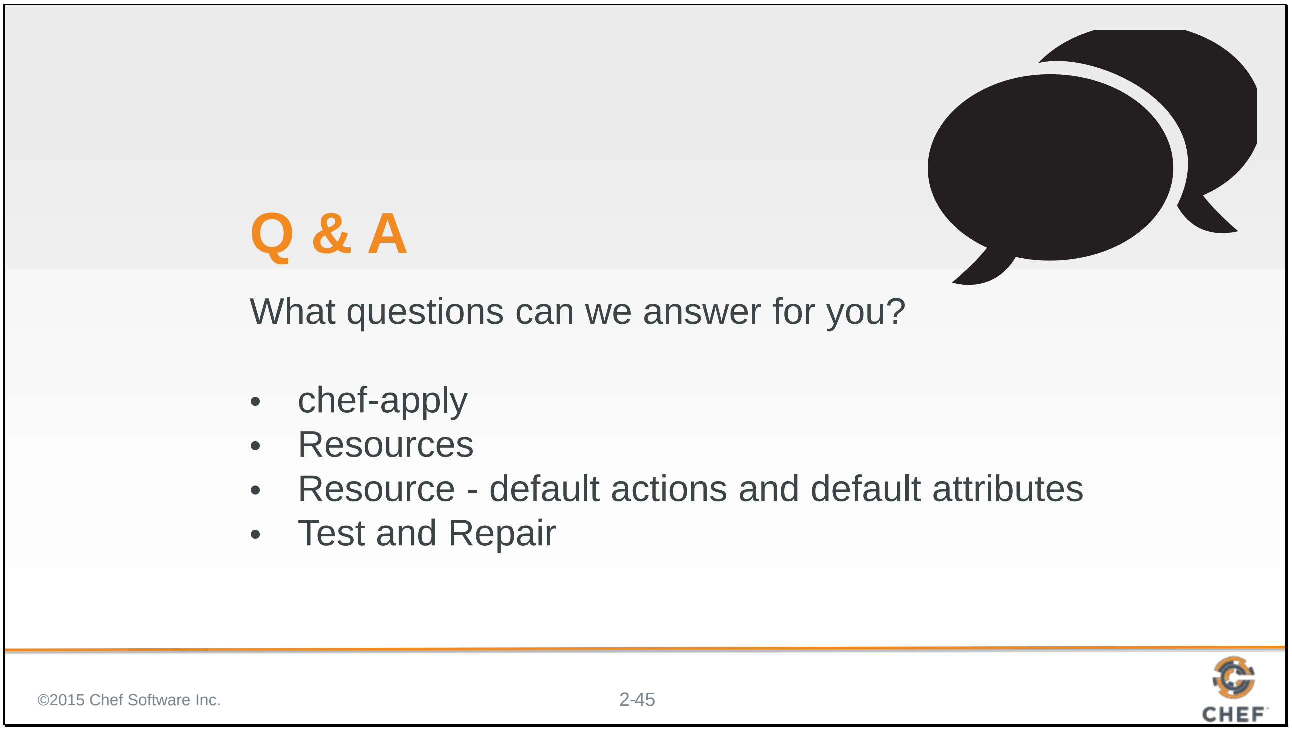


Answer these four questions:

- What is a resource?
- What are some other possible examples of resources?
- How did the examples resources we wrote describe the desired state of an element of our infrastructure?
- What does it mean for a resource to be a statement of configuration policy?

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 45



What questions can we answer for you?

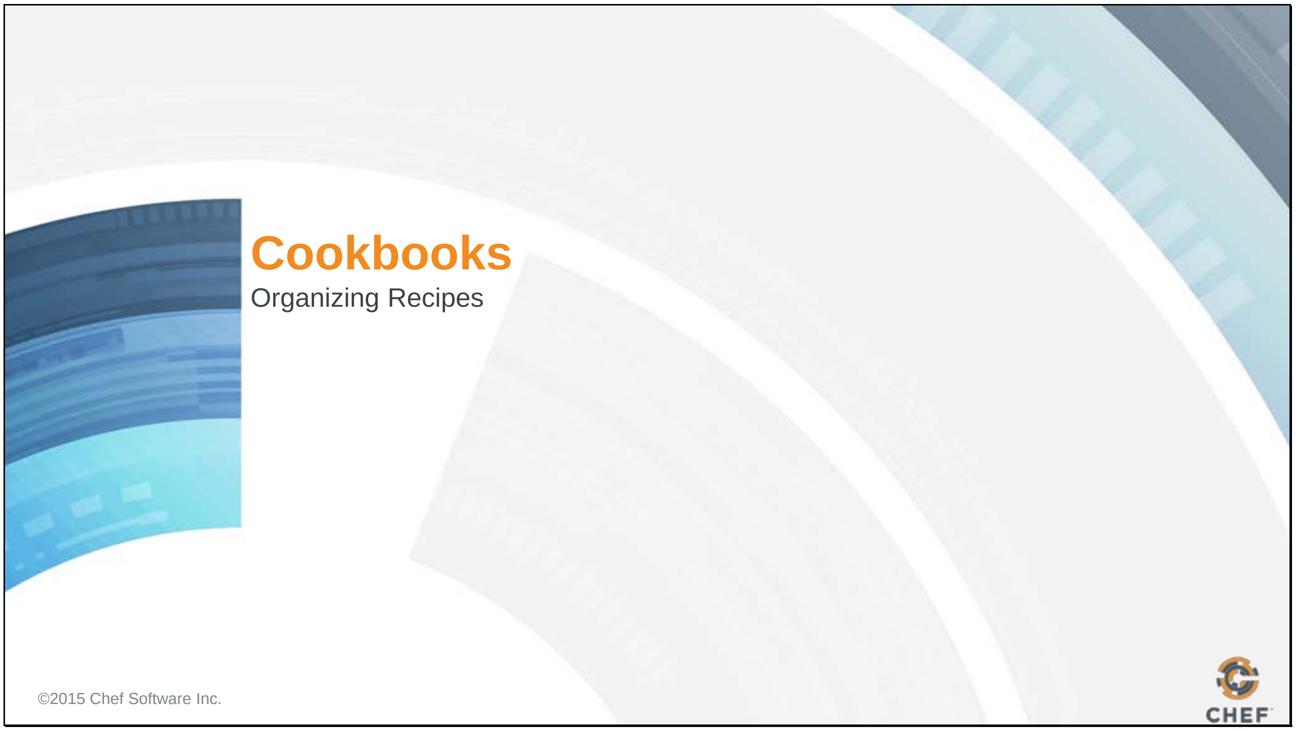
About anything or specifically about:

- `chef-apply`
- resources
- a resources default action and default attributes
- Test and Repair

Slide 46

CHEF™

3: Cookbooks



Slide 2

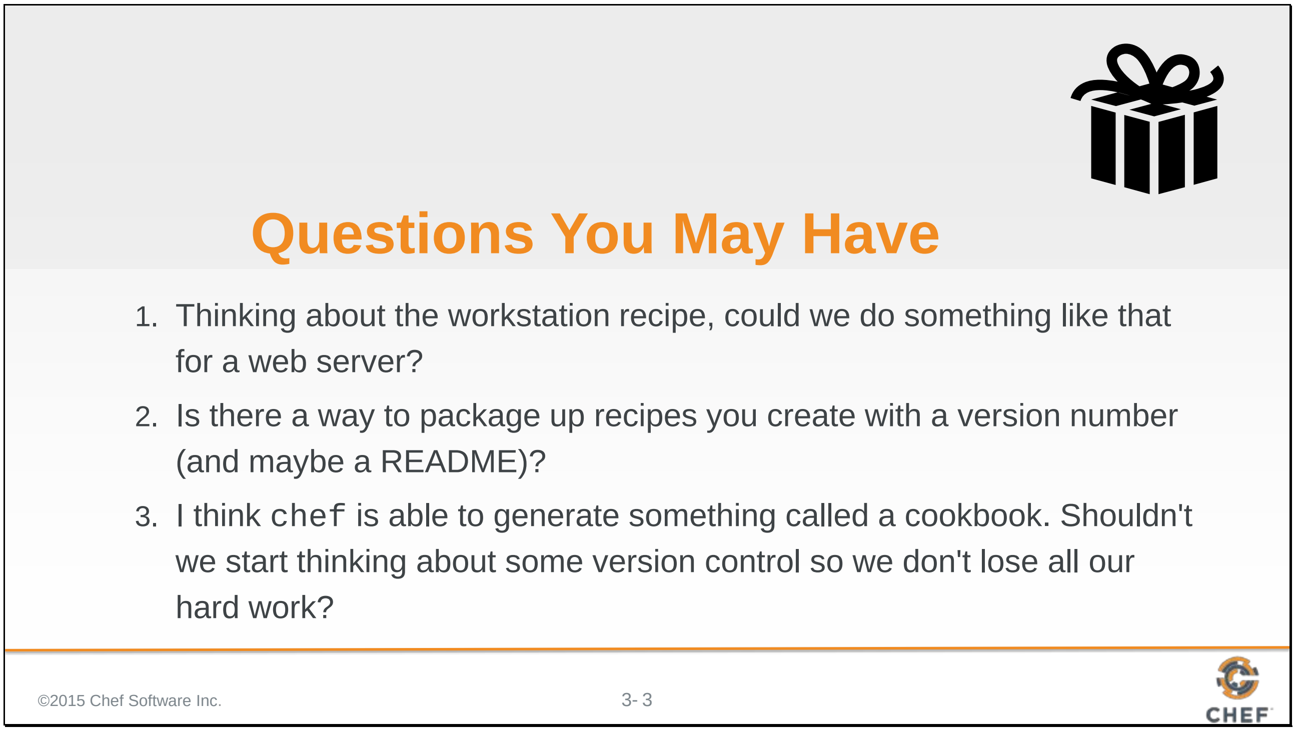
Objectives

After completing this module, you should be able to:

- Modify a recipe
- Use version control
- Generate a Chef cookbook
- Define a Chef recipe that sets up a web server

In this module you will learn how to modify a recipe, use version control, generate a Chef cookbook and define a Chef recipe that sets up a web server.

Slide 3



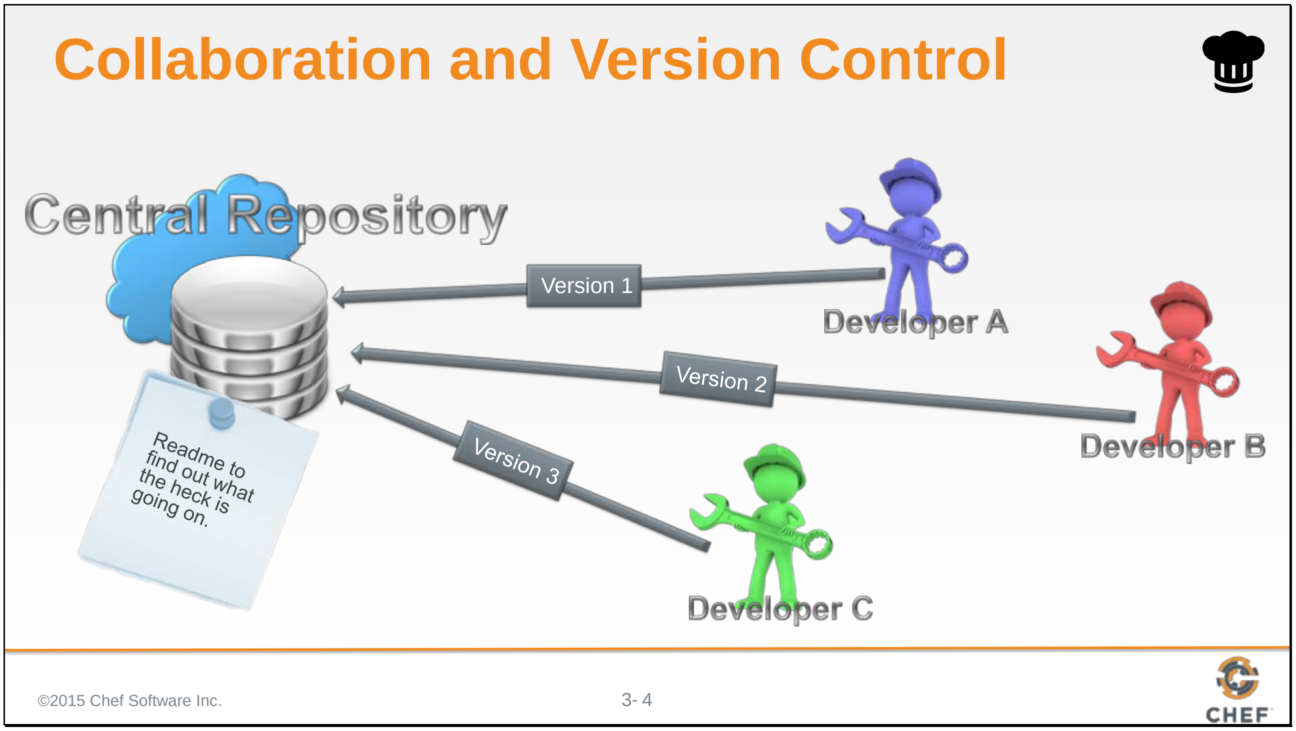
Answers:

1. The recipe that you put together to setup the workstation proved useful--useful enough to see if the same could be done with a webserver. It's a package, a file, and a service. Everything you've already completed. Well, almost everything.

2. Now the request to add version control and a README would definitely make it easier to share the recipes that we create. Without version control we'd have no way to build this software collaboratively or recover our work. Without a README no one would know what the recipe even was suppose to do or what it did.

3. And yes, before we start creating more recipes and cookbooks, we should choose a versioning solution.

Slide 4



Before we answer that question, let's talk about collaboration. Usually, none of us work in a vacuum, and it's important that systems are in place to make collaboration easier. One such system is versioning. Versioning will make it easier to share the recipes that we create.

A versioning system should include:

A Central Repository into which all the developers publish their work.
Each revision should be stored as a new version.
For each change, a readme file should be added so that everyone knows what has or has not been changed

Slide 5

Versioning Pros and Cons

```
$ cp setup.rb setup.rb.bak
or
$ cp foo{,.`date +%Y%m%d%H%M`}
or
$ cp foo{,.`date +%Y%m%d%H%M`-`$USER`}
```

Saving a copy of the original file as another filename.

Lets explore this first option of renaming the file by adding a quick extension, like in the first example shown here. In this way we can keep working on the original file as we add more features. As a group lets talk about the pros and cons of using this strategy.

So obviously a single backup won't do. We need backups more often as we are going to be iterating quickly.

We could use the current date and time down to the minute like in the second example. As a group lets talk about the pros and cons of using this strategy.

Would adding the user's name to the end of the file, like in the third example, solve the problems we are facing with other choices? Again what are the pros and cons of this new approach?

Slide 6

Git Version Control

git is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.

We will be using **git** throughout the rest of this course.

How about we use git?

What are the pros and cons of this approach?

For the rest of this course we will be using git. This may not be the version control software you use on your teams or within your organization and that is alright. Our use of git within this course is used solely to demonstrate the use of version control when developing Chef code. When you develop with Chef you are welcome to use the version control system of your choice.

Slide 7

Is git installed? Do we know if it will be installed with every new instance that is setup?

It sounds like we need the tool now to store our cookbook but we also want to define a policy that git is installed on all of our workstations. Update the setup recipe to define the new policy and apply the setup recipe again.

Slide 8

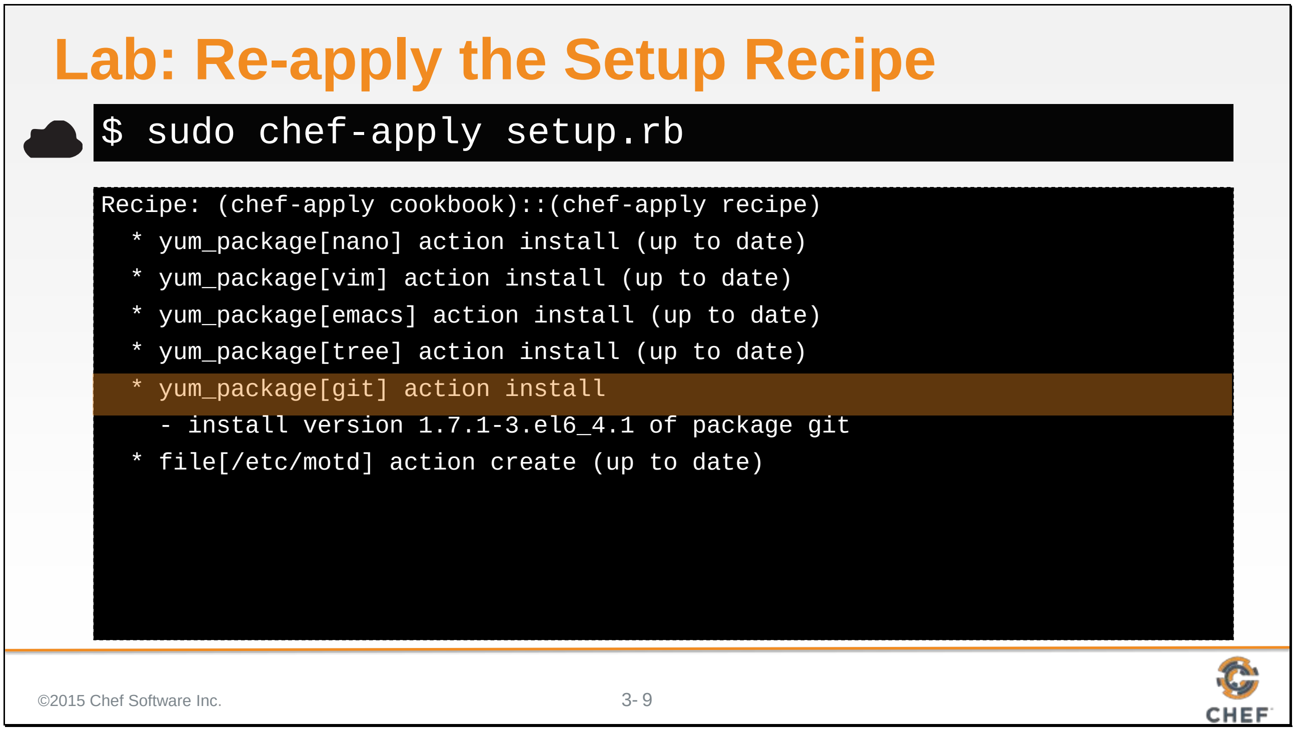
Lab: Adding the git Package

~/setup.rb

```
package "nano"
package "vim"
package "emacs"

package "tree"
package "git"                                              +

file "/etc/motd" do
  content "Property of ..."
end
```

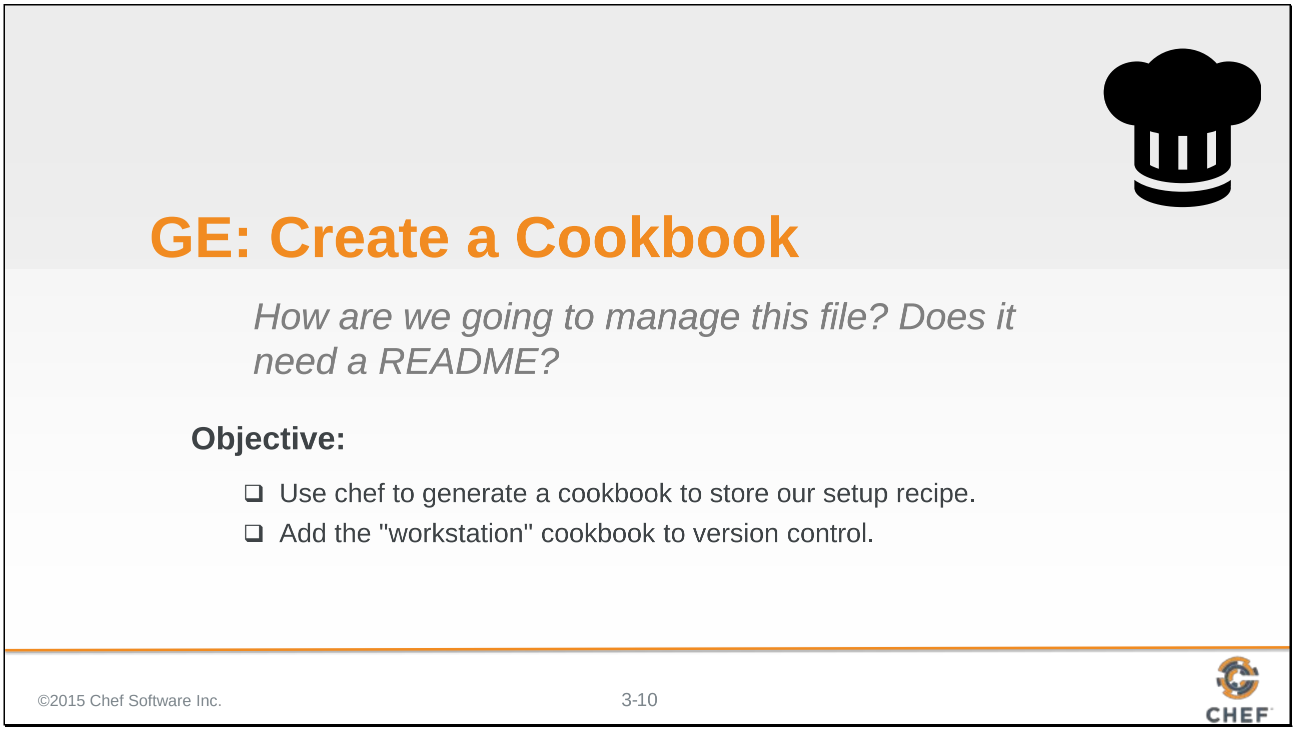

We add a package resource named 'git' to the setup recipe within our workstation cookbook.

Slide 9



Then we use chef-apply to apply our recipe.

Slide 10

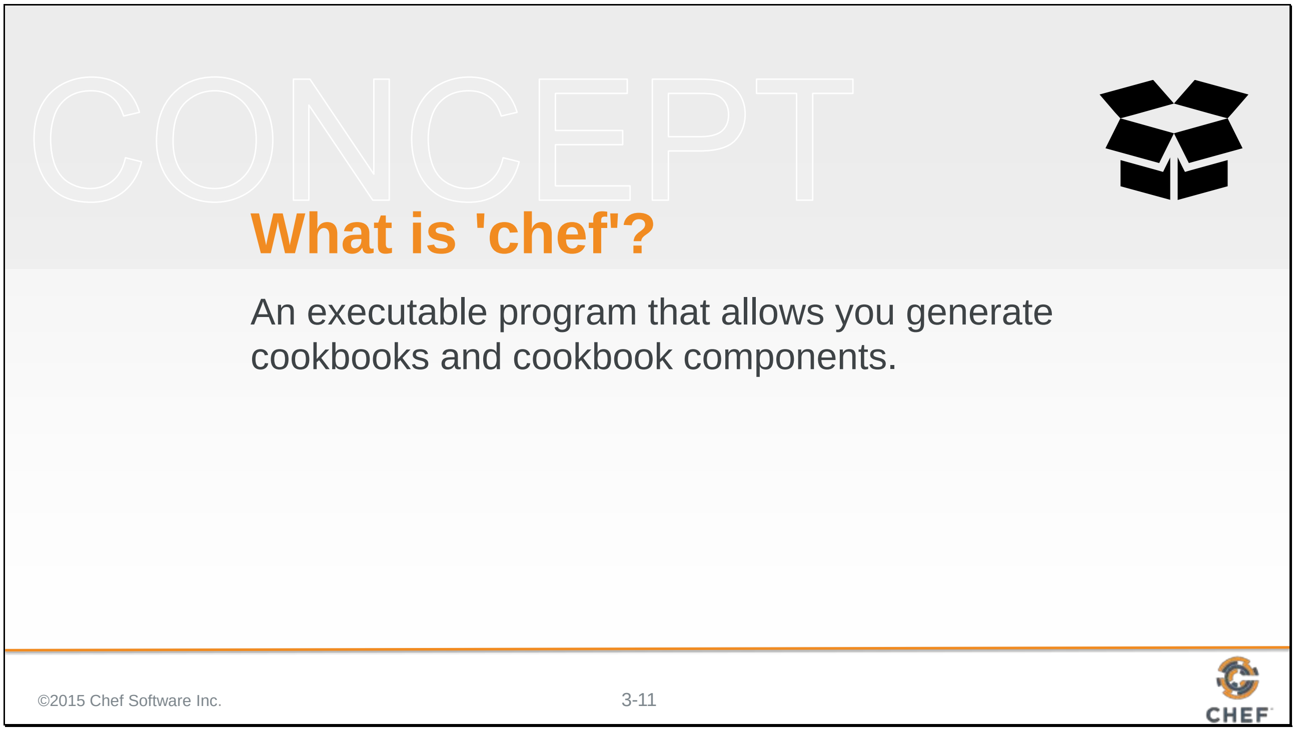


The setup recipe now installs everything we currently need on our workstation.

But before throw this recipe file into a directory with our other scripts we should look at a concept in Chef called a cookbook.

What is a cookbook? How do we create one? Let's ask 'chef'.

Slide 11



In this context, 'chef' is a command, not the company.

What's the best way to learn Chef? Use Chef. We want you to literally run 'chef'.

Slide 12

What can 'chef' do?

```
$ chef --help
```

```
Usage:
    chef -h/--help
    chef -v/--version
    chef command [arguments...] [options...]

Available Commands:
    exec        Runs the command in context of the embedded ruby
    gem         Runs the `gem` command in context of the embedded ruby
    generate    Generate a new app, cookbook, or component
    shell-init  Initialize your shell to use ChefDK as your primary ruby
    install     Install cookbooks from a Policyfile and generate a locked cookbook
set
    update      Updates a Policyfile.lock.json with latest run_list and cookbooks
```


'chef' is a command-line application that does quite a few things. The most important thing to us right now is its ability to generate cookbooks and components.

Alright. So 'chef' can generate a cookbook. But what is the purpose of a cookbook? That sounds like we should read the documentation.

Slide 13

Cookbooks

A Chef cookbook is the fundamental unit of configuration and policy distribution.

Each cookbook defines a scenario, such as everything needed to install and configure MySQL, and then it contains all of the components that are required to support that scenario.

Read the first three paragraphs here: http://docs.chef.io/cookbooks.html

It's important that you learn to read the Chef documentation. Lets look up cookbooks in Chef's documentation. Visit the docs page on cookbooks and read the first three paragraphs.

Slide 14

What Can 'chef generate' Do?

```
$ chef generate --help
```

```
Usage: chef generate GENERATOR [options]

Available generators:
  app         Generate an application repo
  cookbook    Generate a single cookbook
  recipe      Generate a new recipe
  attribute   Generate an attributes file
  template    Generate a file template
  file        Generate a cookbook file
  lwrp        Generate a lightweight resource/provider
  repo        Generate a Chef policy repository
  policyfile  Generate a Policyfile for use with the install/push commands
(experimental)
```


A cookbook is a structure that contains recipes. It also contains a number of other things--but right now we are most interested in a finding a home for our recipes, giving them a version, and providing a README to help describe them.

Slide 15

What Can 'chef generate cookbook' Do?

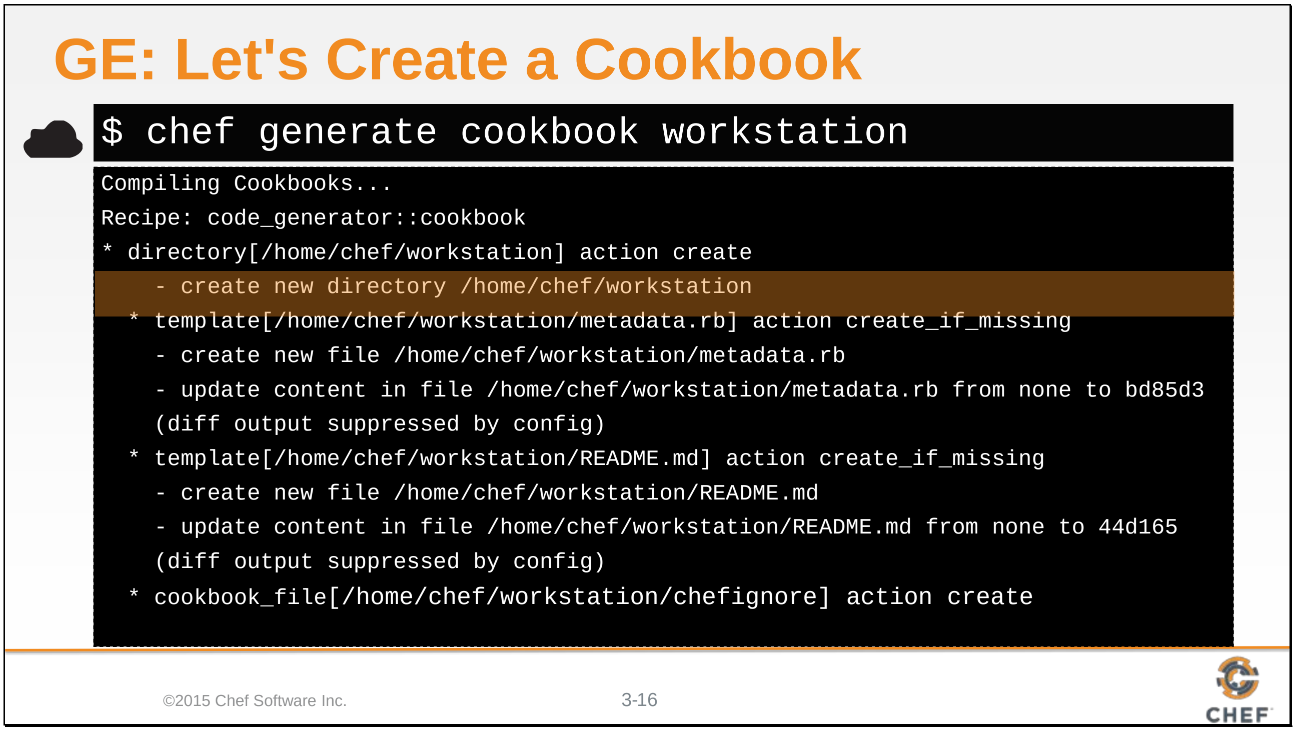
```
$ chef generate cookbook --help
```

```
Usage: chef generate cookbook NAME [options]
    -C, --copyright COPYRIGHT        Name of the copyright holder - defaults
to 'The Authors'
    -m, --email EMAIL                Email address of the author - defaults to
'you@exa...
    -a, --generator-arg KEY=VALUE    Use to set arbitrary attribute KEY to
VALUE in the...
    -I, --license LICENSE            all_rights, httpd, mit, gplv2, gplv3 -
defaults ...
    -g GENERATOR_COOKBOOK_PATH,      Use GENERATOR_COOKBOOK_PATH for the
code_generator...
        --generator-cookbook
```


Lets examine the '`chef generate`' command. We can see that the command is capable of generating a large number of different things for us. It looks like if we want to generate a cookbook we're going to need to use 'chef generate cookbook'.

Lets ask the 'chef generate cookbook' command for help to see how it is used.

Slide 16



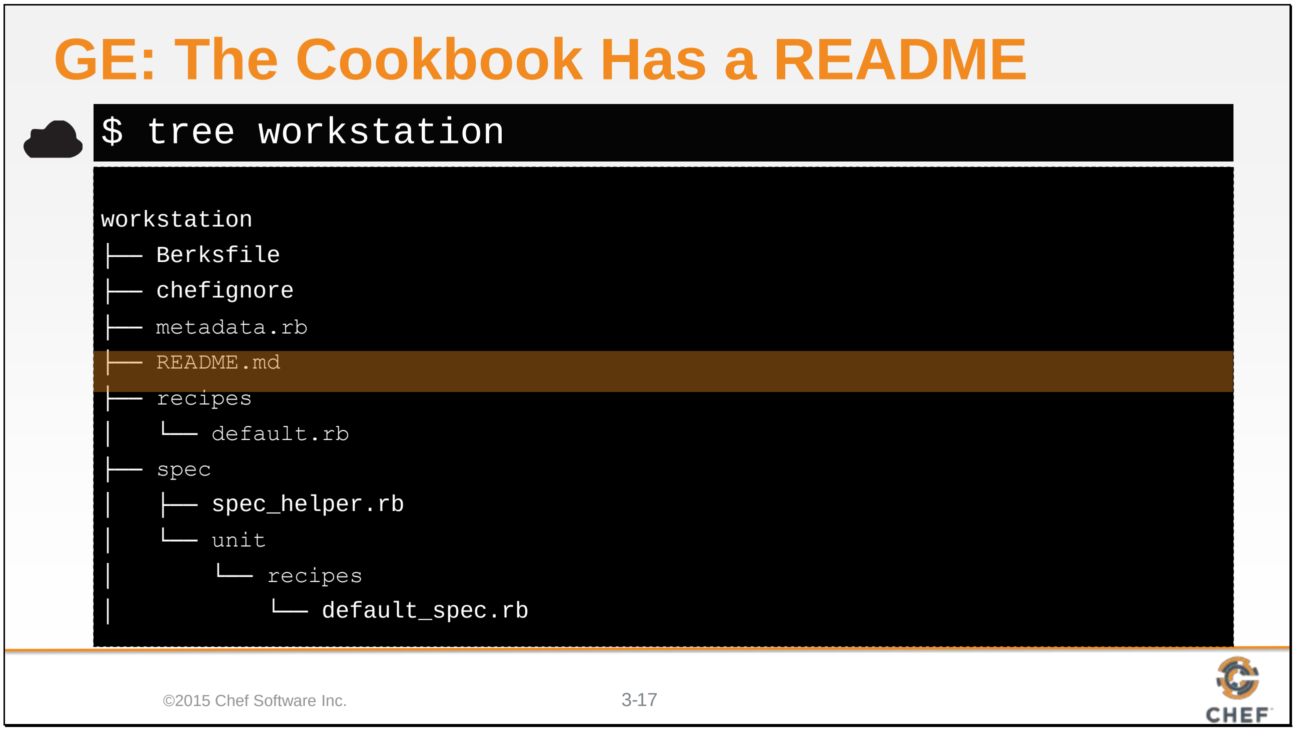
To generate a cookbook, all we have to do is provide it with a name.

Naming things: There are two hard things in Computer Science and one of those is giving something a name.

We have you covered. Call the cookbook *workstation*. That's a generic enough name.

We want you to use '`chef generate`' to generate a cookbook named *workstation*.

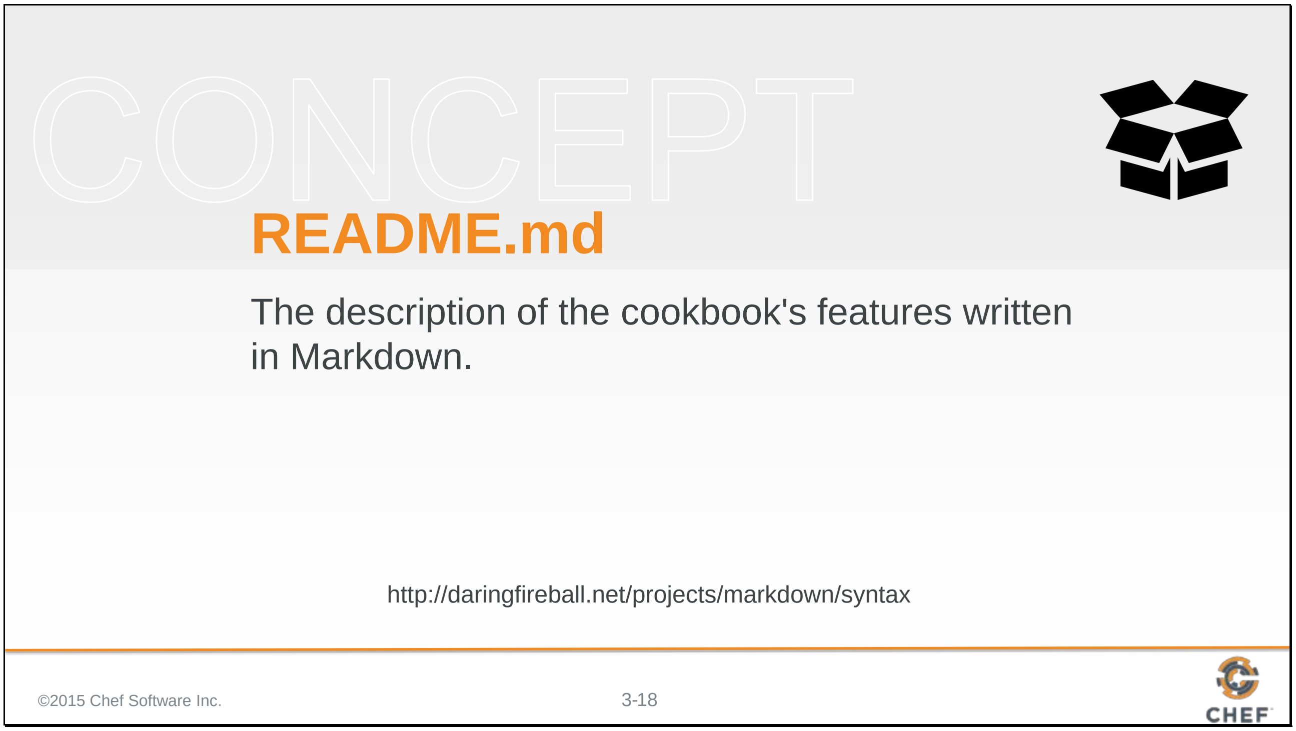
Slide 17



Aren't you curious what's inside it? Lets take a look with the help of the '**`tree`**' command. If we provide '**`tree`**' with a path we will see all the visible files in the specified directory.

So the chef cookbook generator created an outline of a cookbook with a number of default files and folders. The first one we'll focus on is the README.

Slide 18



All cookbooks that 'chef' will generate for you will include a default README file. The extension `.md` means that the file is a markdown file.

Markdown files are text documents that use various punctuation characters to provide formatting. They are meant to be easily readable by humans and can be easily be rendered as HTML or other formats by computers.

Slide 19

GE: The Cookbook Has Some Metadata

```
$ tree workstation
```

```
workstation
├── Berksfile
├── README.md
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
10 directories, 9 files
```


CHEF

The cookbook also has a metadata file.

Slide 20

DOCS

metadata.rb

Every cookbook requires a small amount of metadata. Metadata is stored in a file called metadata.rb that lives at the top of each cookbook's directory.

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

This is a ruby file that contains its own domain specific language (DSL) for describing the details about the cookbook.

Slide 21

GE: Let's Take a Look at the Metadata

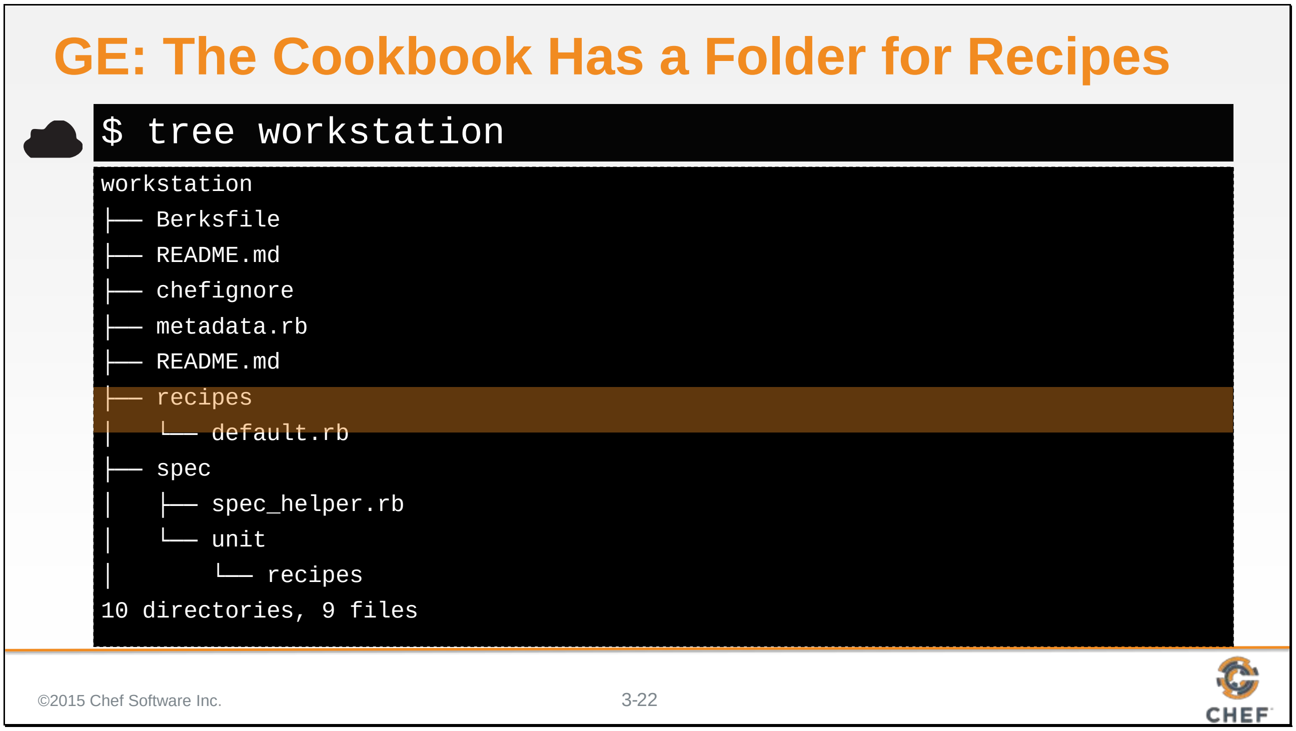
```
$ cat workstation/metadata.rb
```

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


If you view the contents of your new cookbook's metadata, you'll see a number of details that help describe the cookbook:

The name of the cookbook, its maintainer, a way to reach them, how the cookbook is licensed, descriptions, and the cookbook's version number.

Slide 22



The cookbook also has a folder named *recipes*. This is where we store the recipes in our cookbook. You'll see that the generator created a default recipe in our cookbook. What does it do?

Slide 23

GE: The Cookbook Has a Default Recipe

```
$ cat workstation/recipes/default.rb
```

```
# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
```


Looking at the contents of the default recipe you'll find it's empty except for some ruby comments.

A cookbook doesn't have to have a default recipe but most every cookbook has one. It's called *default* because when you think of a cookbook, it is probably the recipe that defines the most common configuration policy.

Slide 24

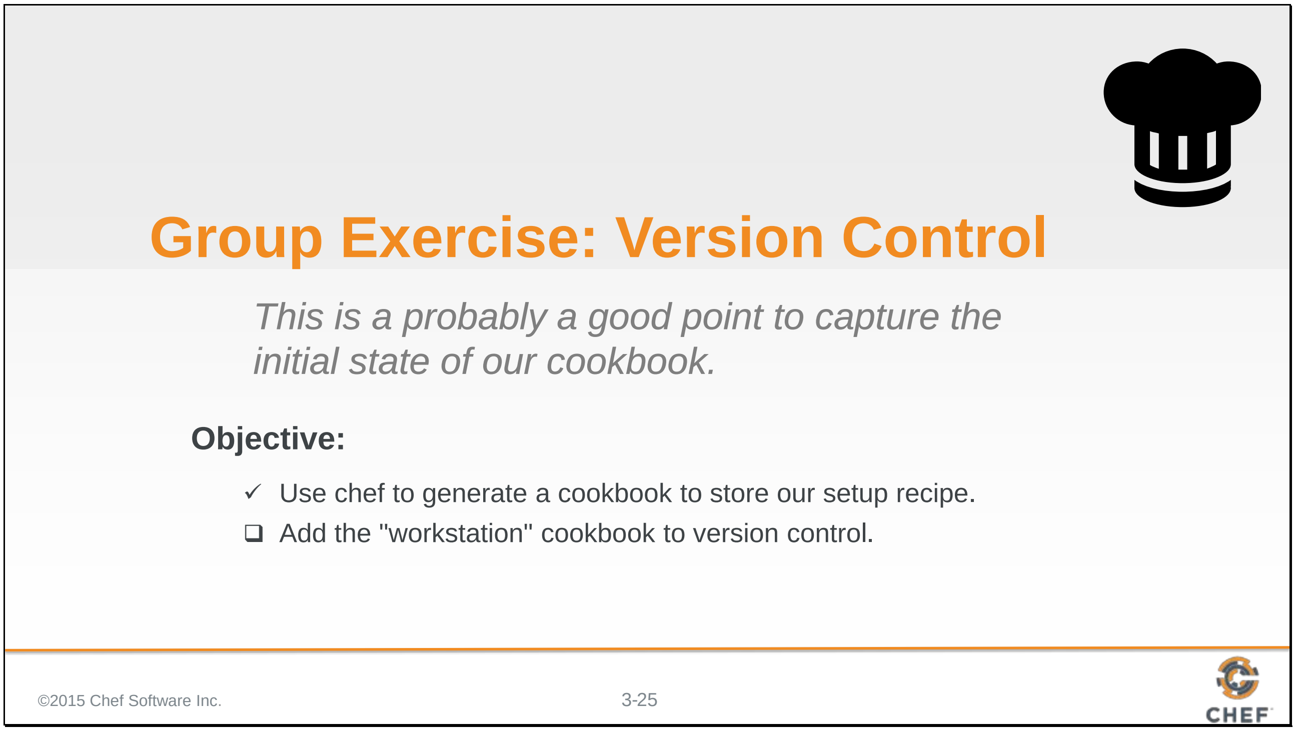
GE: Copy the Recipe into the Cookbook

```
$ mv setup.rb workstation/recipes/setup.rb
```


CHEF

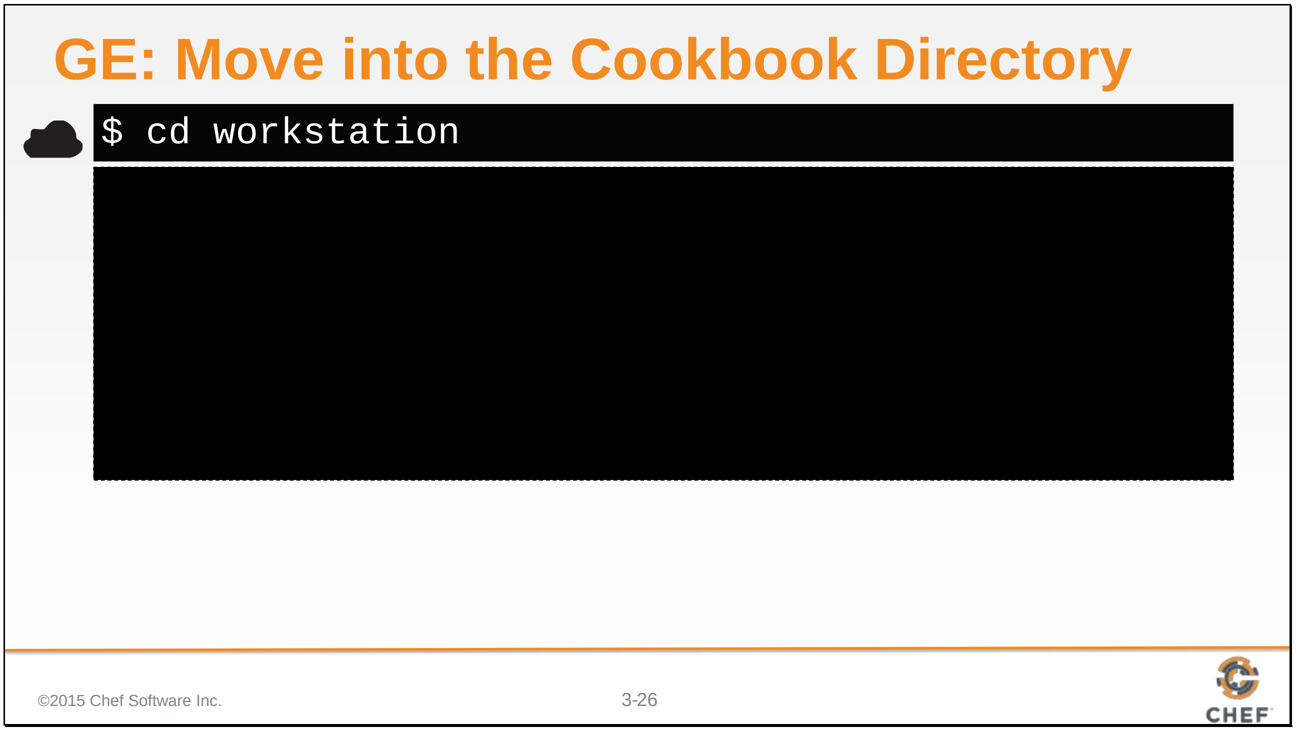
From the Home directory, move your setup.rb recipe to the workstation cookbook and place it alongside our default recipe.

Slide 25



Now that we have our cookbook with its README and version number, it's time to start tracking our changes with git.

Slide 26

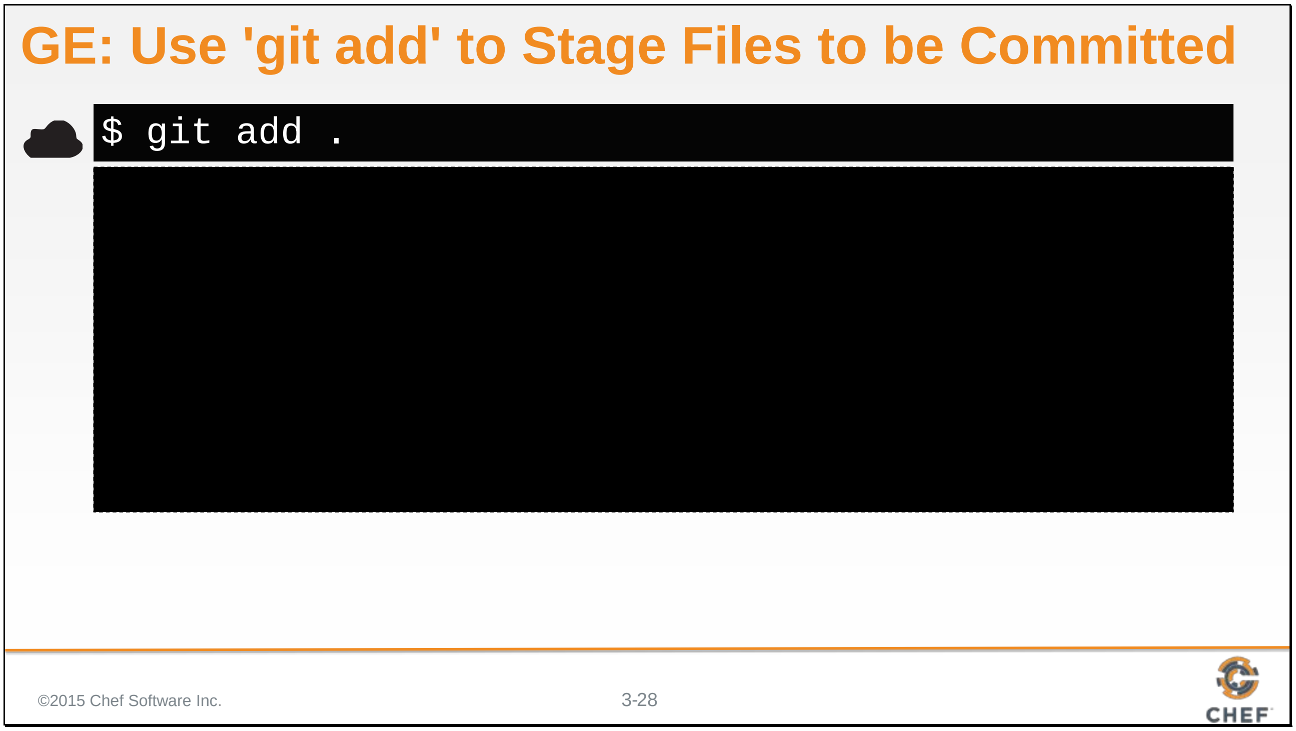


Change into the workstation cookbook directory.

Slide 27

We want git to start tracking the entire contents of this folder and any content in the subfolders. To do that with git, you need to execute the command 'git init' in the parent directory of the cookbook that you want to start tracking.

Slide 28



Now we need to tell git which files it should start tracking in source control. In our case, we want to add all the files to the repository and we can do that by executing `'git add .'` (dot).

This will place all the files into a staging area.

Slide 29

CONCEPT

Staging Area

The staging area has a file, generally contained in your Git directory, that stores information about what will go into your next commit.

It's sometimes referred to as the "index", but it's also common to refer to it as the staging area.

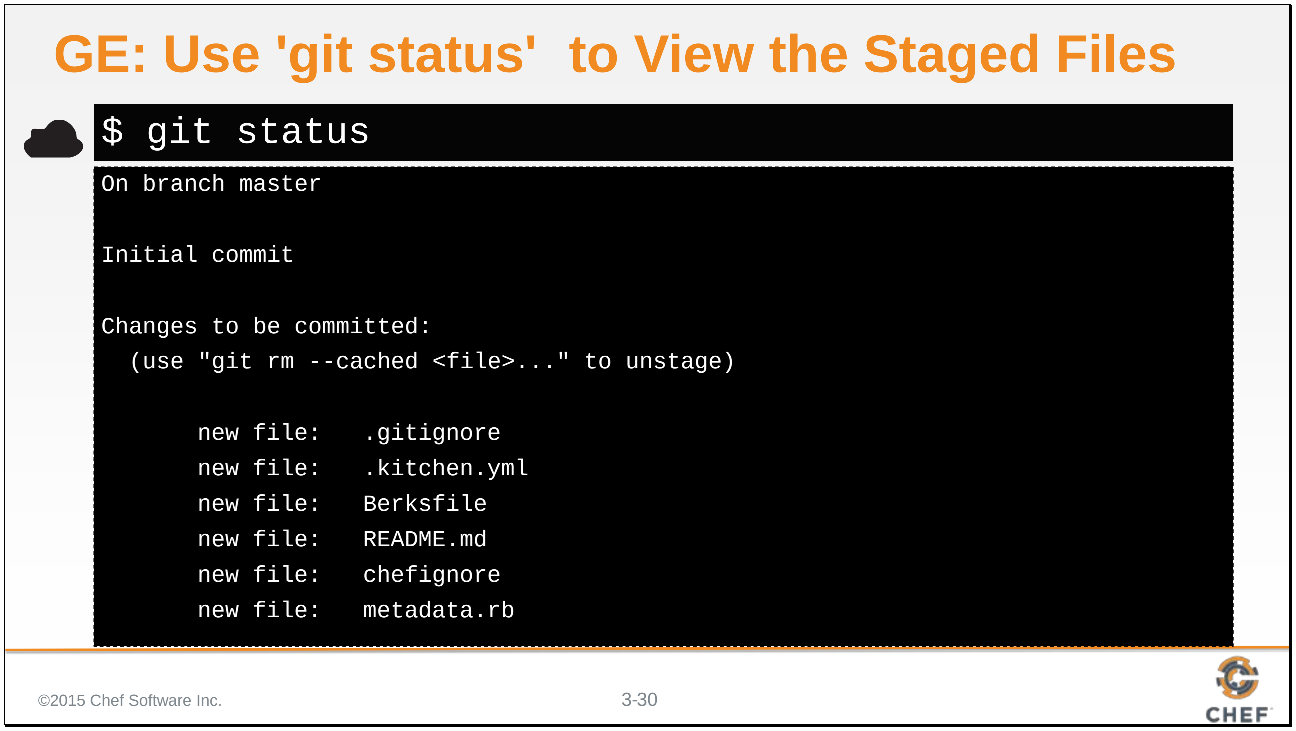
http://git-scm.com/book/en/v2/Getting-Started-Git-Basics

CHEF

You can think of the staging area as a box in which to put a bunch of items -- like a care package you would send to someone.

Staging files means to put them in the box, but don't close it up because you may add a few things, and don't close it up because you may replace or remove a few things. But put the items in the box because eventually we are going to close that box when it is ready to send it off.

Slide 30



Lets see what changes we have placed in the staging area.

Thinking about our care package example, this is like looking inside the box and taking an inventory, allowing us to figure out if we need to move more things in or remove things we accidently threw in there.

Running `git status` allows us to see in the box. Git reports back to us the changes that will be committed.

Slide 31

GE: Use 'git commit' to Save the Staged Changes

```
$ git commit -m "Initial workstation cookbook"
```

```
master (root-commit) 9998472] Initial workstation cookbook
 Committer: ChefDK User <chef@ip-172-31-59-191.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author
```


If everything that is staged looks correct, then we are ready to commit the changes.

This is like saying we're ready to close the box up.

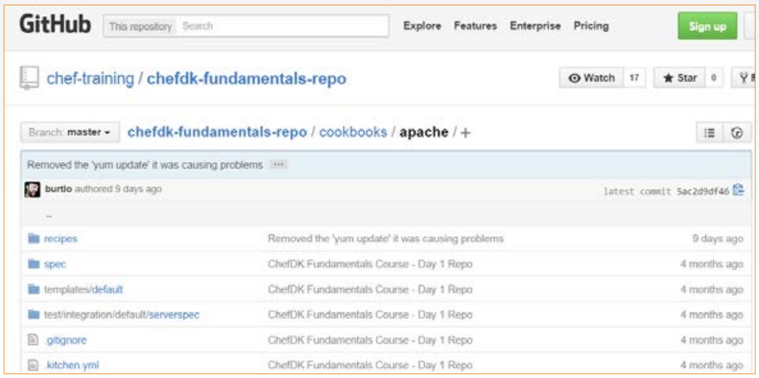
This is done in git with **git commit**. We can optionally provide a message on the command-line and that is done with the **-m** flag and then a string of text that describes that change.

Slide 32

Git Version Control

If you use git versioning you should ultimately push the local git repository to a shared remote git repository.

In this way others could collaborate with you from a centralized location.



git tracks all our commits, all those closed up boxes, locally on the current system. If we wanted to share those commits with other individuals we would need to push those changes to a central repository where we could collaborate with other members of the team.

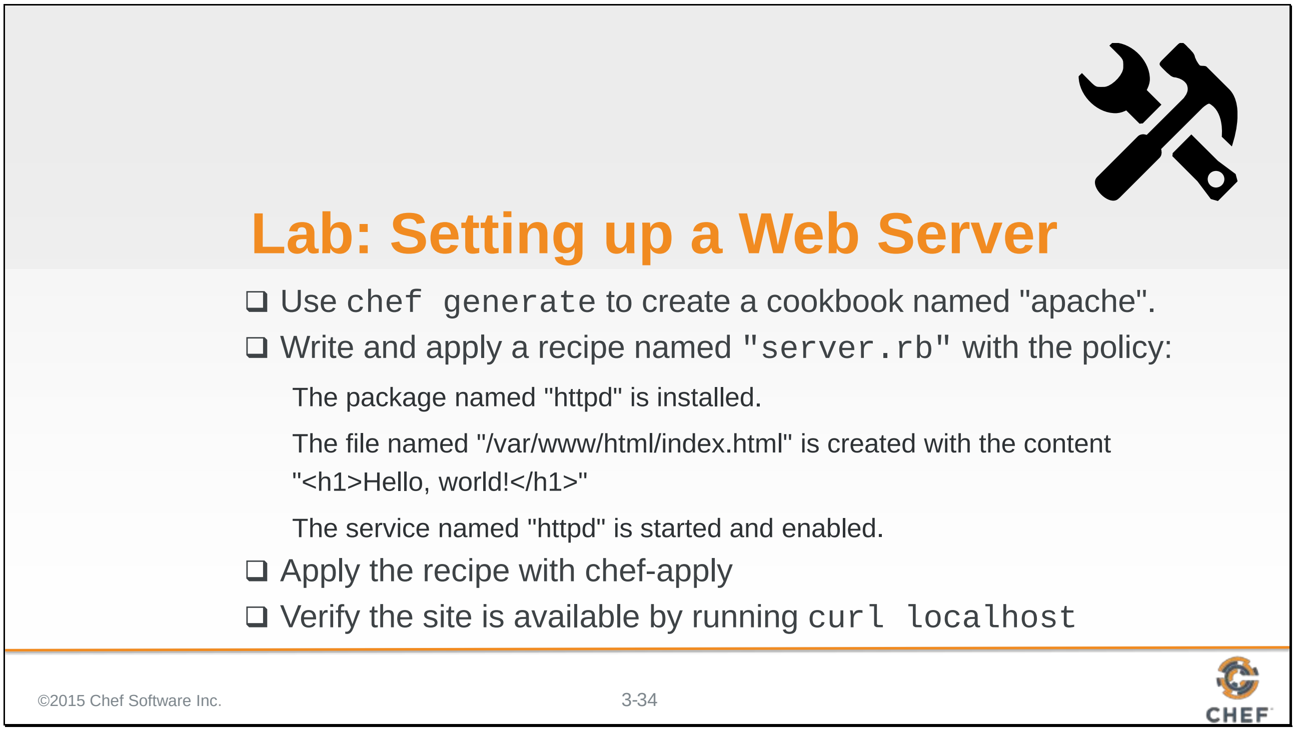
Slide 33

GE: Move out of the Workstation Cookbook

```
$ cd ~
```


Now that we are done adding our workstation cookbook to version control lets return to our home directory.

Slide 34



Now. Here is your latest challenge. Deploying a Web Server with Chef.

Thinking about all that we have accomplished so far that hopefully seems possible.

We need a cookbook named apache that has a server recipe. Within that server recipe we need to install the appropriate package. Write out an example HTML file, and then start and enable the service.

Then we should apply that recipe and make sure the site is up and running by running a command to visit that site.

So show me it can be done!

Slide 35

Lab: Create a Cookbook

```
$ chef generate cookbook apache
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/home/chef/apache] action create
    - create new directory /home/chef
  * template[/home/chef/apache/metadata.rb] action create_if_missing
    - create new file /home/chef/apache/metadata.rb
    - update content in file /home/chef/apache/metadata.rb from none to bd85d3
    (diff output suppressed by config)
  * template[/home/chef/apache/README.md] action create_if_missing
    - create new file /home/chef/apache/README.md
    - update content in file /home/chef/apache/README.md from none to 44d165
    (diff output suppressed by config)
  * cookbook_file[/home/chef/apache/chefignore] action create
```


From the Chef home directory, run the command 'chef generate cookbook apache'. This will place the apache cookbook alongside the setup cookbook.

Slide 36

Lab: Create Apache Recipe

~/apache/recipes/server.rb

```
package "httpd"

file "/var/www/html/index.html" do
  content "<h1>Hello, world!</h1>"
end

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


The server recipe, found at ~/apache/recipes/server.rb, defines the policy:

* The package named **httpd** is installed.

* The file named "/var/www/html/index.html" is created with the content "Hello, world!"

- The service named **httpd** is started and enabled.

Slide 37

Lab: Apply the Server Recipe

```
$ sudo chef-apply apache/recipes/server.rb
```

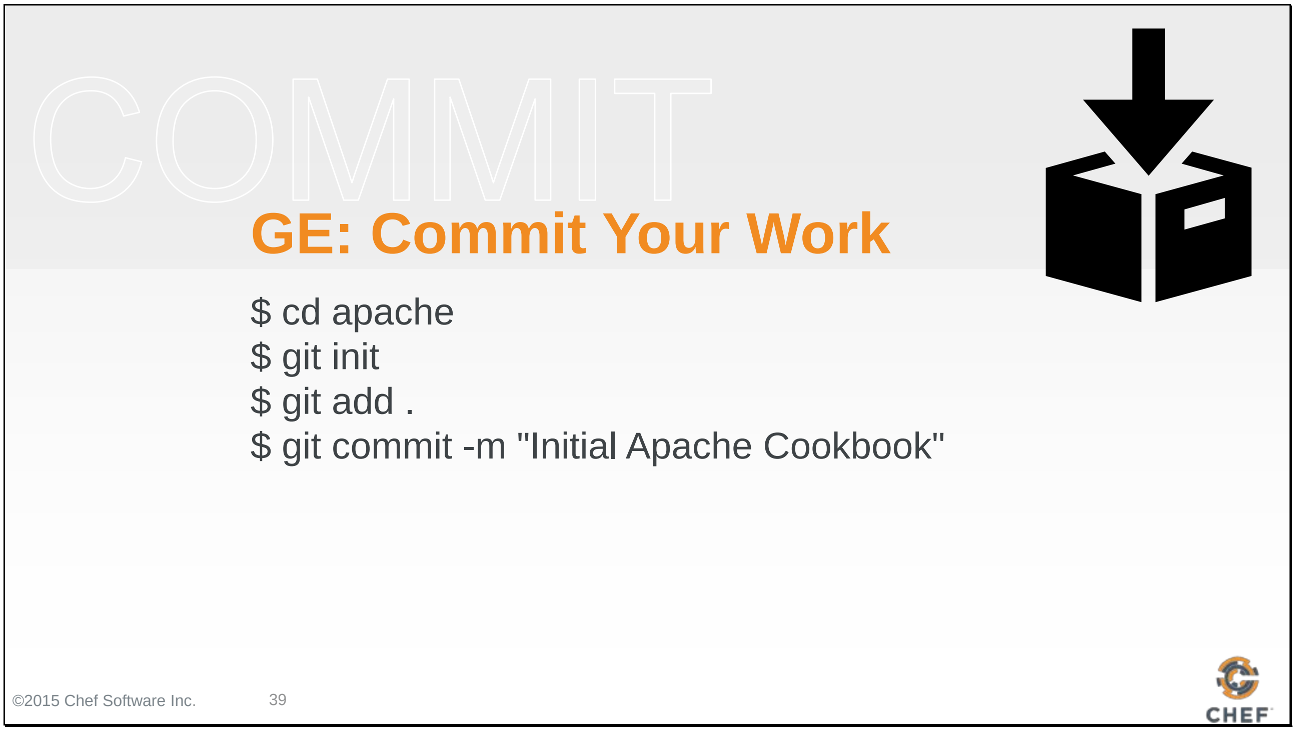
```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * yum_package[httpd] action install
    - install version 2.2.15-47.el6.centos of package httpd
  * file[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to 17d291
    --- /var/www/html/index.html        2015-09-14 22:57:21.151137524 +0000
    +++ /var/www/html/.index.html20150914-2132-n4lsm6   2015-09-14
22:57:21.150137524 +0000
    @@ -1 +1,2 @@
    +<h1>Hello, world!</h1>
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
```


When applying the recipe with '**chef-apply**', you need to specify the partial path to the recipe file within the apache cookbook's recipe folder.

Slide 38

You already setup apache, which is a web server. So verify that the website is available and returns the content we expect to see.

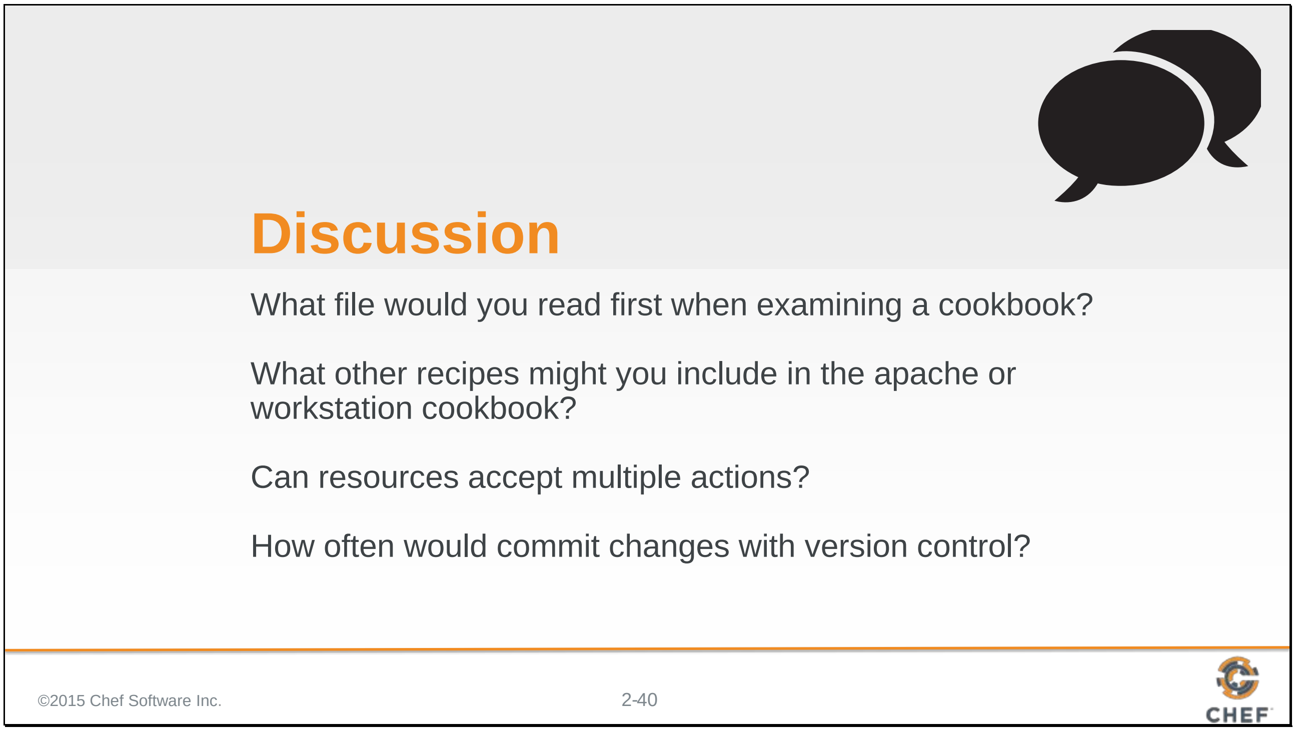
Slide 39



Now, with everything working it is time to add the apache cookbook to version control.

- Move into the apache directory.
- Initialize the cookbook as a git repository.
- Add all the files within the cookbook.
- And commit all the files in the staging area.

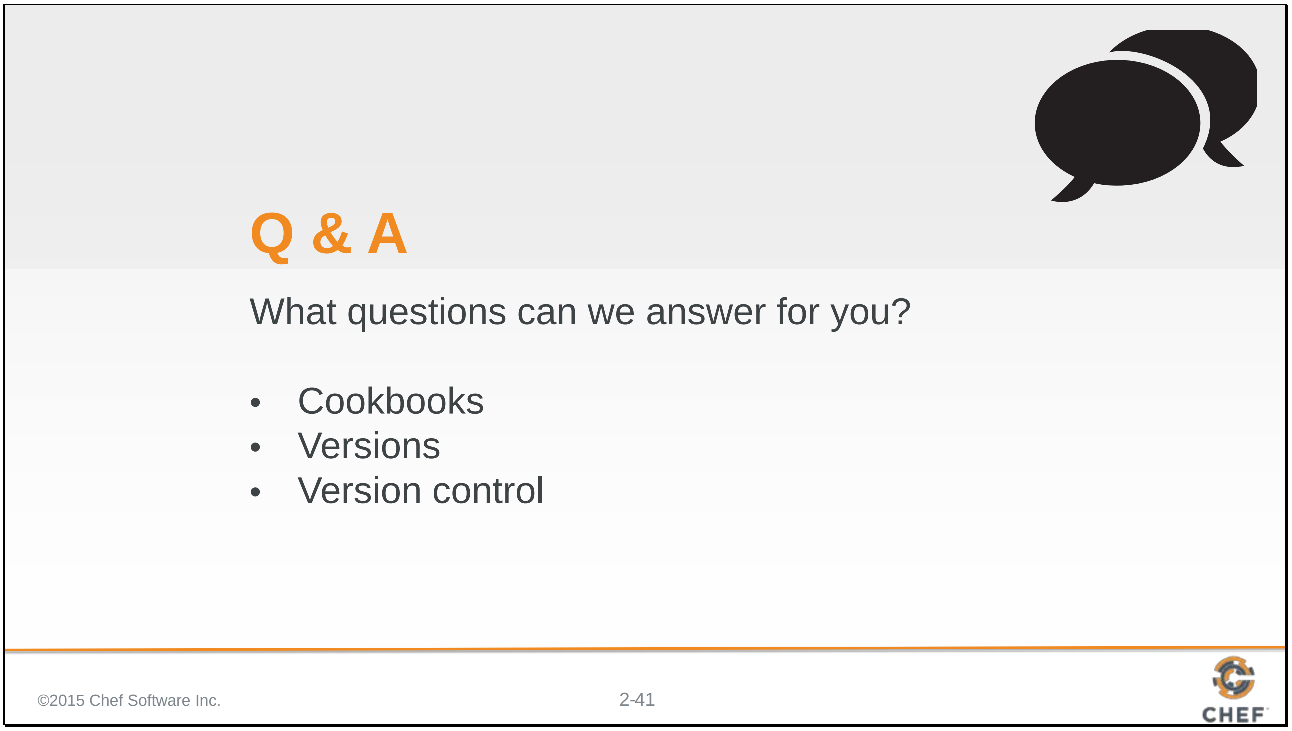
Slide 40



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 41



What questions can we help you answer?

General questions or more specifically about cookbooks, versioning and version control.

Slide 42

4: Chef Client

Slide 2

Objectives

After completing this module, you should be able to use chef-client to:

- Locally apply a cookbook's recipe
- Locally apply multiple cookbooks' recipes
- Include a recipe from within another recipe

In this module you will learn how to use the 'chef-client' command to apply recipes, include a recipe within another recipe and update a cookbook.

Slide 3

'chef-apply' is valuable tool for exploring resources within recipes without having to wrestle with all the folders and files associated with cookbooks. For the remainder of the modules we will not return to using 'chef-apply'.

In the future you will most likely be using 'chef-client'. You may return to 'chef-apply' in your adventures when you find yourself wanting to test out an idea for a new recipe on a new platform or platform version. The speed of the tool is valuable.

Slide 4

CONCEPT

chef-client

chef-client is an agent that runs locally on every node that is under management by Chef.

When a chef-client is run, it will perform all of the steps that are required to bring the node into the expected state.

https://docs.chef.io/chef_client.html

CHEF

In the ChefDK, we package another tool that is called 'chef-client'.

'chef-client' is a command-line application that can be used to apply a recipe or multiple recipes. It also has the ability to communicate with a Chef server – a concept we will talk about in another section. For now think of the Chef Server as a central, artifact repository where we will later store our cookbooks.

Slide 5

Demo: Using 'chef-client' to Locally Apply Recipes

```
$ sudo chef-client --local-mode –r "recipe[workstation::setup]"
```

Applying the following recipes locally:

The 'setup' recipe from the 'workstation' cookbook

Here is an example of using 'chef-client' to locally apply the a run list of recipes. In this case we are applying one recipe and that is the setup recipe within our workstation cookbook.

Slide 6

Demo: Using 'chef-client' to Locally Apply Recipes

```
$ sudo chef-client --local-mode -r "recipe[apache::server]"
```

Applying the following recipes locally:

The 'server' recipe from the 'apache' cookbook

Here is an example of using 'chef-client' to locally apply the server recipe within our apache cookbook.

Slide 7

Demo: Using 'chef-client' to Locally Apply Recipes

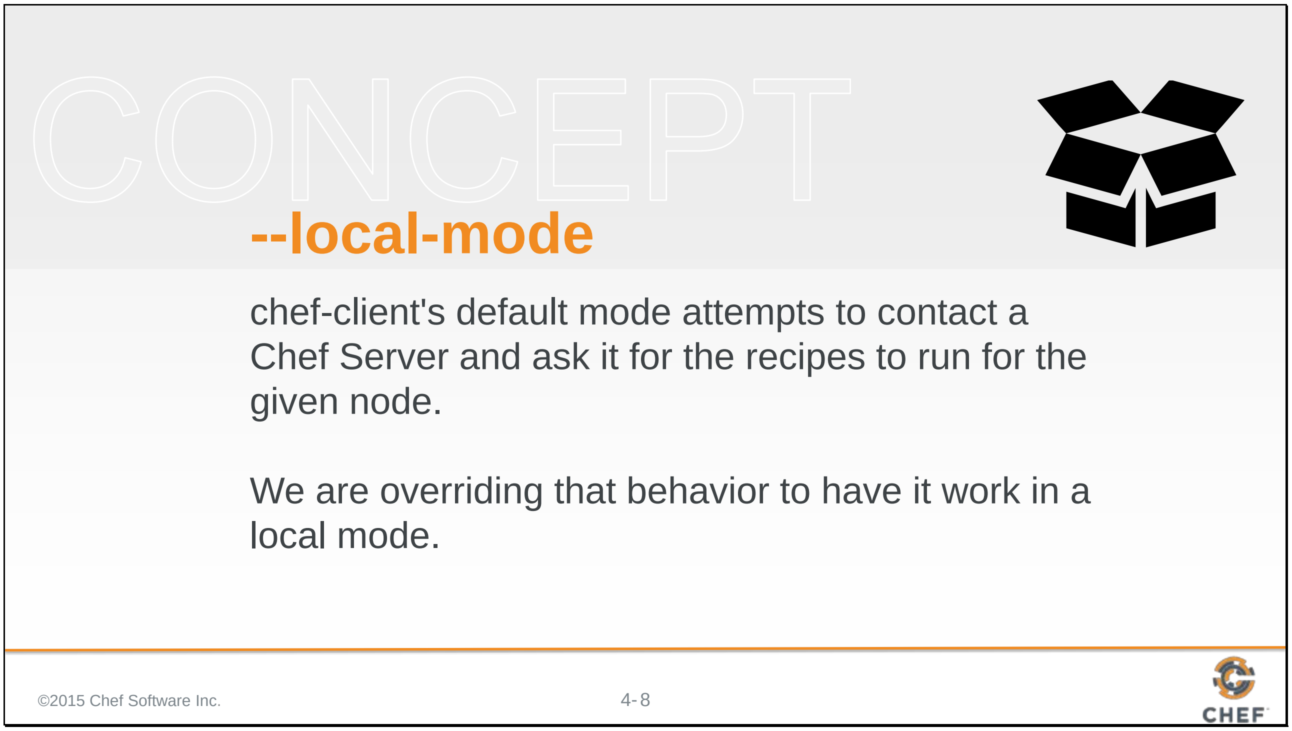
```
$ sudo chef-client --local-mode -r \
"recipe[workstation::setup],recipe[apache::server]"
```

Applying the following recipes locally:

- The 'setup' recipe from the 'workstation' cookbook
- The 'server' recipe from the 'apache' cookbook

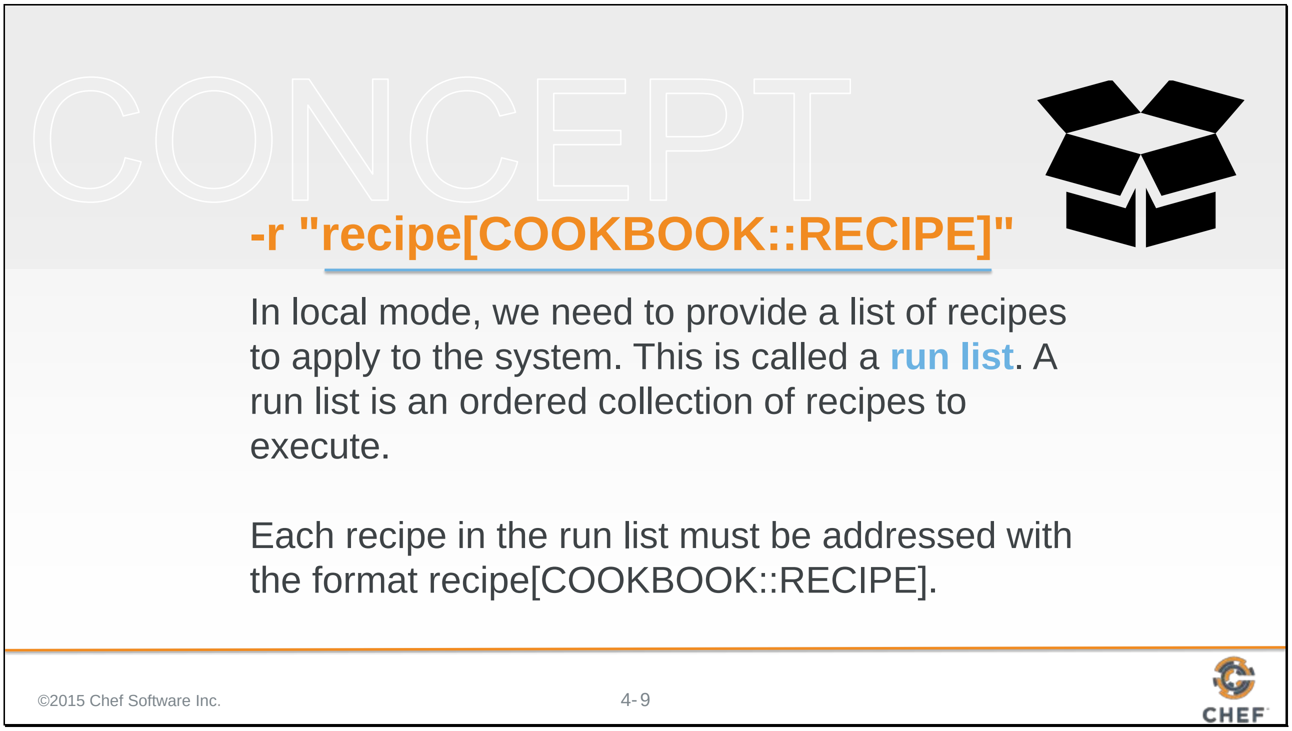
Here is an example of using 'chef-client' to locally apply two recipes -- the setup recipe from the workstation cookbook and the server recipe within our apache cookbook.

Slide 8



Applying recipes with 'chef-client' is different than 'chef-apply' and that is because chef-client's default behavior is to communicate with a Chef server. So we use the '--local-mode' flag to ask 'chef-client' to look for the cookbooks locally.

Slide 9



When we apply a recipe with 'chef-client', we define a run list. This is an ordered list of recipes that we want to apply to the system. When you define a recipe from a cookbook on the run list, there is a particular convention:

"recipe[COOKBOOK::RECIPE]"

COOKBOOK means the name of the Cookbook.
RECIPE means the name of the Recipe without the Ruby file extension.

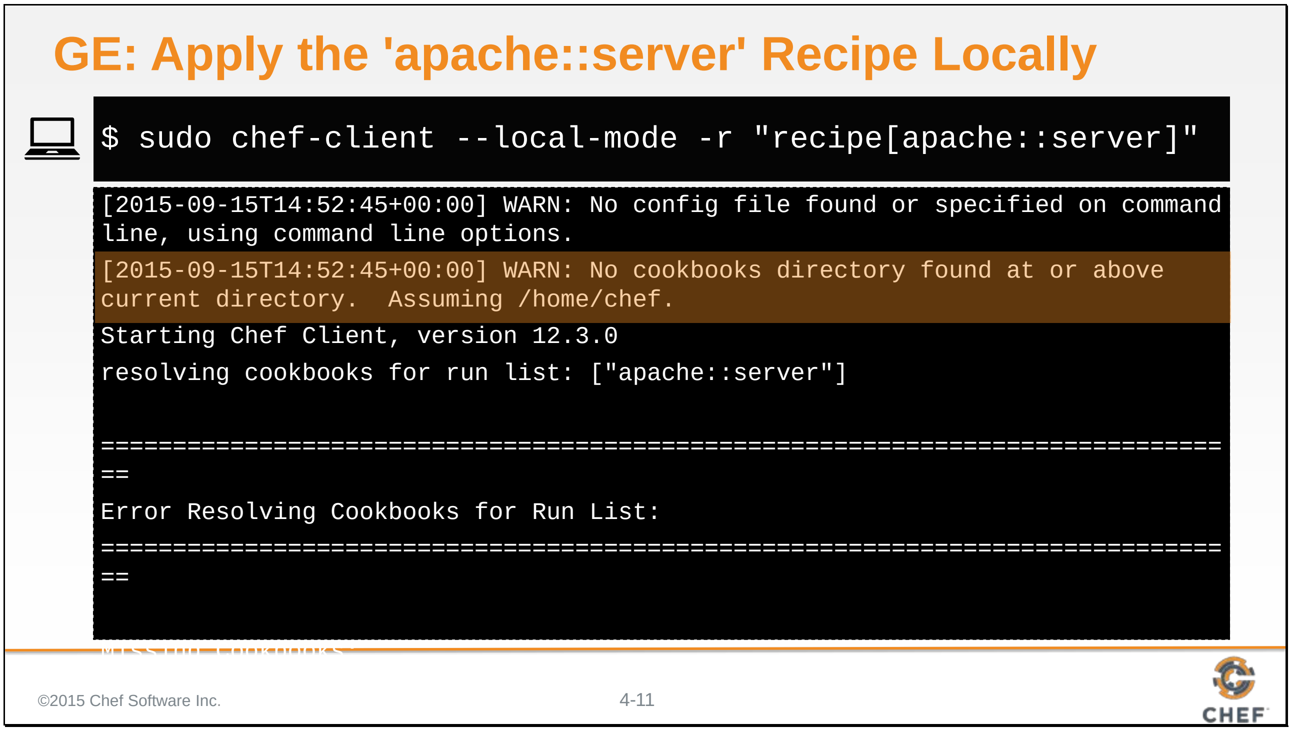
Slide 10

Group Exercise: Return Home First

```
$ cd ~
```


Before you start applying cookbooks through 'chef-client', make sure you are in your home directory.

Slide 11



Try applying our server recipe from the apache cookbook using `chef-client` in local mode.

Upon execution you unfortunately are presented with an error.

When executed we find that `chef-client` has an additional requirement. `chef-client` expects our cookbooks to be maintained in a directory named 'cookbooks'.

That seems simple enough to accommodate and a good way to start organizing the cookbooks that we are creating.

Slide 12

GE: Create Cookbooks Dir and Move the Cookbook

```
$ mkdir cookbooks
$ mv workstation cookbooks
$ mv apache cookbooks
```


Make a directory named 'cookbooks'. Then move the workstation cookbook and apache cookbook into the cookbooks directory.

Slide 13

GE: Apply the Cookbook Recipe Locally

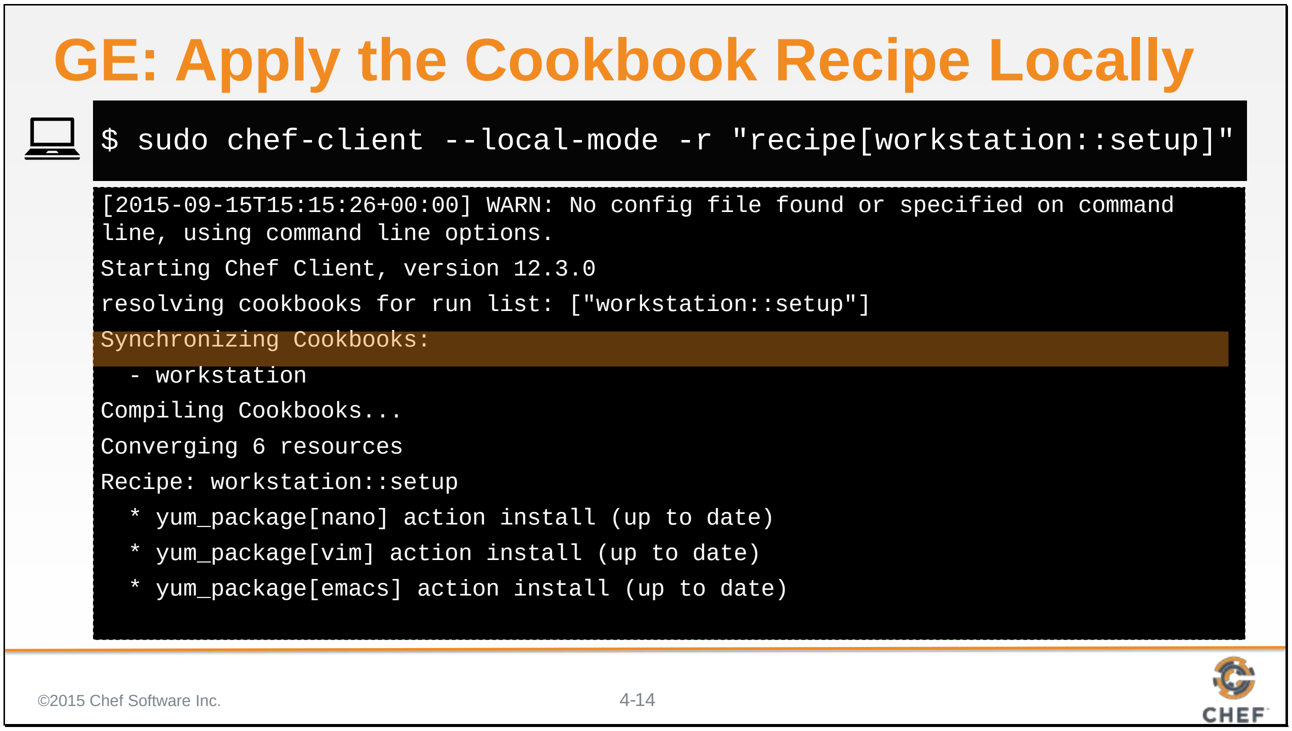
```
$ sudo chef-client --local-mode -r "recipe[apache::server]"
```

```
[2015-09-15T14:54:45+00:00] WARN: No config file found or specified on command
line, using command line options.
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["apache::server"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
Converging 4 resources
Recipe: apache::server
  * yum_package[httpd] action install (up to date)
  * file[/var/www/html/index.html] action create (up to date)
  * service[httpd] action enable (up to date)
```


Lets try that again--this time with all of our cookbooks in the cookbooks directory like `chef-client` expects.

Try applying the apache cookbook's recipe named server.

Slide 14



Try applying the workstation cookbook's recipe named 'setup'.

Slide 15

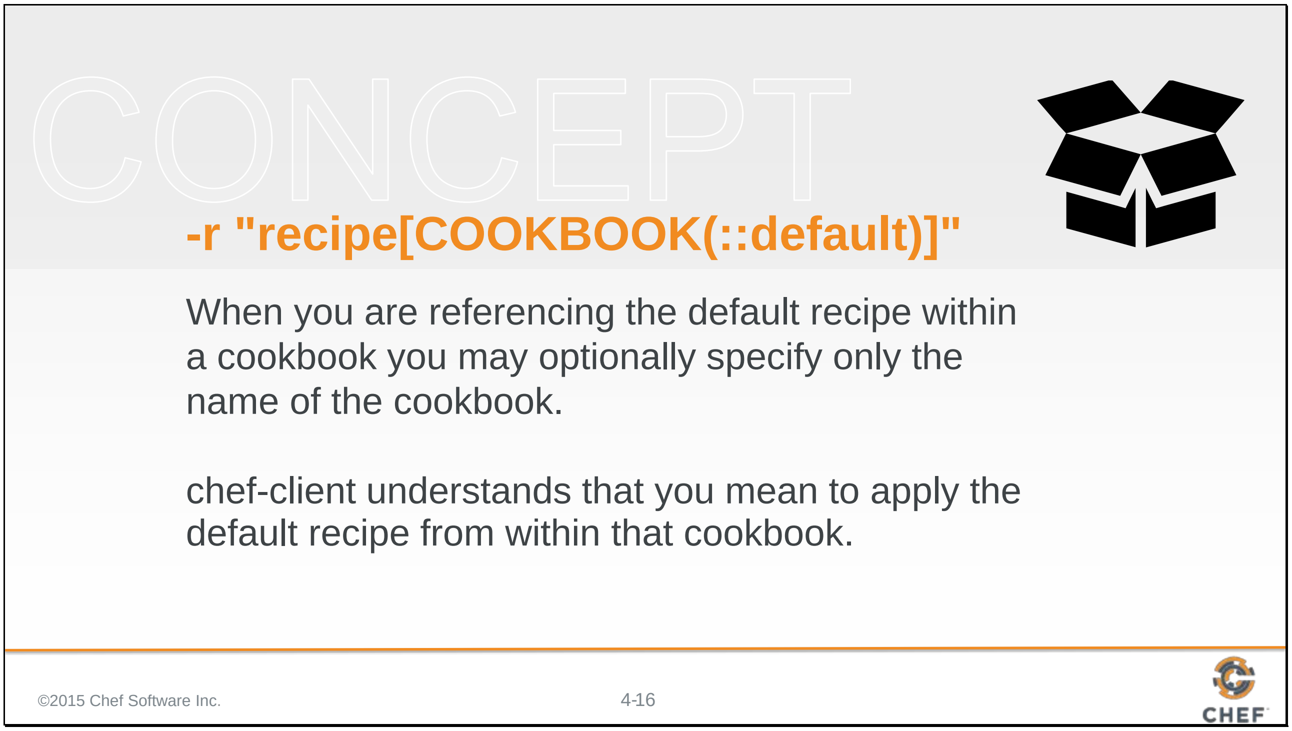
GE: Apply Both Recipes Locally

```
$ sudo chef-client --local-mode \ -r
"recipe[apache::server],recipe[workstation::setup]"
```

```
[2015-09-15T15:17:27+00:00] WARN: No config file found or specified on
command line, using command line options.
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["workstation::setup"]
Synchronizing Cookbooks:
  - workstation
Compiling Cookbooks...

Running handlers:
[2015-09-15T15:17:30+00:00] ERROR: Running exception handlers
Running handlers complete
```


Try applying both recipes from both cookbooks again at one time.

Slide 16



Actually, we didn't tell you everything about specifying the run list for the `chef-client` command.

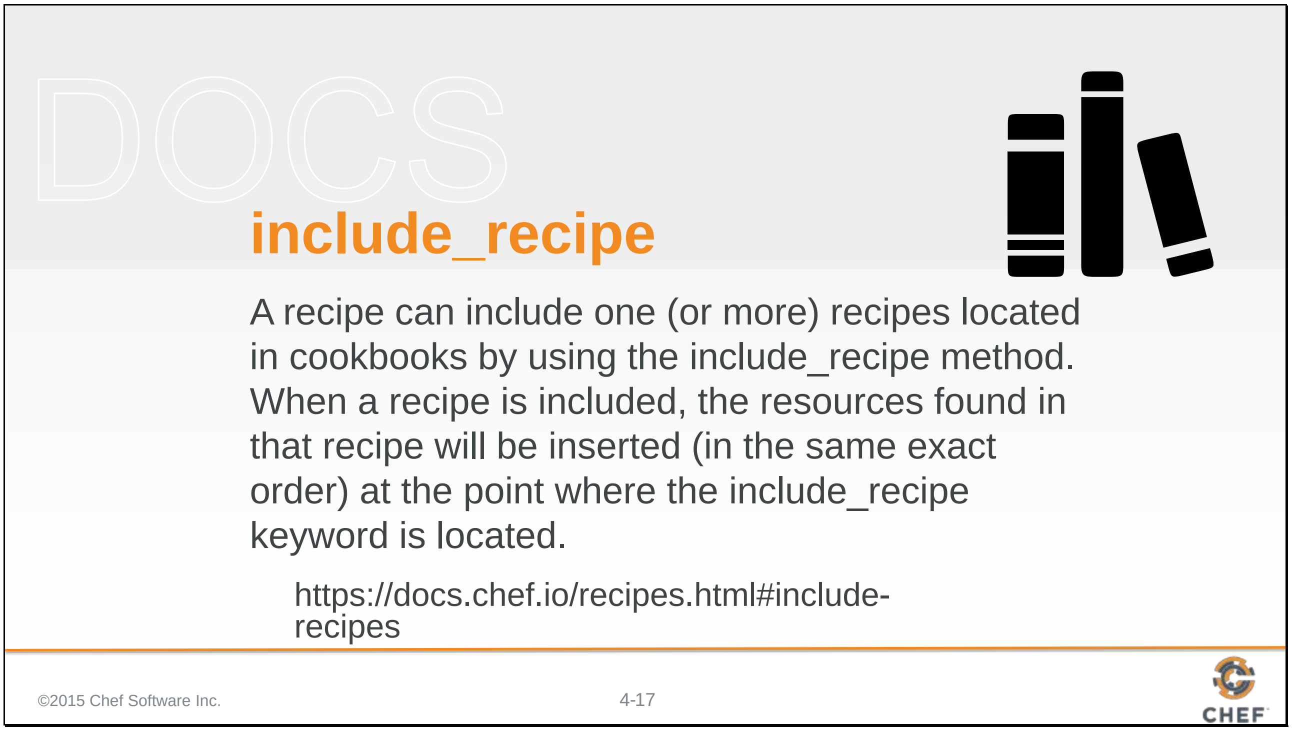
When defining a recipe in the run list you may omit the name of the recipe, and only use the cookbook name, when that recipe's name is 'default'.

Similar to how resources have default actions and default attributes Chef uses the concept of providing sane defaults. This makes our faster when we understand the concepts.

A cookbook doesn't have to have a default recipe but most every cookbook has one. It's called default because when you think of a cookbook it is the recipe that defines the most common configuration policy.

When you think about the two cookbooks that we created -- the apache cookbook with the server recipe and the workstation cookbook with the setup recipe -- it seems like those recipes would be good default recipes for their respective cookbooks.

Slide 17



A simple solution would be to rename the setup recipe to the default recipe. However, a better practice would instead leave our recipes as they are and have the default recipe include the recipes with a method called `include_recipe`

This allows us to maintain all the current policies within its own recipe file and that way we can more easily switch our cookbooks default behavior, which can be useful when new requirements surface.

Slide 18

Demo: Including a Recipe

```
include_recipe "workstation::setup"
```

Include the "setup" recipe from the "workstation" cookbook in this recipe

In this example we are including the "workstation" cookbook's "setup" recipe.

Slide 19

Demo: Including a Recipe

```
include_recipe "apache::server"
```

Include the "server" recipe from the "apache" cookbook in this recipe

In this example, we are including the "apache" cookbook's "server" recipe.

Slide 20

GE: The Default Recipe Includes the Setup Recipe

~/cookbooks/workstation/recipes/default.rb

```
#
# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights
Reserved.

include_recipe "workstation::setup"
```


We are interested in having the default recipe for our workstation cookbook run the contents of the setup recipe.

Within the default recipe, define the `include_recipe` method and provide one parameter, which is the name of our recipe as it appears within a run list: cookbook_name::recipe_name.

Slide 21

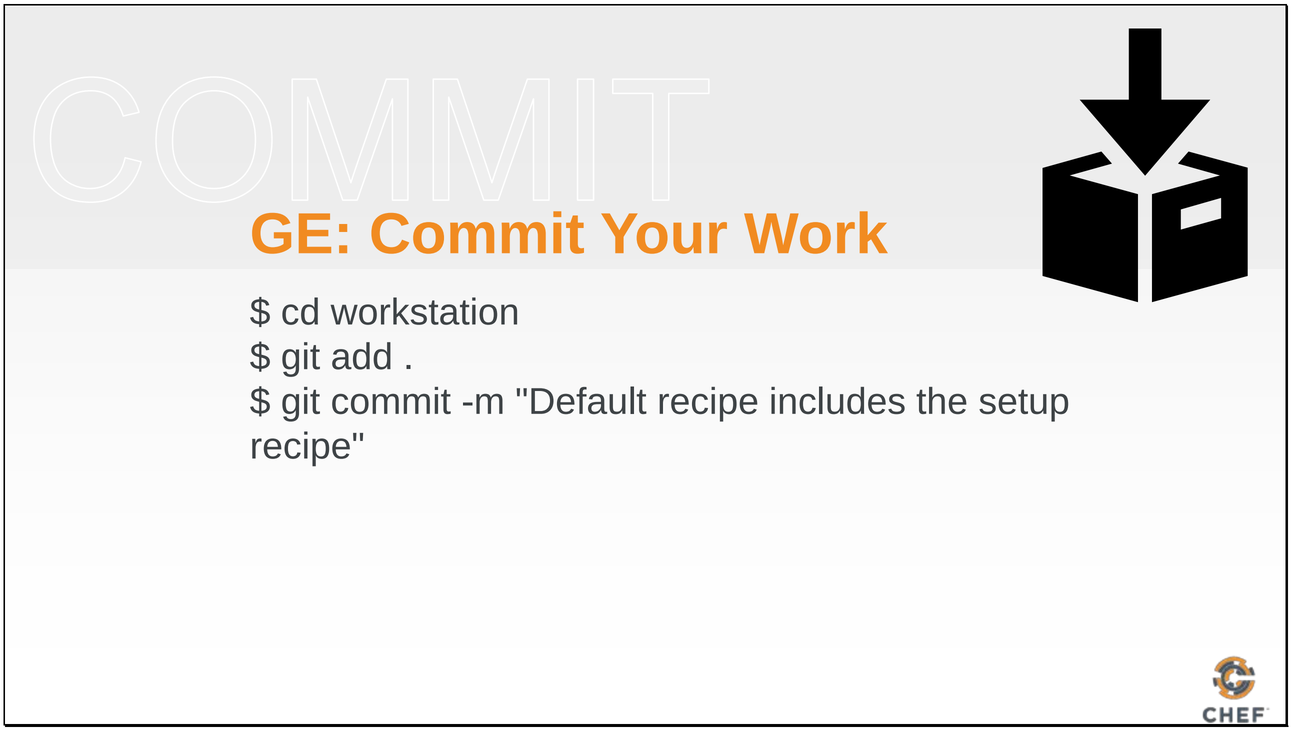
GE: Apply the Cookbook's Default Recipe

```
$ sudo chef-client --local-mode -r "recipe[workstation]"
```

```
 WARN: No config file found or specified on command line, using command line
options.
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["workstation"]
Synchronizing Cookbooks:
  - workstation
Compiling Cookbooks...
Converging 0 resources

Running handlers:
Running handlers complete
Chef Client finished, 0/0 resources updated in 3.300489827 seconds
```


Use 'chef-client' to locally apply the cookbook named workstation. This will load your workstation cookbook's default recipe, which in turn loads the workstation cookbook's setup recipe.

Slide 22



With everything working it is time to commit the latest changes.

Slide 23

In this lab you will update the apache cookbook's default recipe to include the apache cookbook's recipe named server.

Slide 24

Lab: The Default Recipe Includes the Apache Recipe

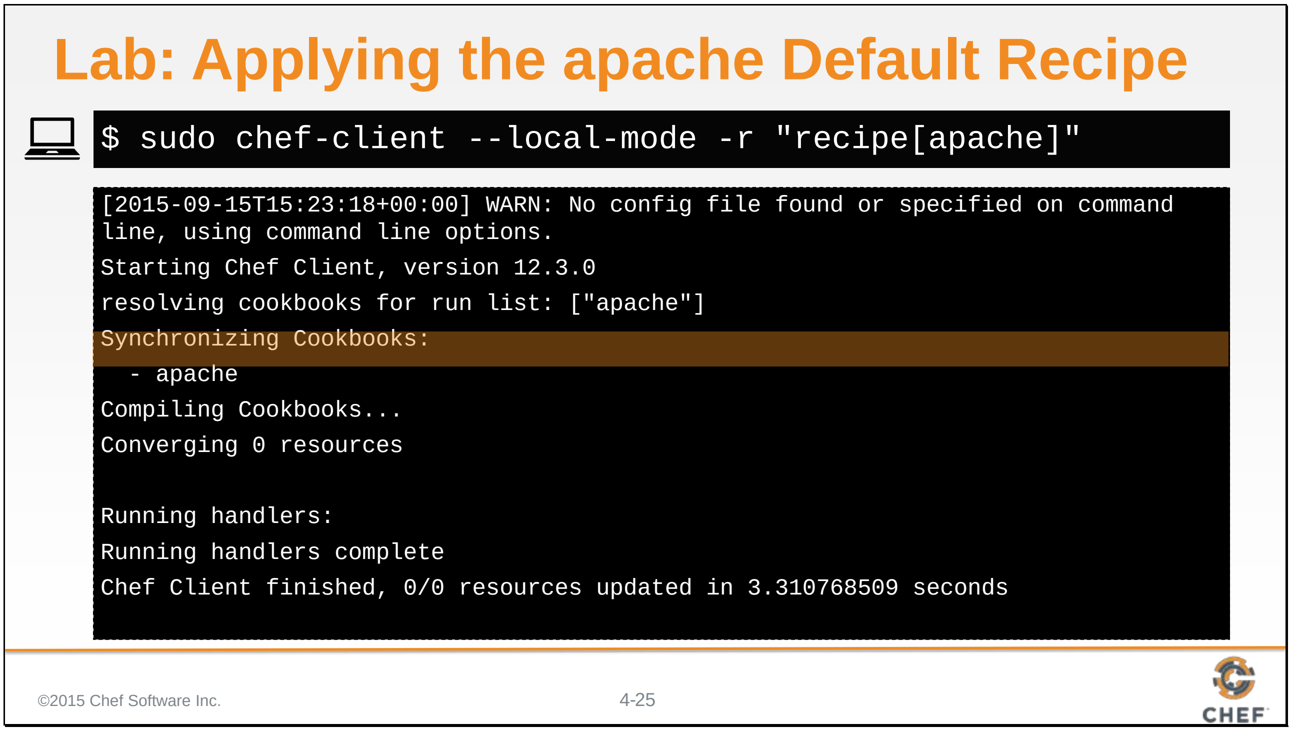
~/cookbooks/apache/recipes/default.rb

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights
Reserved.

include_recipe "apache::server"
```


We are interested in having the default recipe for our apache cookbook run the contents of the server recipe.

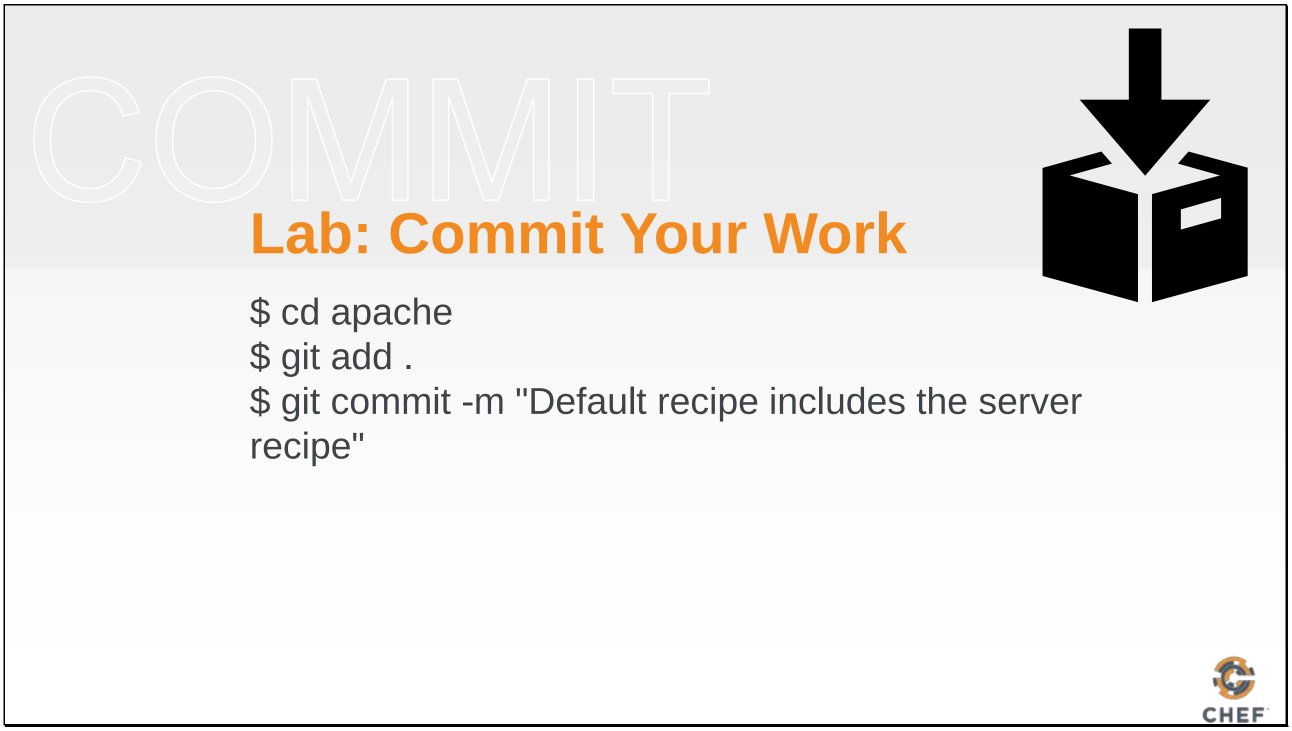
Within the default recipe, define the `include_recipe` method and provide one parameter, which is the name of our recipe as it appears within a run list: cookbook_name::recipe_name.

Slide 25



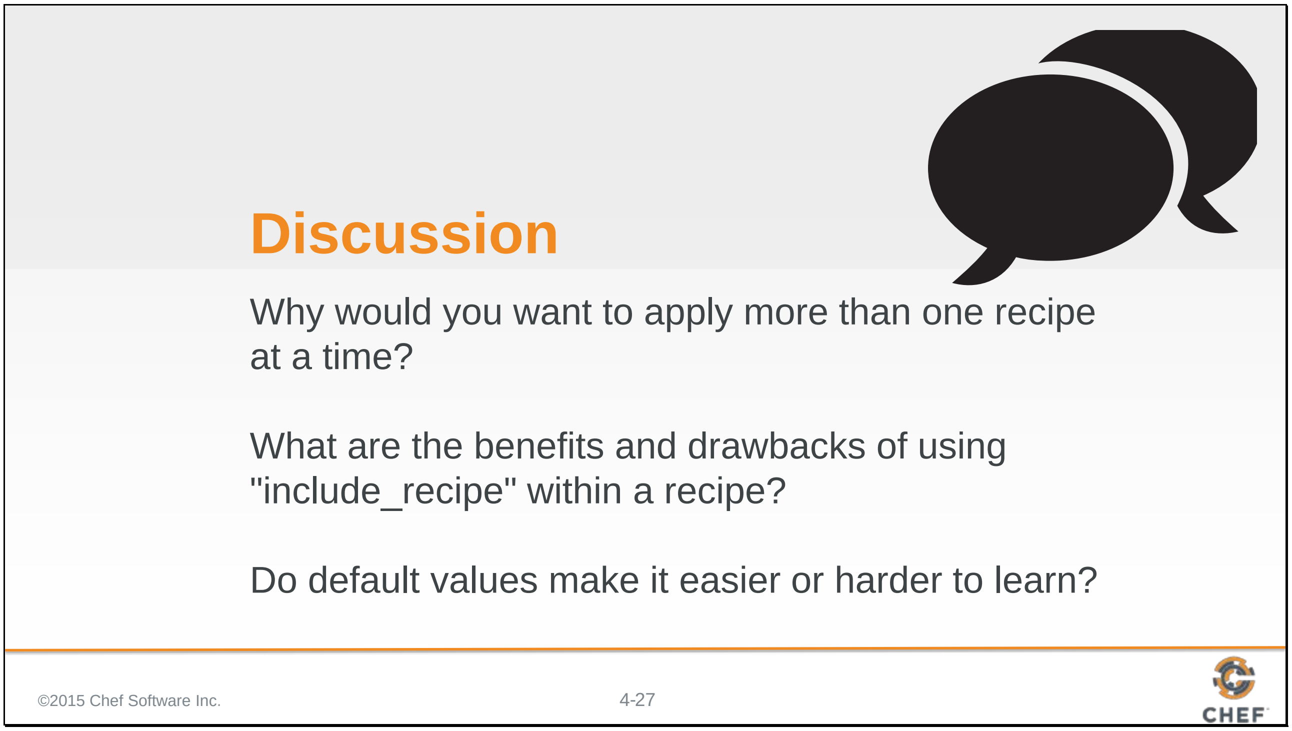
Use 'chef-client' to locally apply the cookbook named apache. This will load your apache cookbook's default recipe, which in turn loads the apache cookbook's server recipe.

Slide 26



With everything working it is time to commit the latest changes.

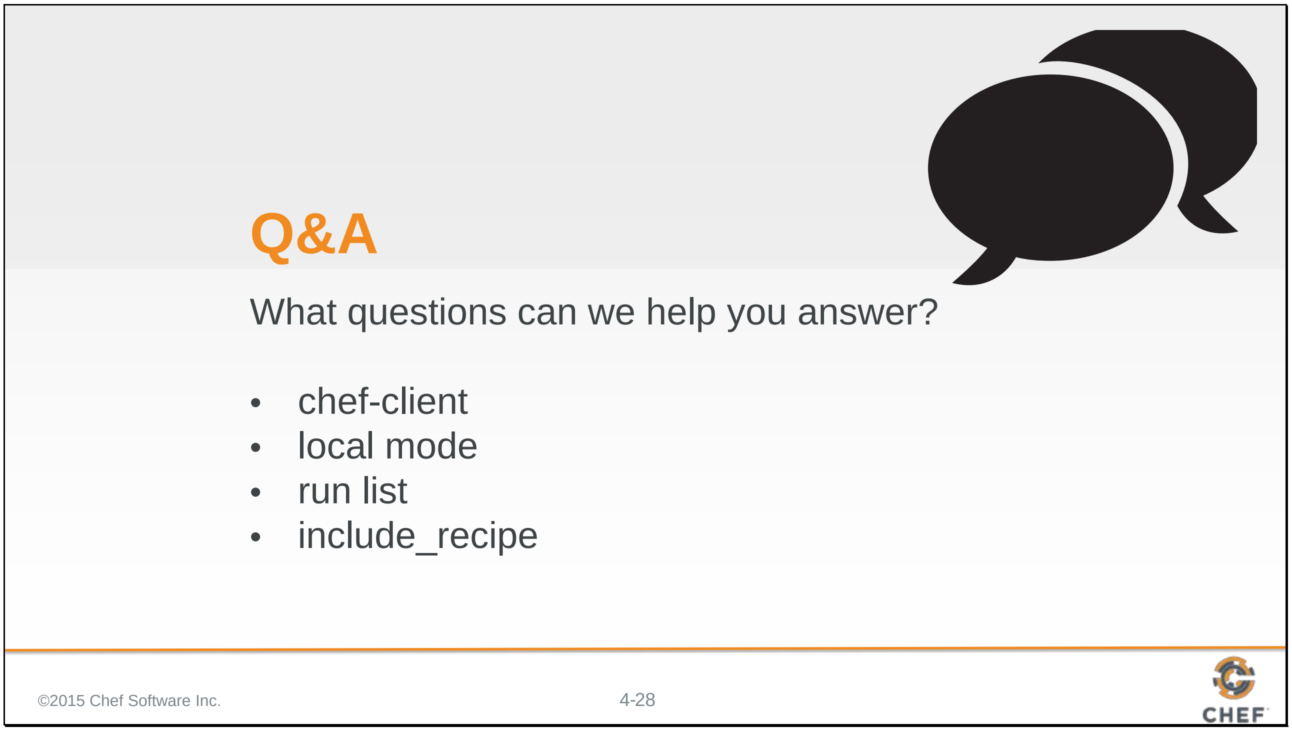
Slide 27



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 28



What questions can we help you answer?

Generally or specifically about chef-client, local mode, run lists, and include_recipe.

Slide 29

CHEF™

5: Testing Cookbooks

Slide 2

Objectives

After completing this module, you should be able to

- Use Test Kitchen to verify your recipes converge on a virtual instance
- Read the ServerSpec documentation
- Write and execute tests

In this module you will learn how to use the Test Kitchen tool to execute your configured code, write and execute tests, and use Serverspec to test your servers' actual state.

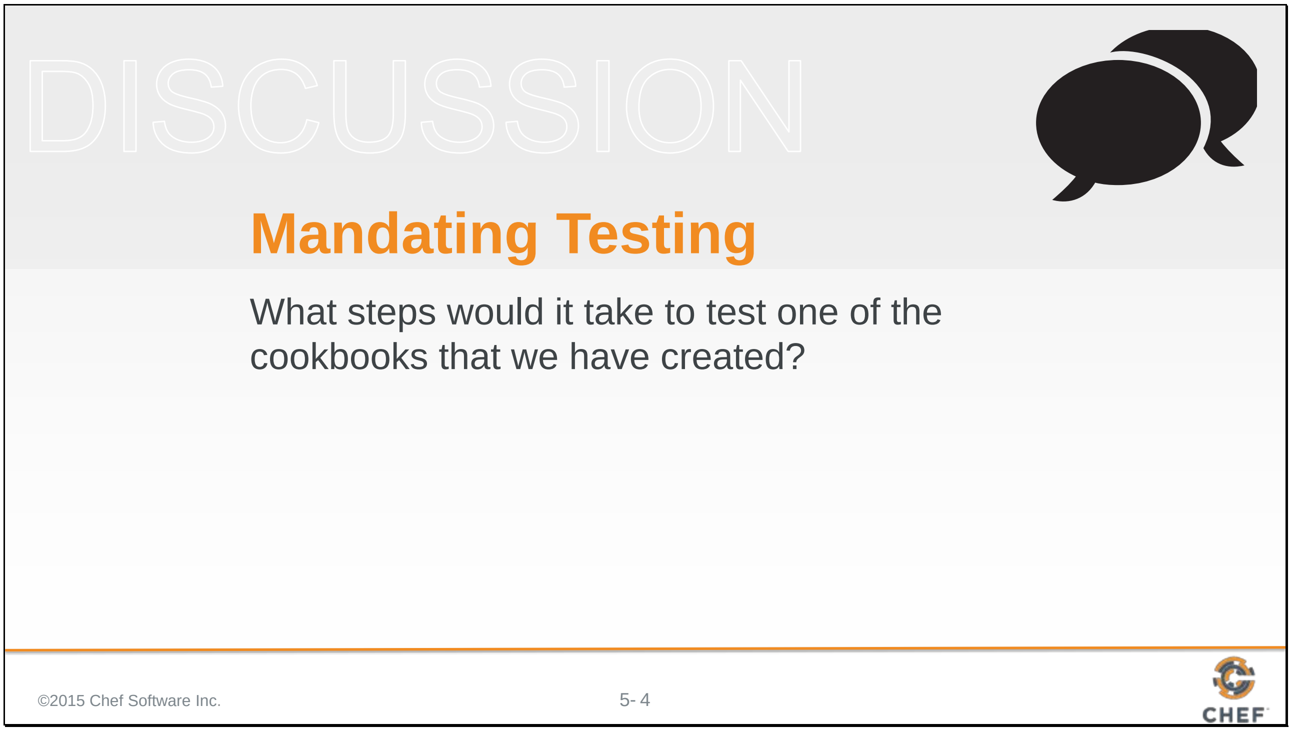
Slide 3

Will the recipes that we created work on another system similar to this one? Will they work in production?

When we develop our automation we need to start thinking about verifying it. Because it is all too common a story of automation failing when it reaches production because it was never validated against anything other than "my machine".

So how could we solve a problem like this?

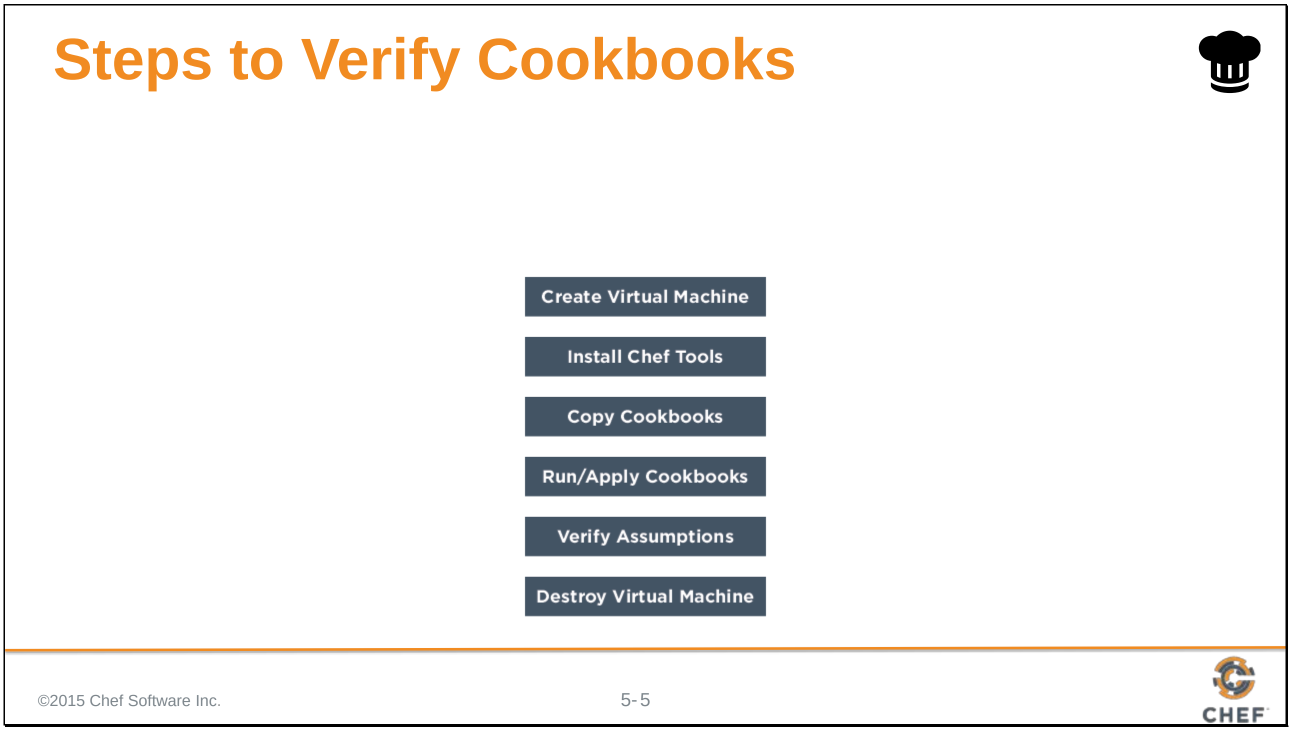
Slide 4



Write down or type out as many of the steps you can think of required to test one of the cookbooks.

When you are ready turn to another person and compare your lists. Create a complete list with all the steps that you have identified. Then as a group we will discuss all the steps necessary to test a cookbook.

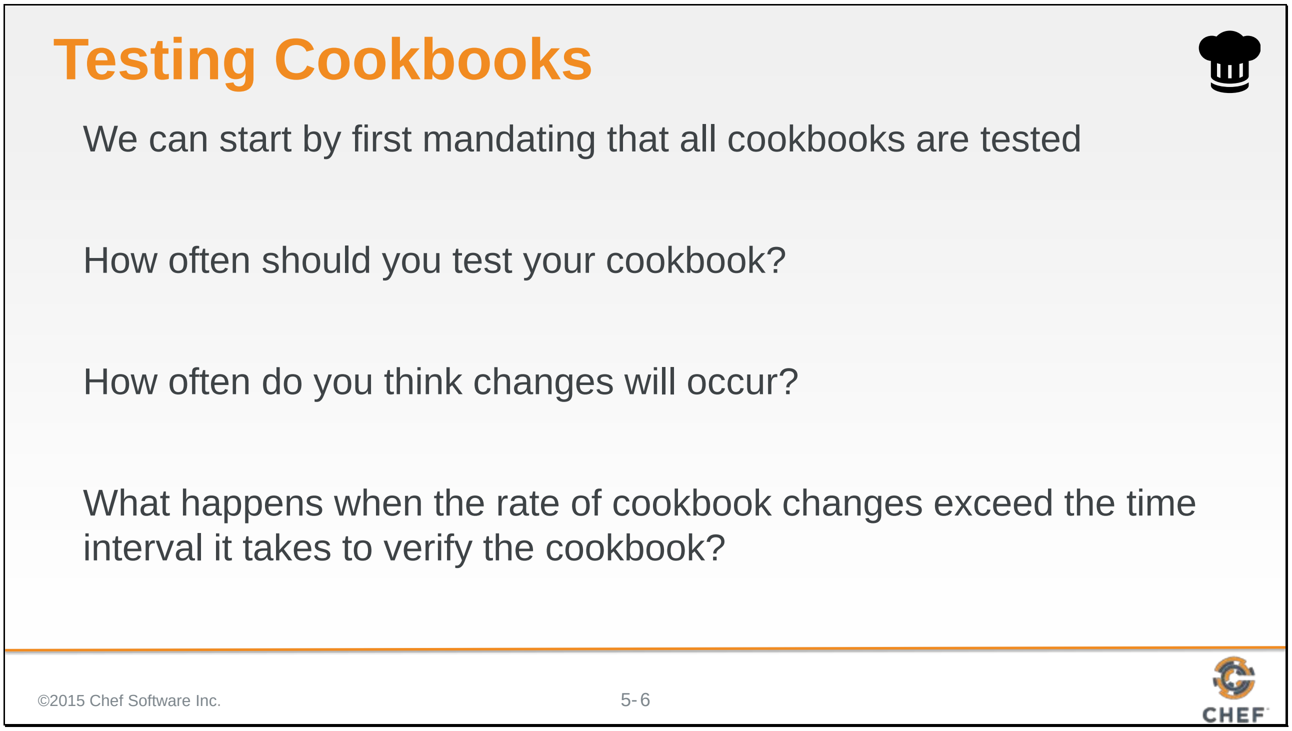
Slide 5



Here are the steps necessary to verify one of the cookbooks that you created.

- Create a virtual machine or setup an instance that resembles your current production infrastructure
- Install the necessary Chef tools
- Copy the cookbooks to this new instance
- Apply the cookbooks to the instance
- Verify that the instance is the desired state by executing various commands
- Clean up that instance by destroying it or rolling it back to a previous snapshot

Slide 6

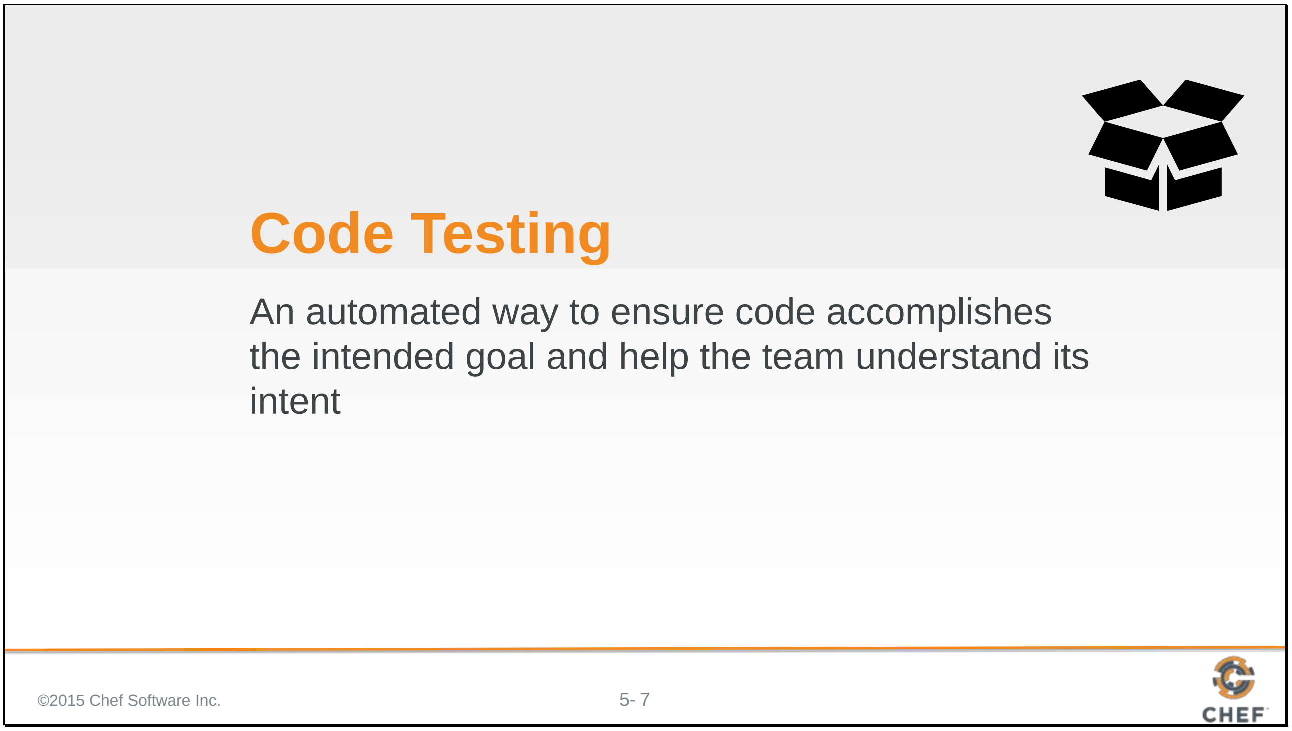


So we can start by mandating that all cookbooks are tested.

But we need to consider how often we need to test a cookbook and how often changes to our cookbooks will occur.

And what would happen if the rate of rate of cookbook changes exceed the time interval it takes to verify the cookbook?

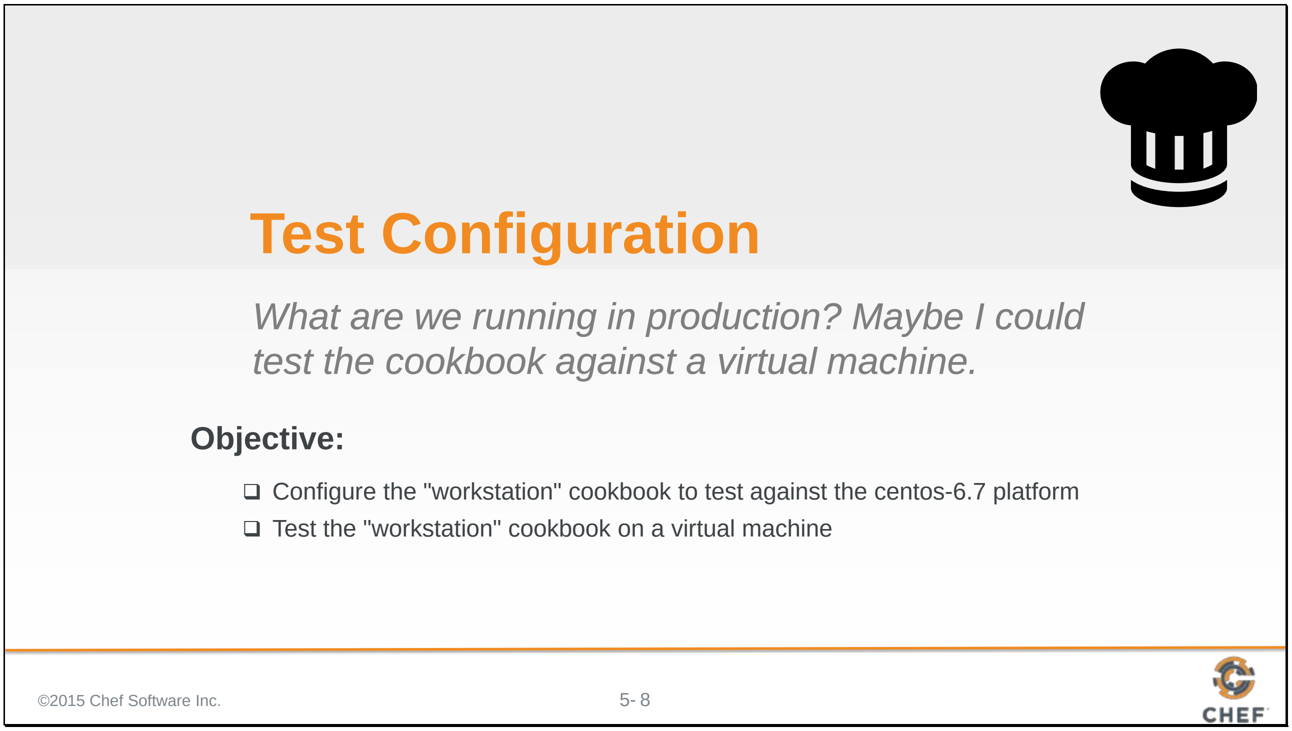
Slide 7



Testing tools provide automated ways to ensure that the code we write accomplishes its intended goal. It also helps us understand the intent of our code by providing executable documentation. We add new cookbook features and write tests to preserve this functionality.

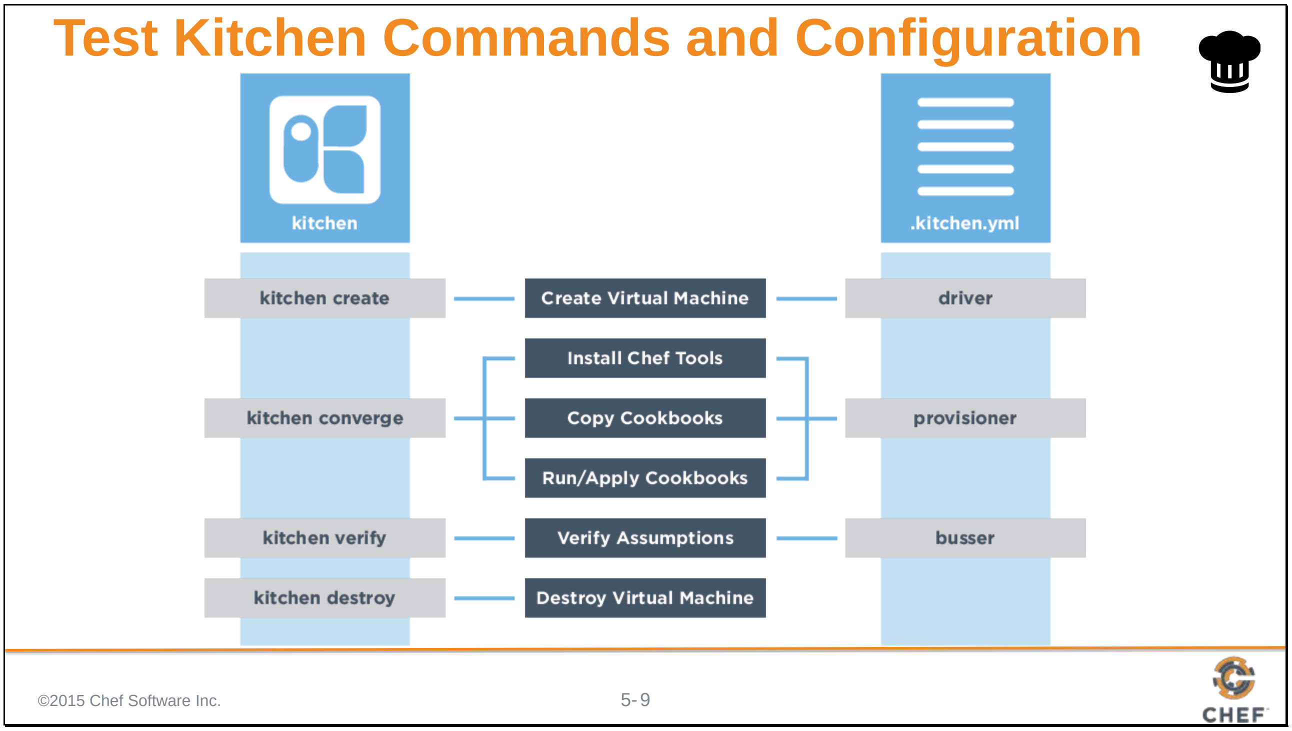
This provides us, or anyone else on the team, the ability to make new changes with a less likely chance of breaking something. Whether returning to the cookbook code tomorrow or in six months.

Slide 8



Well if Chef is to replace our existing tools, it is going to need to provide a way to make testing the policies more delightful.

Slide 9



Test Kitchen allows us to create an instance solely for testing, installs Chef, converge a run list of recipes on that instance, verify that the instance is in the desired state, and then destroy the instance.

On the left are the commands by the kitchen command that map to the stages of the testing lifecycle.

On the right are the configuration fields within a kitchen configuration file.

These commands the configuration will be explained in more detail.

Slide 10

What Can 'kitchen' Do?

```
$ kitchen --help
```

```
Commands:
  kitchen console                        # Kitchen Console!
  kitchen converge [INSTANCE|REGEXP|all] # Converge one or more instances
  kitchen create [INSTANCE|REGEXP|all]   # Create one or more instances
  kitchen destroy [INSTANCE|REGEXP|all]  # Destroy one or more instances
  ...
  kitchen help [COMMAND]                 # Describe available commands or one specif...
  kitchen init                           # Adds some configuration to your cookbook...
  kitchen list [INSTANCE|REGEXP|all]     # Lists one or more instances
  kitchen setup [INSTANCE|REGEXP|all]    # Setup one or more instances
  kitchen test [INSTANCE|REGEXP|all]     # Test one or more instances
  kitchen verify [INSTANCE|REGEXP|all]   # Verify one or more instances
  kitchen version                        # Print Kitchen's version information
```


CHEF

Kitchen is a command-line application that enables us to manage the testing lifecycle.

Similar to other tools within the ChefDK, we can ask for help to see the available commands.

The `init` command, by its name, seems like a good place to get started.

Slide 11

What Can 'kitchen init' Do?

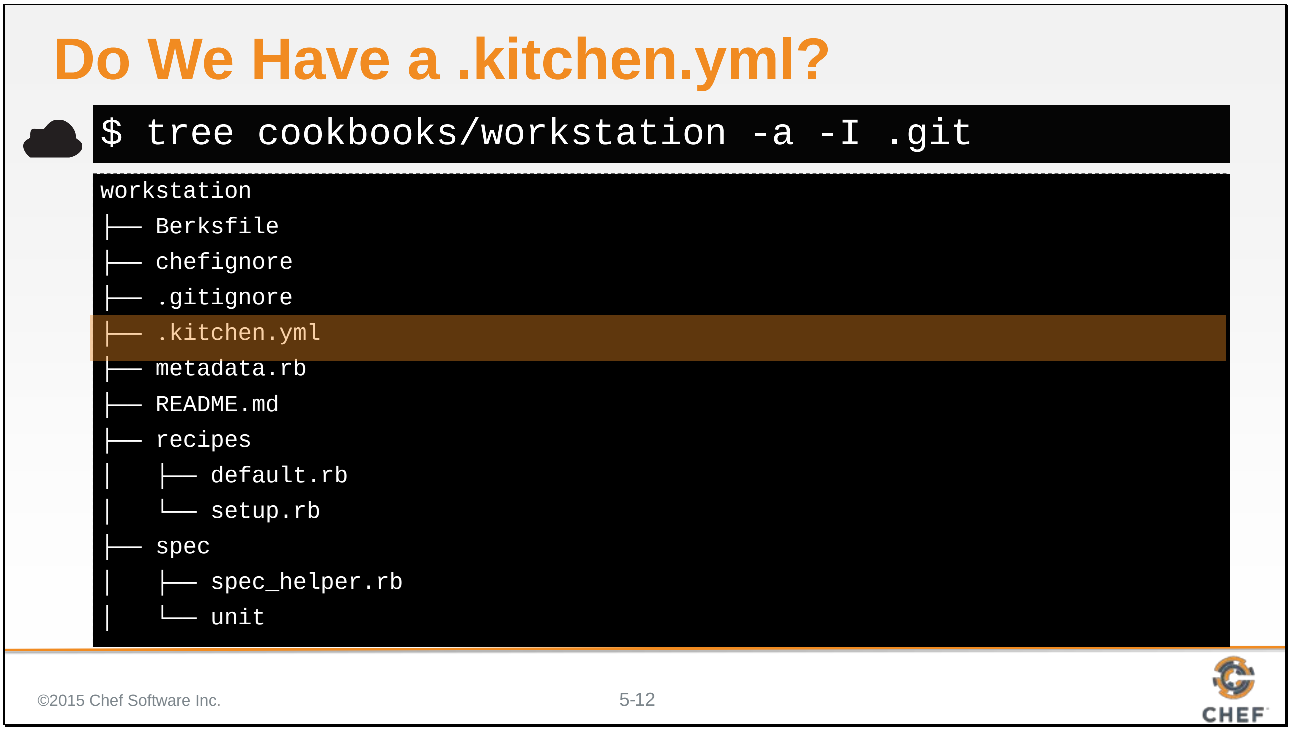
```
$ kitchen help init
```

```
Usage:
  kitchen init
  -D, [--driver=one two three]                   # One or more Kitchen Driver gems ...
                                                 # Default: kitchen-vagrant
  -P, [--provisioner=PROVISIONER]                # The default Kitchen Provisioner to
use
                                                 # Default: chef_solo
      [--create-gemfile], [--no-create-gemfile]  # Whether or not to create a Gemfi ...

Description:
  Init will add Test Kitchen support to an existing project for convergence
  integration testing. A default .kitchen.yml file (which is intended to be
  customized) is created in the project's root directory and one or more gems will be
  added to the project's Gemfile.
```


`kitchen help init` tells us that it will add Test Kitchen support to an existing project. It creates a .kitchen.yml file within the project's root directory.

There are a number of flags and other options but let's see if the cookbooks we created even needs us to initialize test kitchen.

Slide 12



Using `tree` to look at the workstation cookbook, showing all hidden files and ignoring all git files, it looks like our cookbook already has a .kitchen.yml.

It was actually created alongside the other files when we ran the `chef generate cookbook` command when we originally created this cookbook.

Let's take a look at the contents of this file.

Slide 13

What is Inside .kitchen.yml?

```
$ cat cookbooks/workstation/.kitchen.yml
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

platforms:
  - name: ubuntu-12.04
  - name: centos-6.4

suites:
  - name: default
```


5-13
CHEF

The .kitchen.yml file defines a number of configuration entries that the kitchen command uses during execution.

Slide 14

We don't need to run `kitchen init` because we already have a default kitchen file. We may still need to update it to accomplish our objectives so lets learn more about the various fields in the configuration file.

Slide 15

Demo: The kitchen Driver

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

platforms:
  - name: ubuntu-12.04
  - name: centos-6.5

...
```

The driver is responsible for creating a machine that we'll use to test our cookbook.

Example Drivers:

- docker
- vagrant

CHEF

The first key is driver, which has a single key-value pair that specifies the name of the driver Kitchen will use when executed.

The driver is responsible for creating the instance that we will use to test our cookbook. There are lots of different drivers available--two very popular ones are the docker and vagrant driver.

Slide 16

Demo: The kitchen Provisioner

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

platforms:
  - name: ubuntu-12.04
  - name: centos-6.5

...
```

This tells Test Kitchen how to run Chef, to apply the code in our cookbook to the machine under test.

The default and simplest approach is to use chef_zero.

The second key is provisioner, which also has a single key-value pair which is the name of the provisioner Kitchen will use when executed. This provisioner is responsible for how it applies code to the instance that the driver created. Here the default value is chef_zero.

Slide 17

Demo: The kitchen Platforms

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

platforms:
  - name: ubuntu-12.04
  - name: centos-6.5

...
```

This is a list of operation systems on which we want to run our code.

The third key is platforms, which contains a list of all the platforms that Kitchen will test against when executed. This should be a list of all the platforms that you want your cookbook to support.

Slide 18

Demo: The kitchen Suites

~/cookbooks/workstation/.kitchen.yml

```
...

suites:
  - name: default
    run_list:
      - recipe[workstation::default]
    attributes:
```

This section defines what we want to test. It includes the Chef run-list of recipes that we want to test.

We define a single suite named "default".

The fourth key is suites, which contains a list of all the test suites that Kitchen will test against when executed. Each suite usually defines a unique combination of run lists that exercise all the recipes within a cookbook.

In this example, this suite is named 'default'.

Slide 19

Demo: The kitchen Suites

~/cookbooks/workstation/.kitchen.yml

```
...

suites:
  - name: default
    run_list:
      - recipe[workstation::default]
    attributes:
```

The suite named "default" defines a run_list.

Run the "workstation" cookbook's "default" recipe file.

This default suite will execute the run list containing: The workstation cookbook's default recipe.

Slide 20

Kitchen Test Matrix

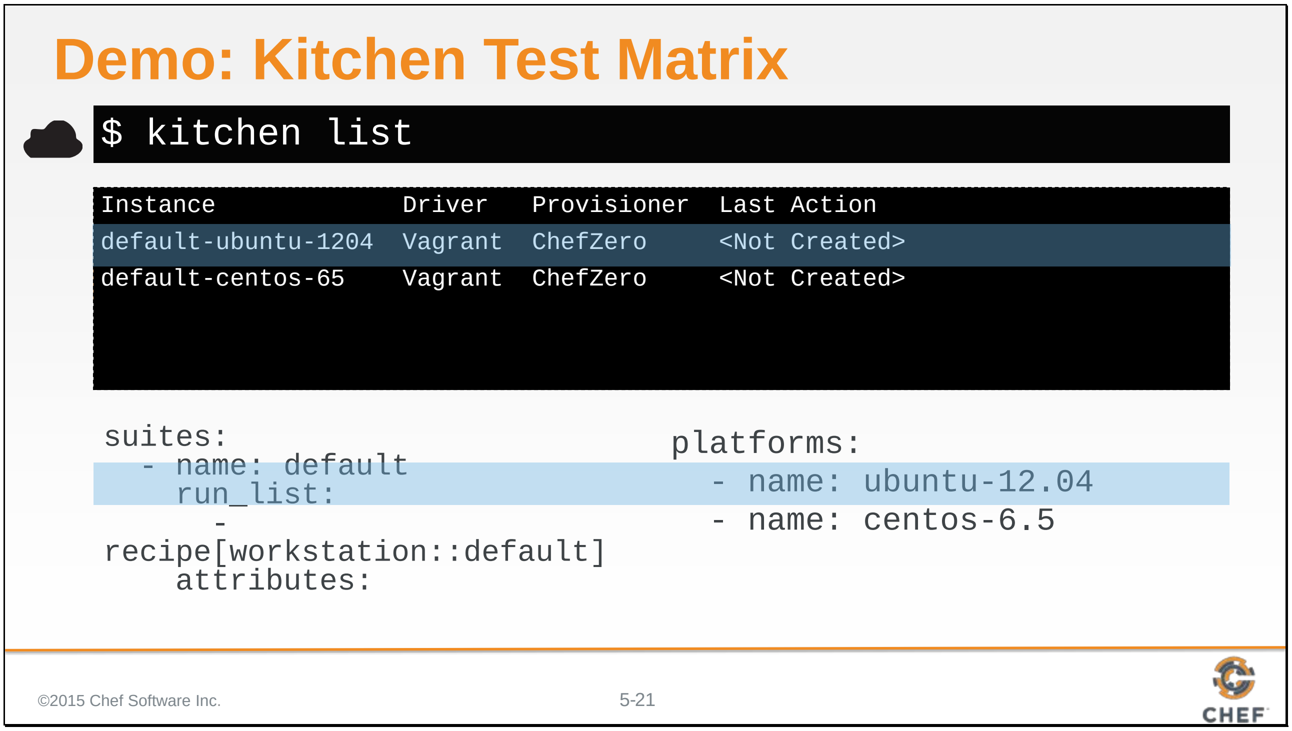
Kitchen defines a list of instances, or test matrix, based on the platforms multiplied by the suites.

PLATFORMS x SUITES

Running kitchen list will show that matrix.

It is important to recognize that within the .kitchen.yml file we defined two fields that create a test matrix; The number of platforms we want to support multiplied by the number of test suites that we defined.

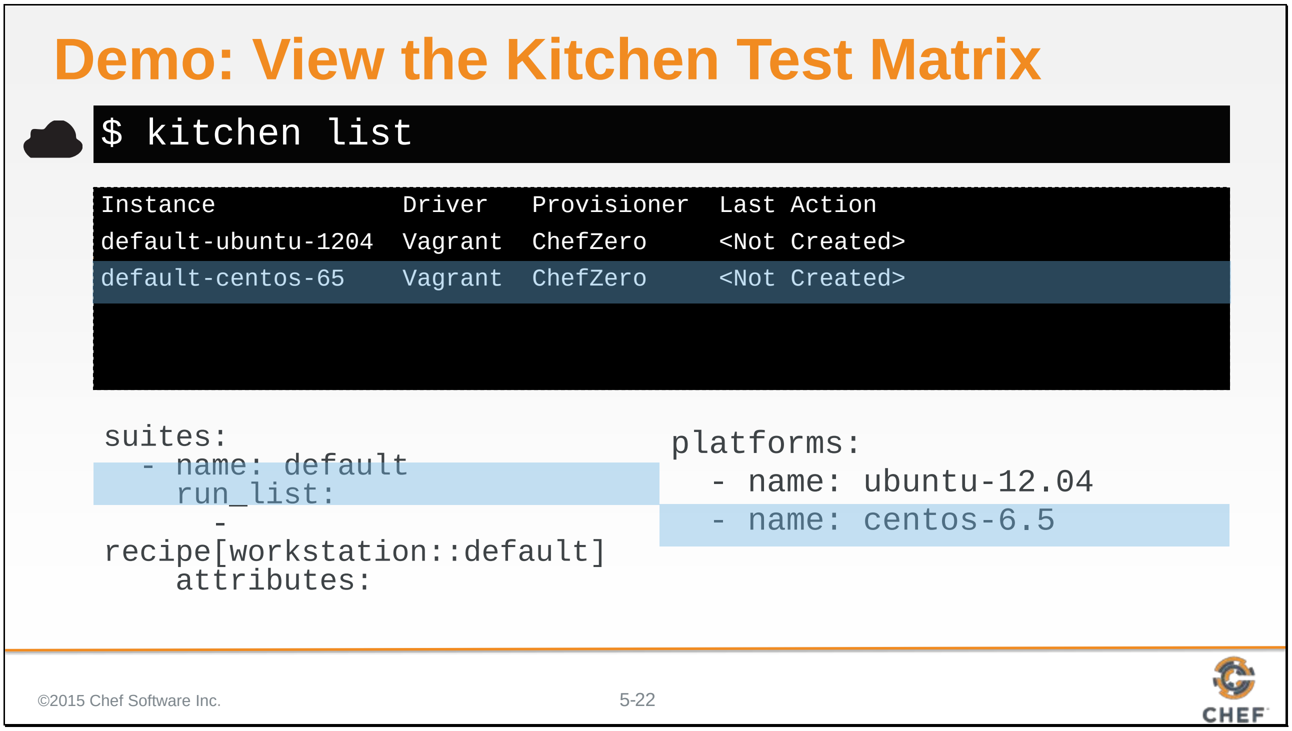
Slide 21



We can visualize this test matrix by running the command `kitchen list`.

In the output you can see that an instance is created in the list for every suite and every platform. In our current file we have one suite, named 'default', and two platforms. First the ubuntu 12.04 platform.

Slide 22

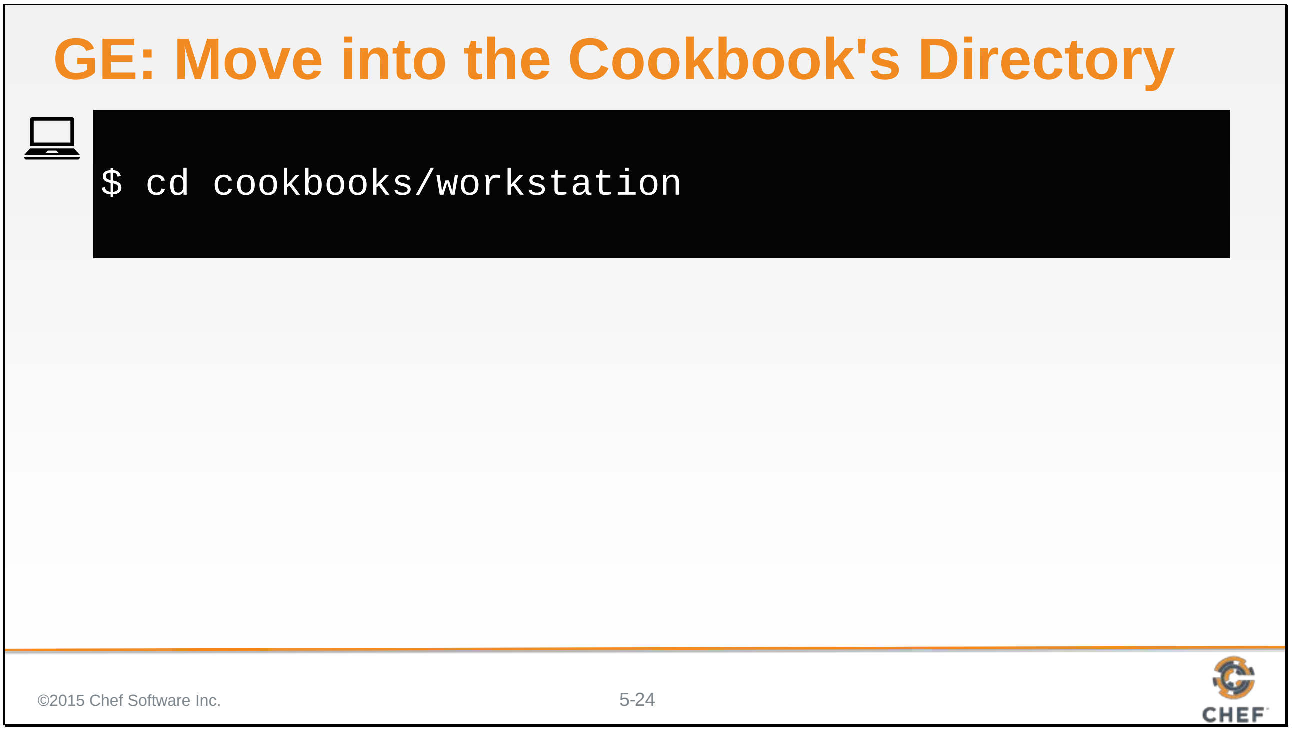


And the second centos 6.5 platform.

Slide 23

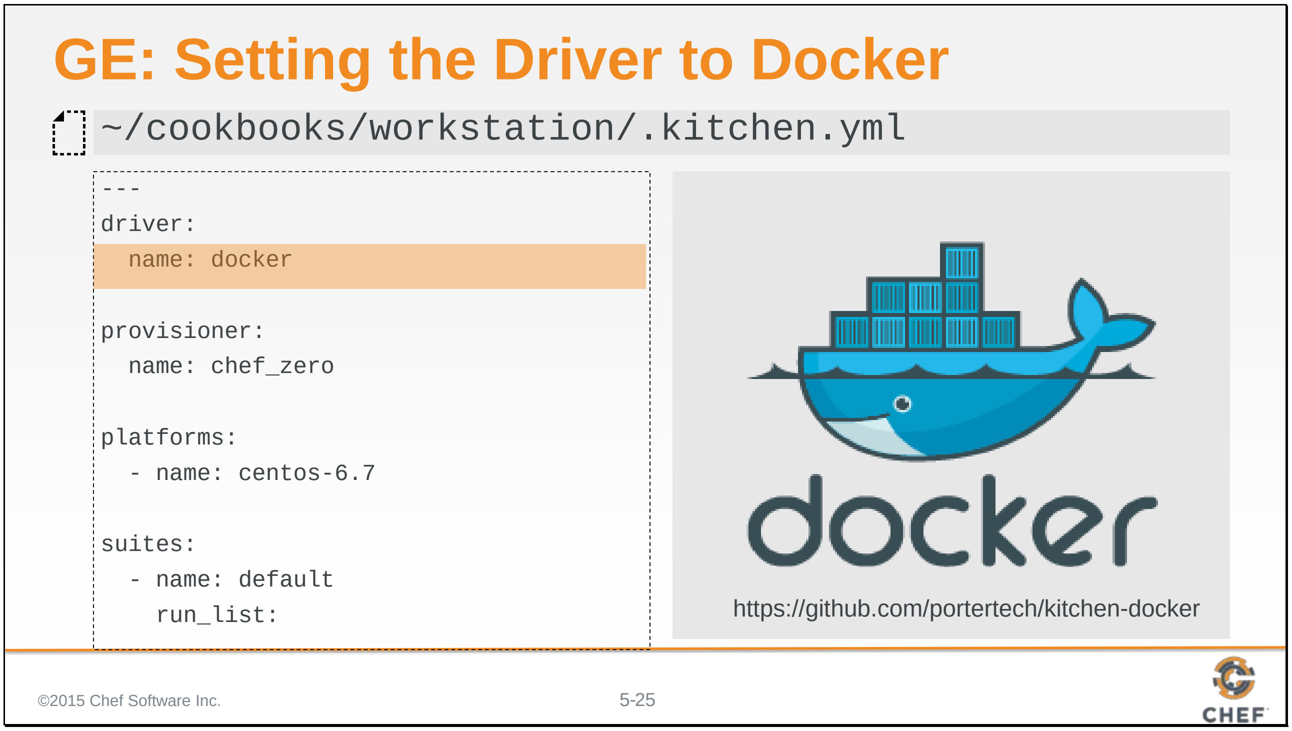
Remembering our objective, we want to update our .kitchen.yml file to use the Docker driver and we want to test against a single platform named centos 6.7.

Slide 24



Let's change into our workstation cookbook's directory.

Slide 25



Docker is a driver. So replace the existing vagrant driver, in your .kitchen.yml, with the Docker driver.

.

Slide 26

GE: Setting the Platform to centos-6.7

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

platforms:
  - name: centos-6.7

suites:
  - name: default
    run_list:
```

https://www.centos.org

CHEF

We also want to update our platforms to list only centos-6.7.

Slide 27

GE: Look at the Test Matrix

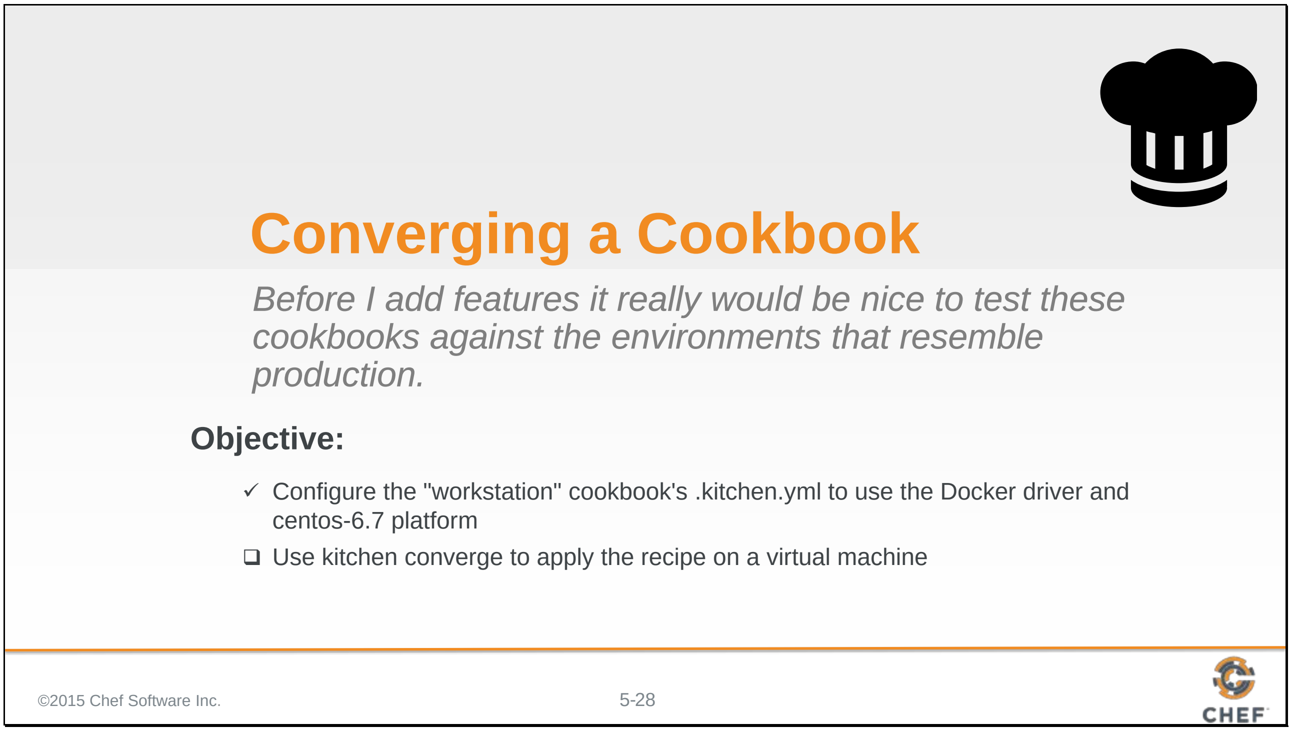
```
$ kitchen list
```

```
Instance            Driver  Provisioner  Verifier  Transport  Last Action
default-centos-67   Docker  ChefZero     Busser    Ssh        <Not Created>
```


CHEF

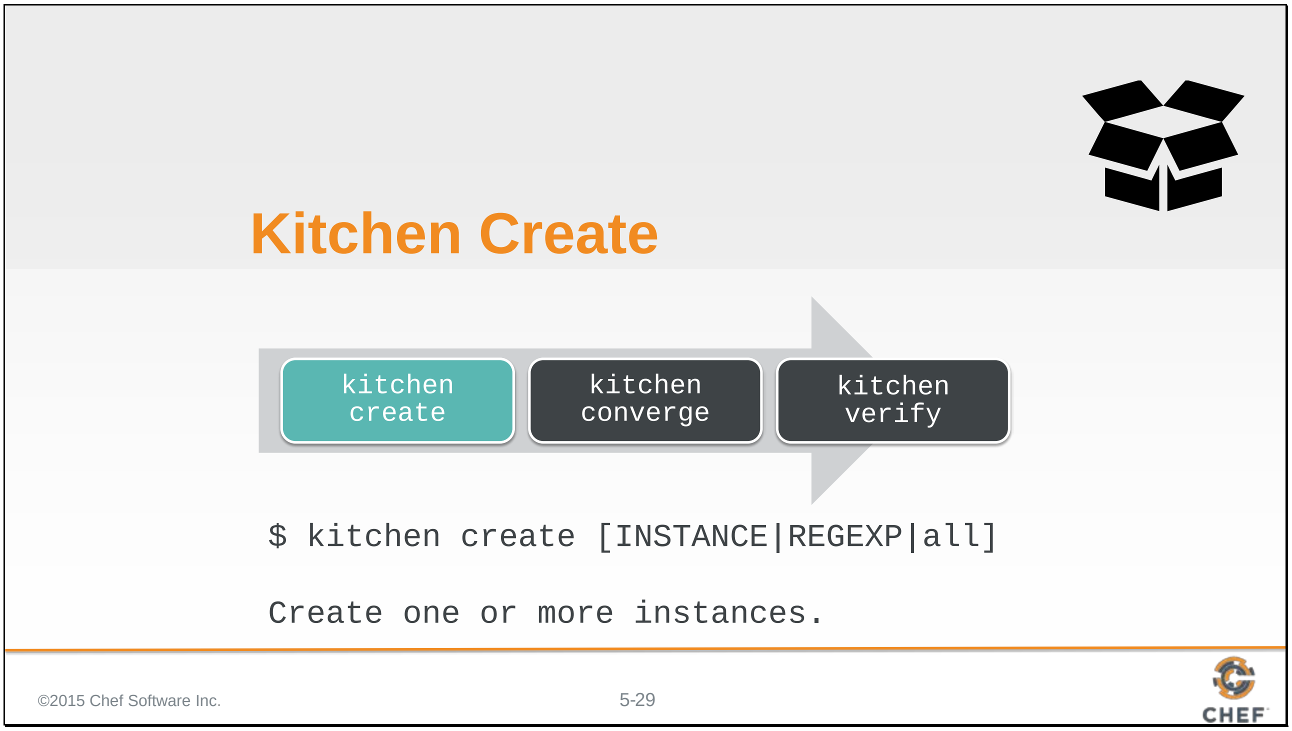
Run the `kitchen list` command to display our test matrix. You should see a single instance.

Slide 28



Now that we've defined the test matrix that we want to support, it is time to understand how to use Test Kitchen to create an instance, converge a run list of recipes on that instance, verify that the instance is in the desired state, and then destroy the instance.

Slide 29



The first kitchen command is `kitchen create`.

To create an instance means to turn on virtual or cloud instances for the platforms specified in the kitchen configuration.

In our case, this command would use the Docker driver to create a docker image based on centos-6.7.

Slide 30

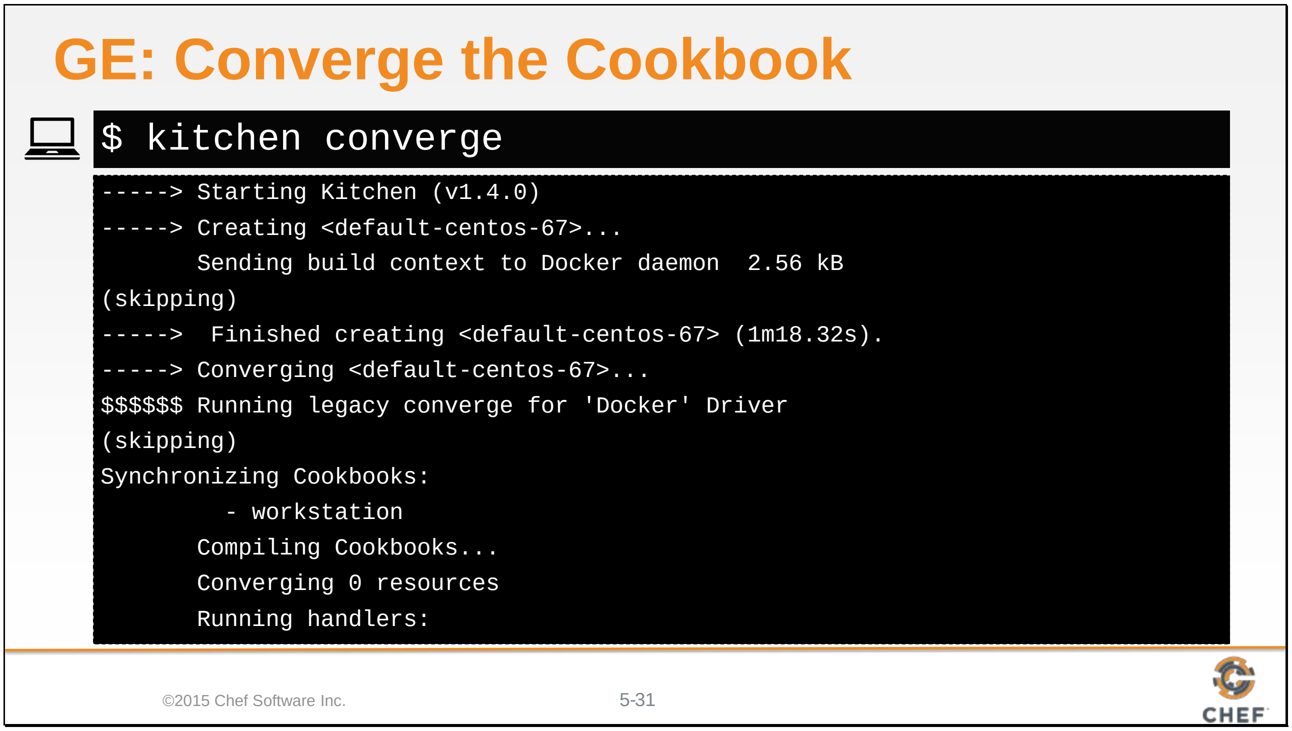
Creating an image gives us a instance to test our cookbooks but it still would leave us with the work of installing chef and applying the cookbook defined in our .kitchen.yml run list.

So let's introduce you to the second kitchen command: `kitchen converge`.

Converging an instance will create the instance if it has not already been created. Then it will install chef and apply that cookbook to that instance.

In our case, this command would take our image and install chef and apply the workstation cookbook's default recipe.

Slide 31



Be sure you are at ~/cookbooks/workstation and then run `kitchen converge` to verify that the workstation cookbook is able to converge the default recipe against the platform centos 6.7.

The workstation cookbook should successfully apply the default recipe. If an error occurs, lets stop and troubleshoot the issues.

Slide 32

Do the same thing again for the apache cookbook. Update the .kitchen.yml file so that it converges the apache cookbook's default recipe on the centos-6.7 platform with the docker driver.

Slide 33

Lab: Configuring Test Kitchen for Apache

~/cookbooks/apache/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

platforms:
  - name: centos-6.7

suites:
  - name: default
    run_list:
```


Like you did before, update the .kitchen.yml file to use the docker driver and the centos-6.7 platform.

Slide 34

Lab: Return Home and Move into the Cookbook

```
$ cd ~/cookbooks/apache
```


Change into the apache cookbook folder.

Slide 35

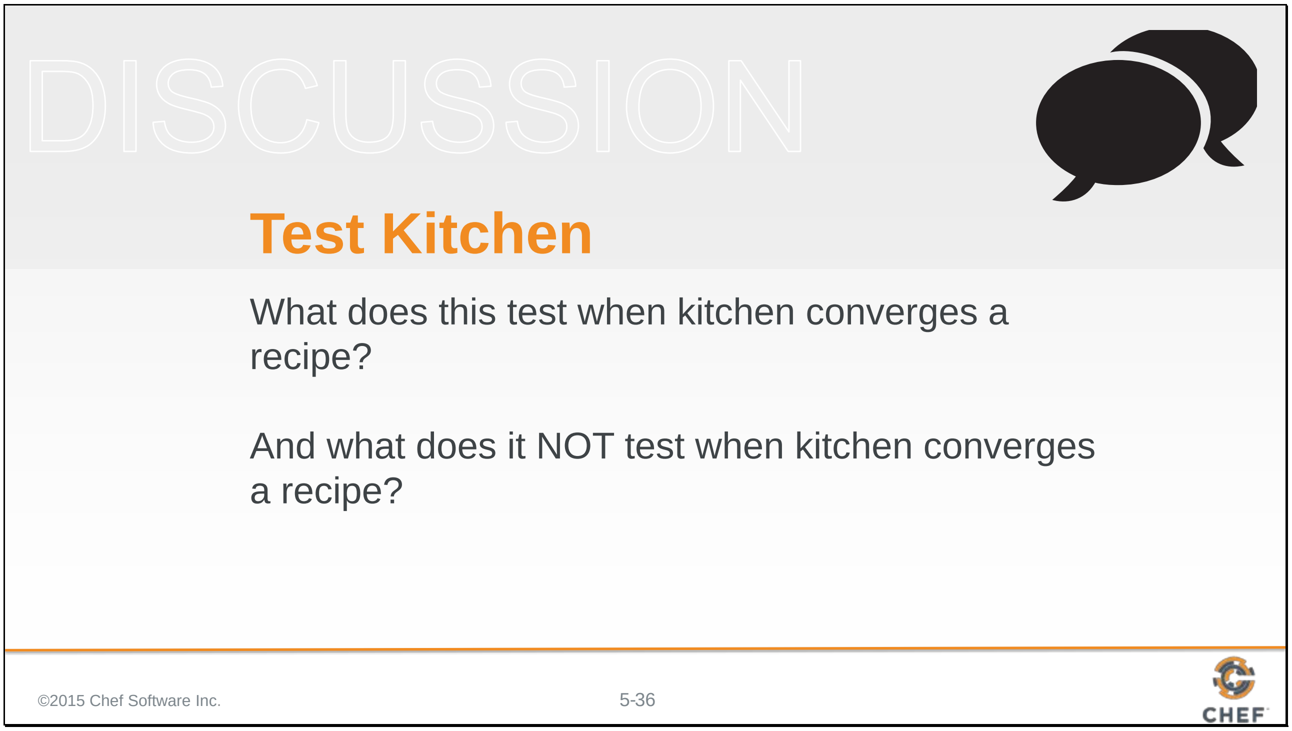
Lab: Converge the Cookbook

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.4.0)
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  2.56 kB
       Sending build context to Docker daemon
(skipping)
 Installing Chef
       installing with rpm...
       warning: /tmp/install.sh.23/chef-12.4.1-1.el6.x86_64.rpm: Header V4
DSA/SHA1 Signature, key ID 83ef826a: NOKEY
(skipping)
 Synchronizing Cookbooks:
         - apache
       Compiling Cookbooks...
```


Execute `kitchen converge` to validate that our apache cookbook's default recipe is able to converge on the centos-6.7 instance.

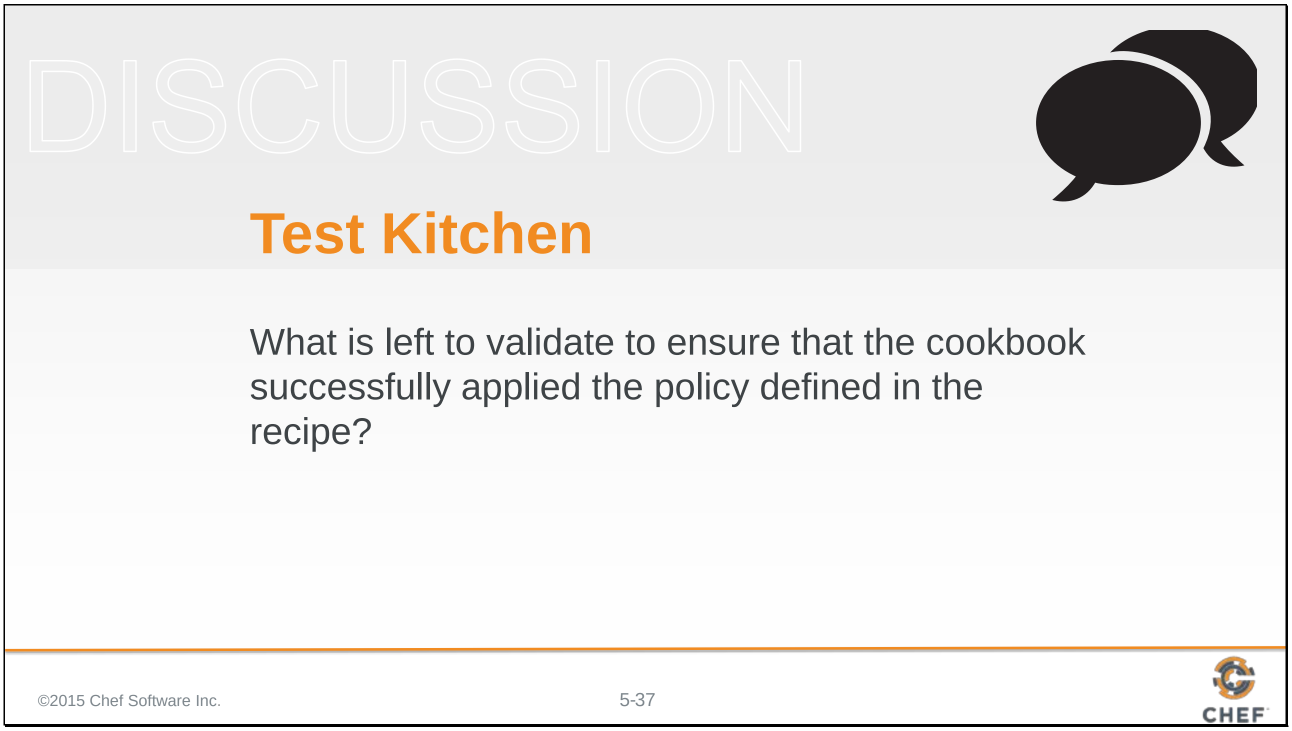
Slide 36



So what does this test when kitchen converges a recipe?

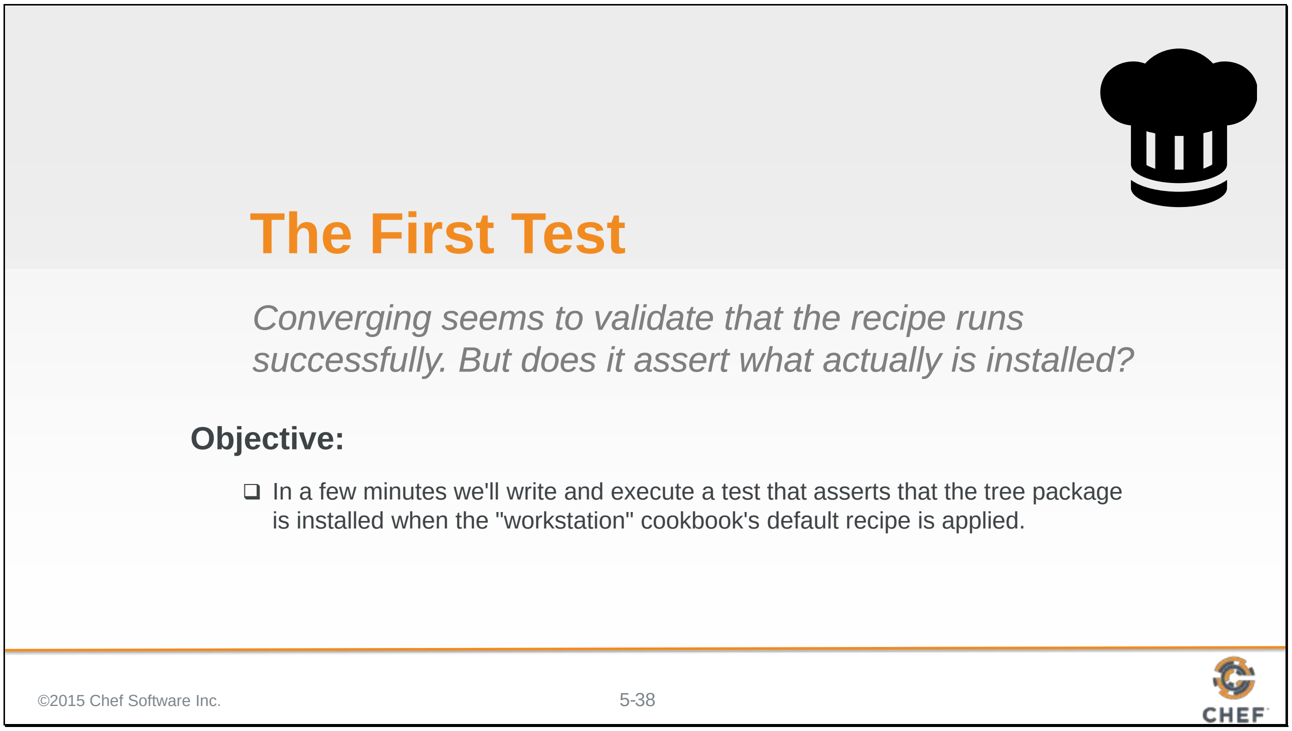
What does it NOT test when kitchen converges a recipe?

Slide 37



What is left to validate to ensure that the cookbook successfully applied the policy defined in the recipe?

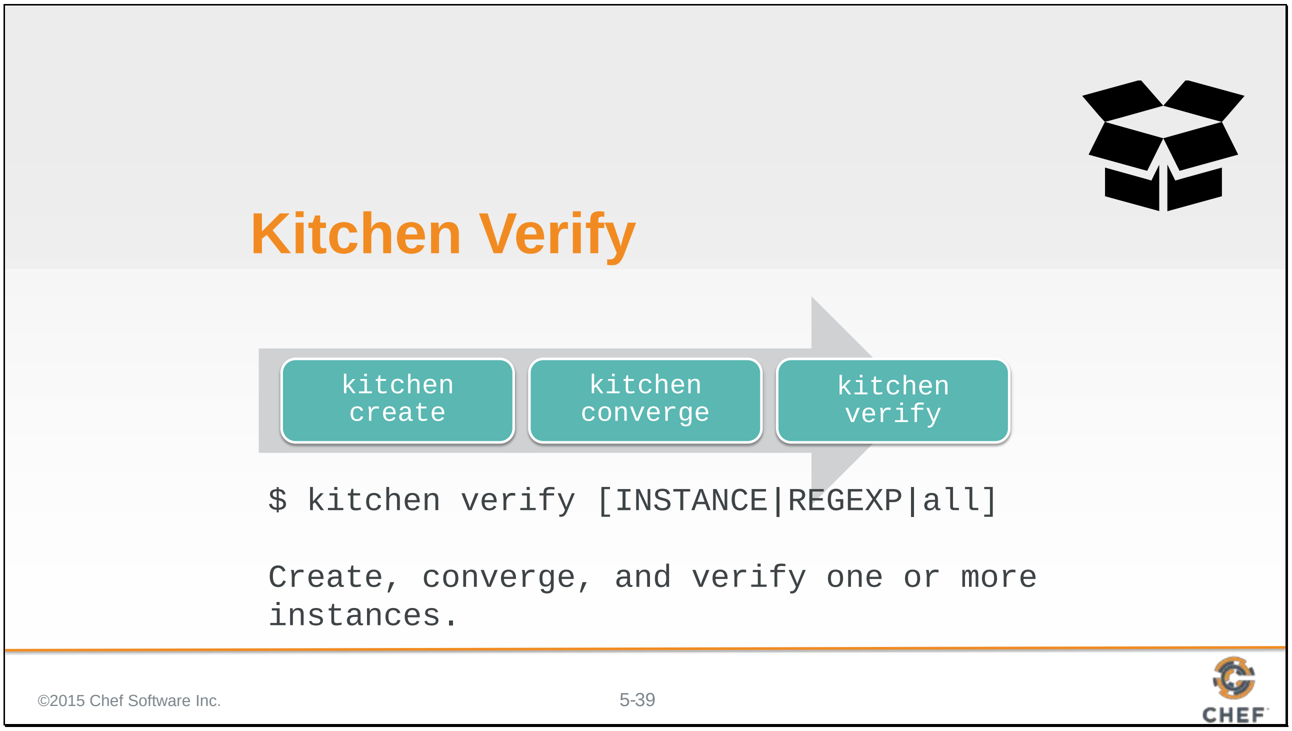
Slide 38



There is no automation that automatically understands the intention defined in the recipes we create. To do that we will define our own automated test.

Lets explore testing by adding a simple test to validate that the tree package is installed after converging the workstation cookbook's default recipe. We'll do this together in a few minutes.

Slide 39



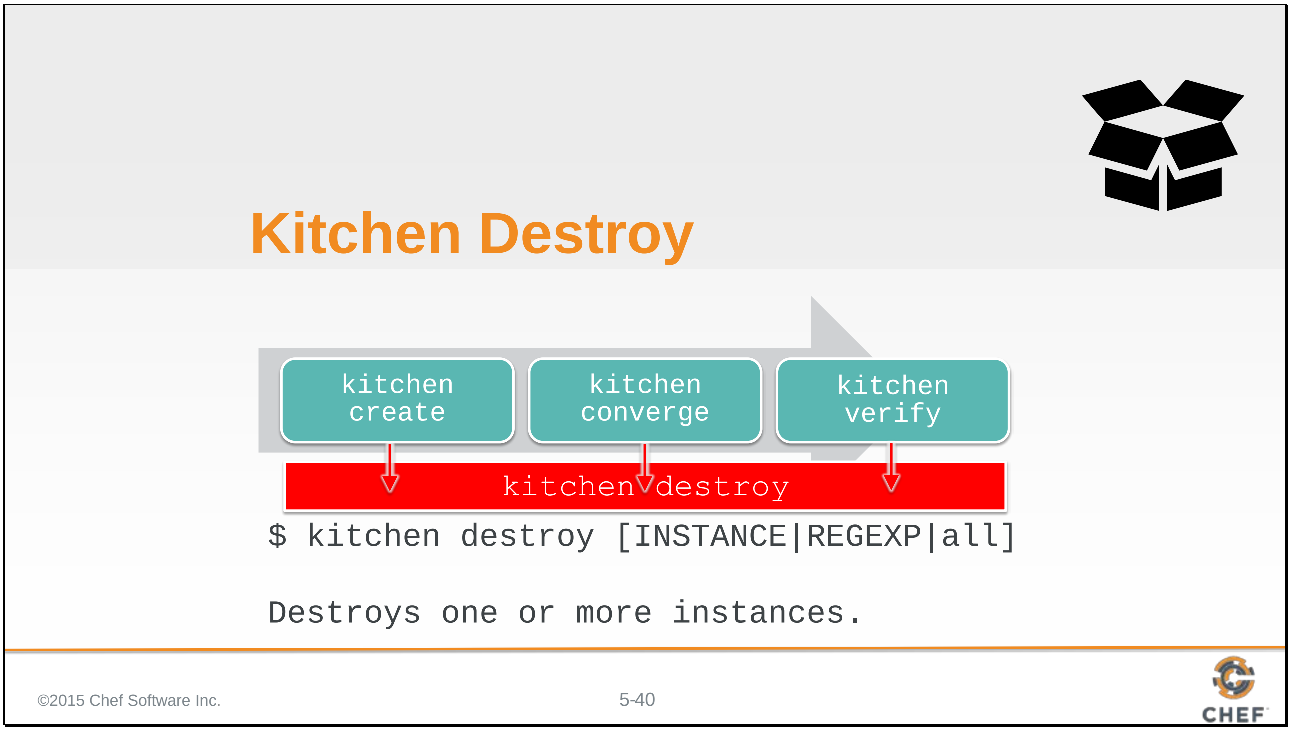
The third kitchen command is `kitchen verify`.

To verify an instance means to:

- Create a virtual or cloud instances, if needed
- Converge the instance, if needed
- And then execute a collection of defined tests against the instance

In our case, our instance has already been created and converged so when we run `kitchen verify` it will execute the tests that we will later define.

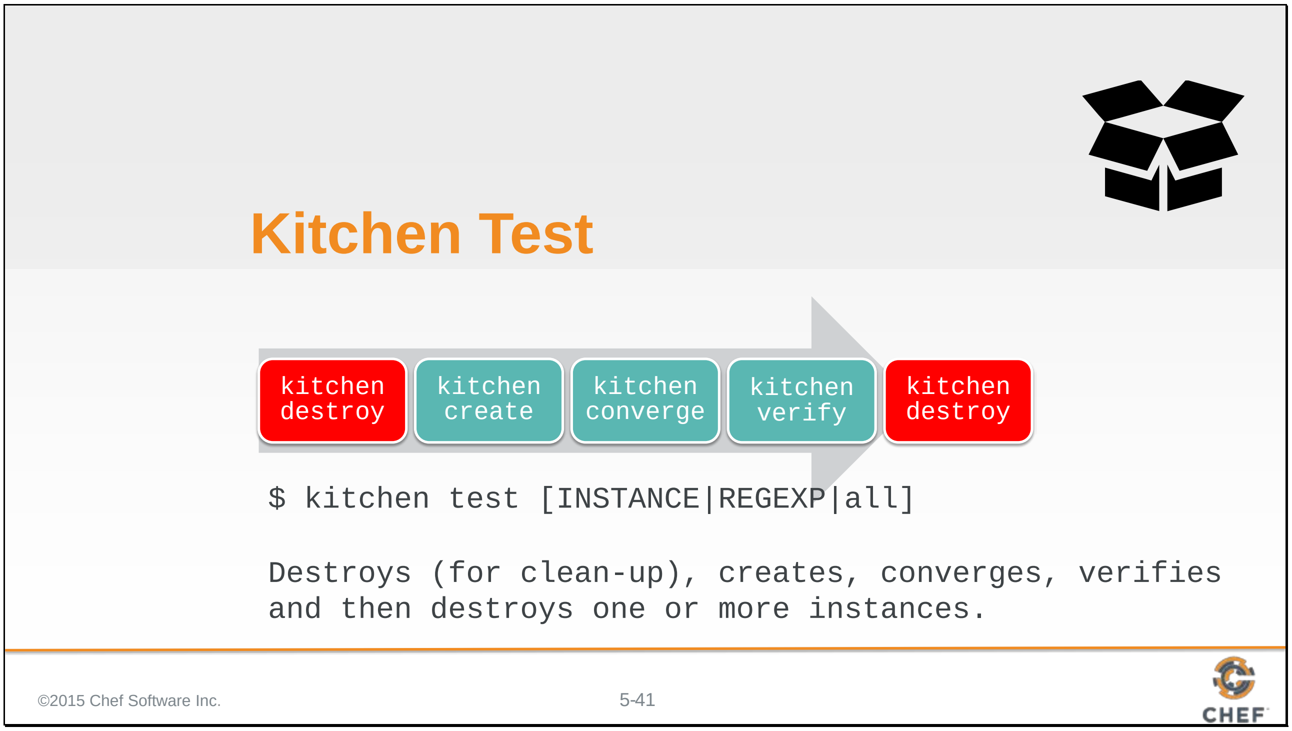
Slide 40



The fourth kitchen command is `kitchen destroy`.

Destroy is available at all stages and essentially cleans up the instance.

Slide 41

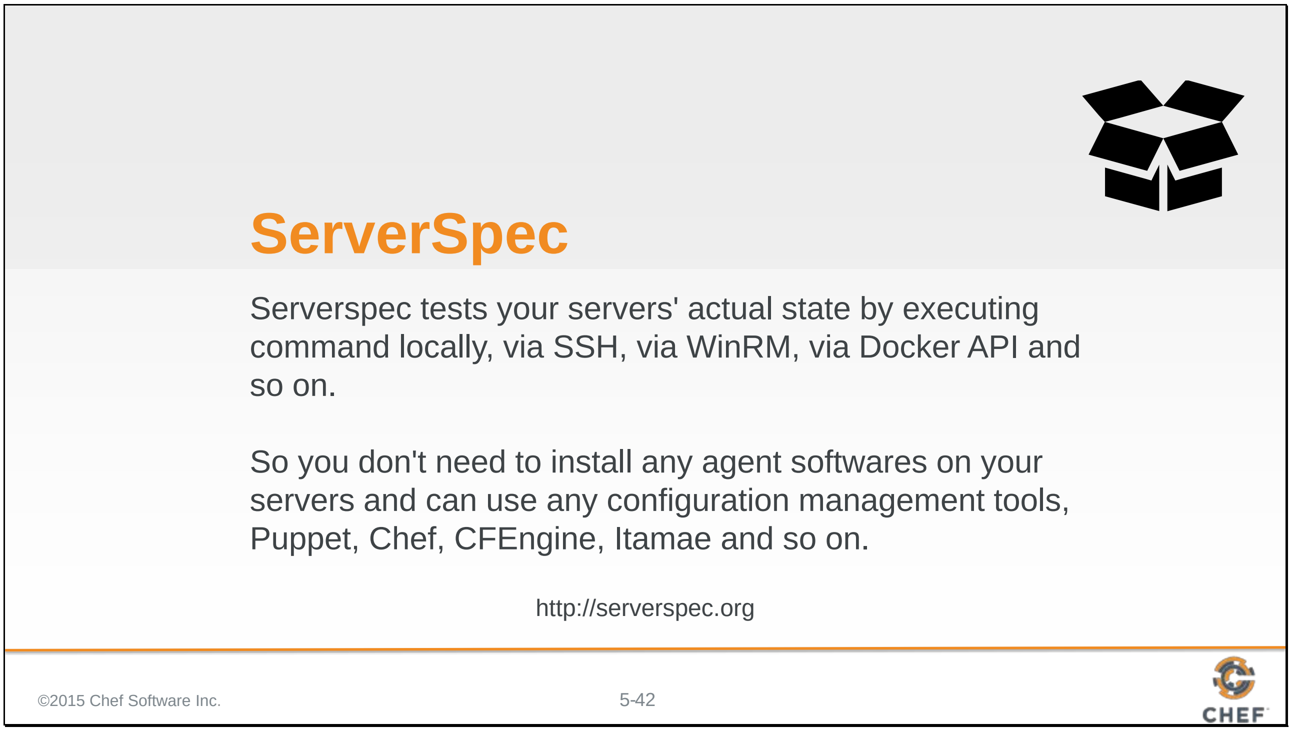


There a single command that encapsulates the entire workflow - that is `kitchen test`.

Kitchen test ensures that if the instance was in any state - created, converged, or verified - that it is immediately destroyed. This ensures a clean instance to perform all of the steps: create; converge; and verify. `kitchen test` completes the entire execution by destroying the instance at the end.

Traditionally this all encompassing workflow is useful to ensure that we have a clean state when we start and we do not leave a mess behind us.

Slide 42



So `kitchen verify` and `kitchen test` are the two kitchen commands that we can use to execute a body of tests against our instances. Now it is time to define those tests with ServerSpec.

ServerSpec is one of many possible test frameworks that Test Kitchen supports. It is a popular choice for those doing Chef cookbook development because ServerSpec is built on a Ruby testing framework named RSpec.

RSpec is similar to Chef - as it is a Domain Specific Language, or DSL, layered on top of Ruby. Where Chef gives us a DSL to describe the policy of our system, RSpec allows us to describe the expectations of tests that we define. ServerSpec adds a number of helpers to RSpec to make it easy to test the state of a system.

Slide 43

Example: Is the 'tree' Package Installed?

```
describe package('tree') do
  it { should be_installed }
end
```

I expect the package tree should be installed.

http://serverspec.org/resource_types.html#package

Here is an example of an isolated ServerSpec expectation that states: We expect the package named 'tree' to be installed.

Slide 44

GE: Requiring a Test Helper Tools

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require "spec_helper"

describe "workstation::default" do

  describe package('tree') do
    it { should be_installed }
  end

end
```

Loads a helper file with that name in the same directory.

http://kitchen.ci/docs/getting-started/writing-test

CHEF

For our test to work with Test Kitchen there are a number of conventions that we need to adhere to have our test code load correctly.

First, we need to create a test file, often referred to as a spec file at the following path. The structure of the path is a convention defined by Test Kitchen and will automatically be loaded when we run `kitchen verify`. Fortunately for us the test file has already been created when we used 'chef' to generate the workstation cookbook.

Within the spec file we need to first require a helper file. The helper is were we keep common helper methods and library requires in one location. This is likely already present within the generated test file.

Slide 45

GE: Describing the Test Context

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require "spec_helper"

describe "workstation::default" do

  describe package('tree') do
    it { should be_installed }
  end

end
```

Describing a body of tests for the "workstation" cookbook's default recipe.

https://relishapp.com/rspec/rspec-core/v/3-3/docs

Second, we define a describe method. RSpec, which ServerSpec is built on uses an english-like syntax to help us describe the various scenarios and examples that we are testing.

The 'describe' method takes two parameters - the first is the name of fully-qualifed recipe to execute (cookbook_name::recipe_name).

The second parameter is the block between the **do** and **end**. Within that block we can define more describe blocks that allow us to further refine the scenario we are testing.

Slide 46

GE: Our Assertion in a spec File

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require "spec_helper"

describe "workstation::default" do

  describe package('tree') do
    it { should be_installed }
  end

end
```

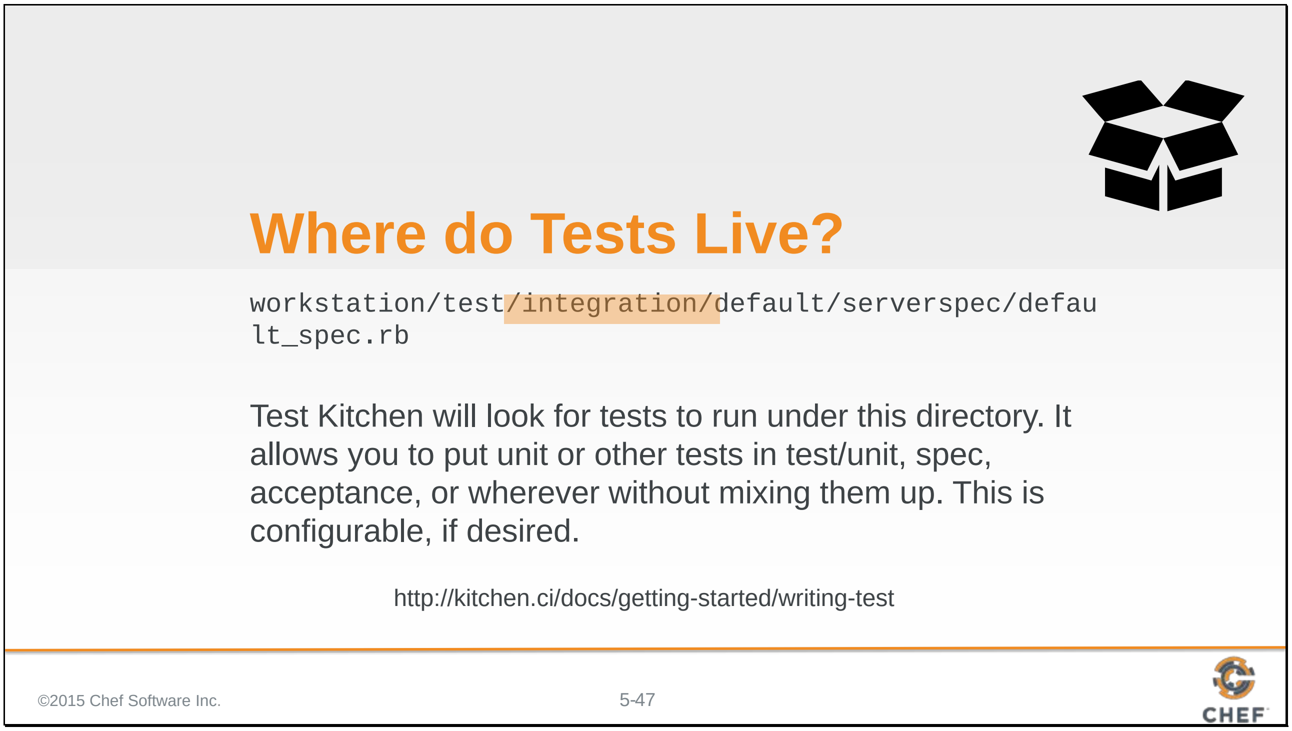
When we converge the workstation cookbook's default recipe we expect the package named tree to be installed.

http://serverspec.org/resource_types.html#package

Here is that example expectation that we showed you earlier except now it is displayed here within this context. This states that when we converge the workstation cookbook's default recipe we want to assert that the tree package has been installed.

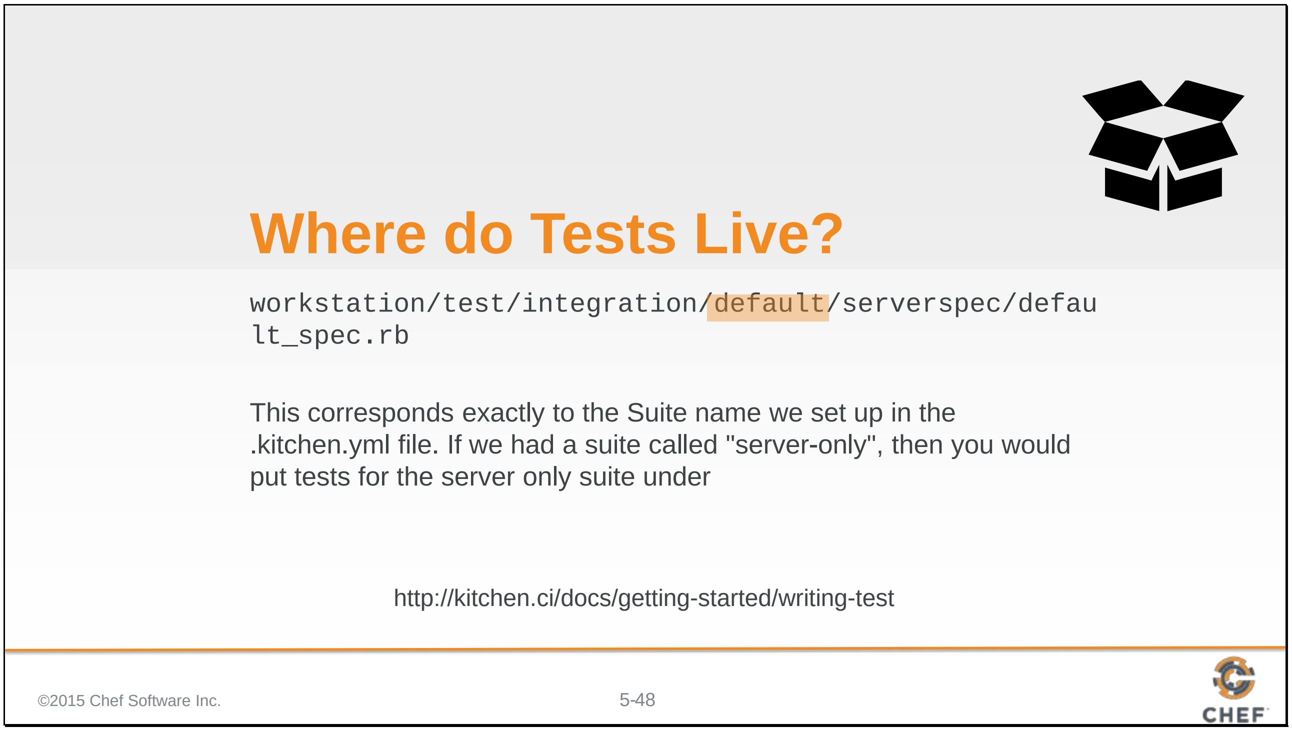
Add this expectation to the specification file at the specified path.

Slide 47



Lets take a moment to describe the reason behind this long directory path. Within our cookbook we define a test directory and within that test directory we define another directory named 'integration'. This is the basic file path that Test Kitchen expects to find the specifications that we have defined.

Slide 48



The next part the path, 'default', corresponds to the name of the test suite that is defined in the .kitchen.yml file. In our case the name of the suite is 'default' so when test kitchen performs a `kitchen verify` for the default suite it will look within the 'default' folder for the specifications to run.

Slide 49

Where do Tests Live?

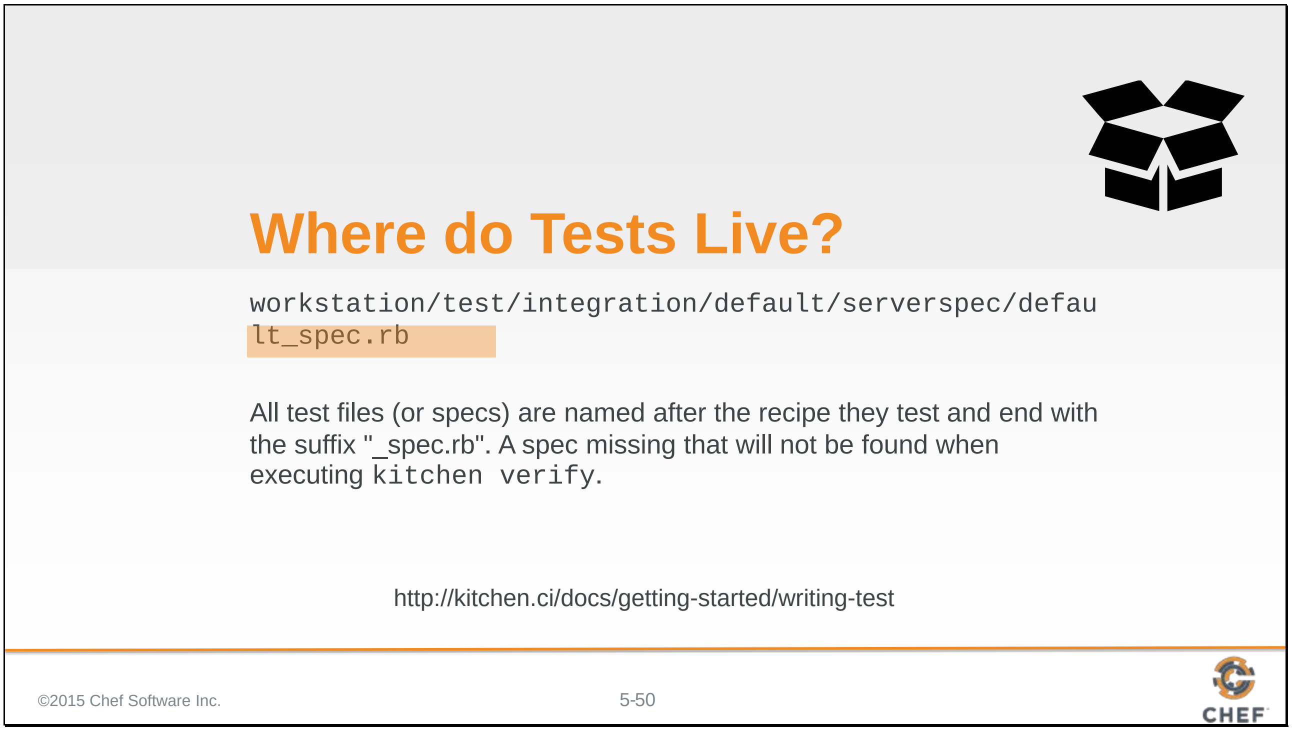
```
workstation/test/integration/default/serverspec/default_spec.rb
```

This tells Test Kitchen (and Busser) which Busser runner plugin needs to be installed on the remote instance.

http://kitchen.ci/docs/getting-started/writing-test

'serverspec' is the kind of tests that we want to define. Test Kitchen supports a number of testing frameworks.

Slide 50



The final part of the path is the specification file. This is a ruby file. The naming convention for this file is the recipe name with the appended suffix of _spec.rb. All specification files must end with _spec.rb.

Slide 51

GE: Return Home and Move into the Cookbook

```
$ cd ~/cookbooks/workstation
```


Change into the workstation cookbook directory.

Slide 52

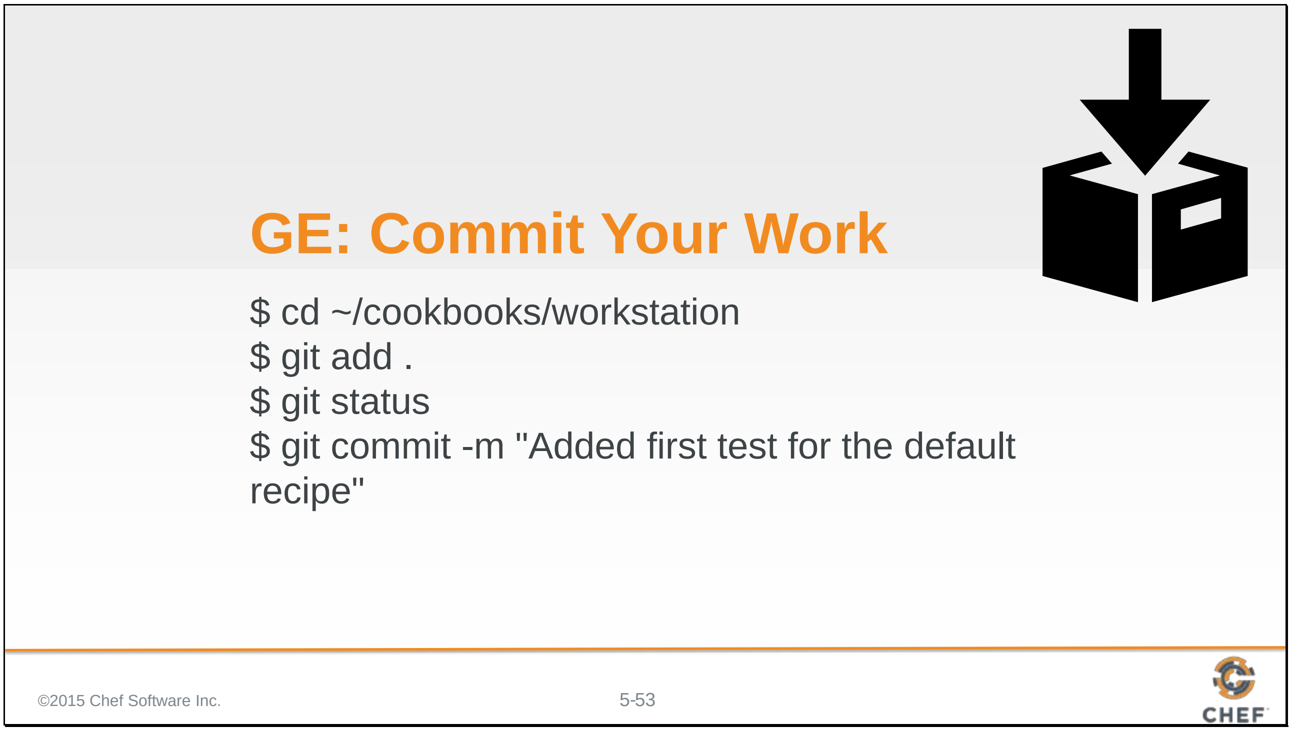
GE: Running the Specification

```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.4.0)
-----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
(skipping)
-----> Chef Omnibus installation detected (install only if missing)
       Transferring files to <default-centos-67>
       Starting Chef Client, version 12.4.1
(skipping)
       Running handlers:
       Running handlers complete
       Chef Client finished, 6/6 resources updated in 64.426896317 seconds
       Finished converging <default-centos-67> (1m9.02s).
-----> Kitchen is finished. (1m9.69s)
```

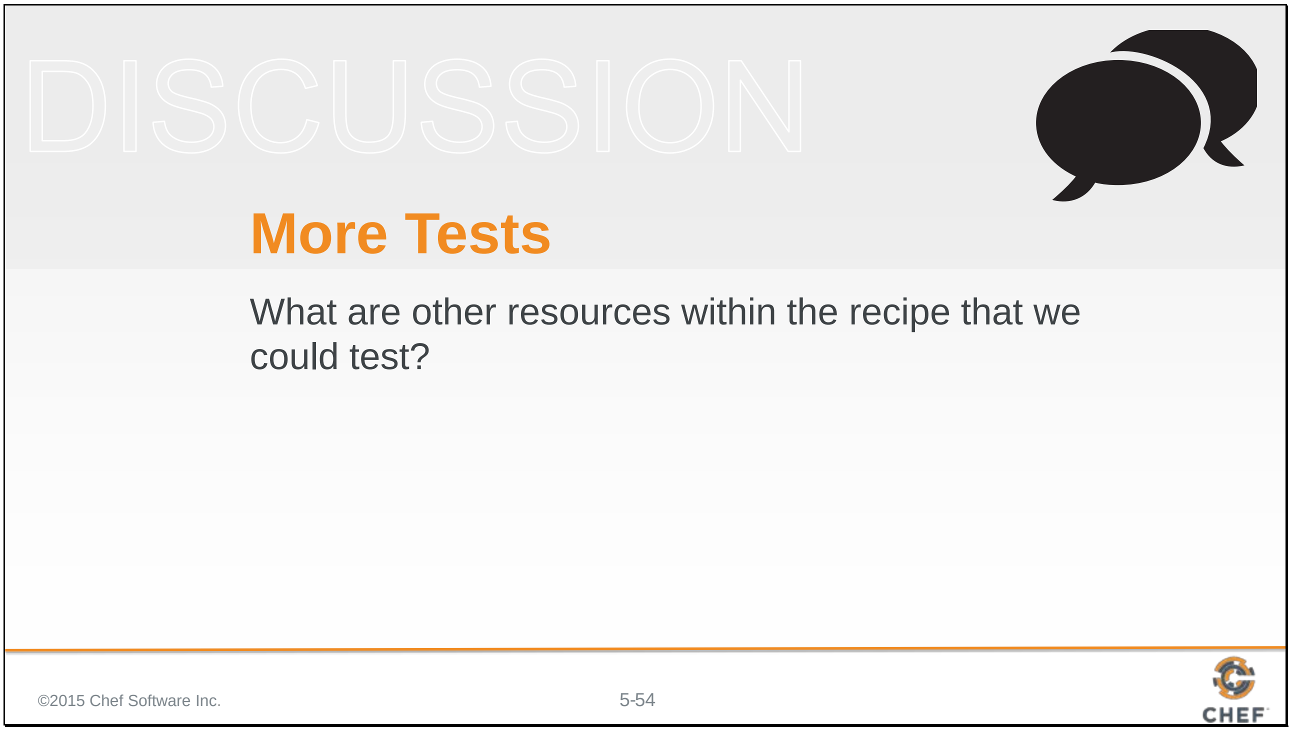

With the first test created, lets verify that the package named 'tree' is installed when we apply the workstation cookbooks default recipe using the `kitchen verify` command to execute our test

Slide 53



With the first test completed. It is time to commit the changes to source control.

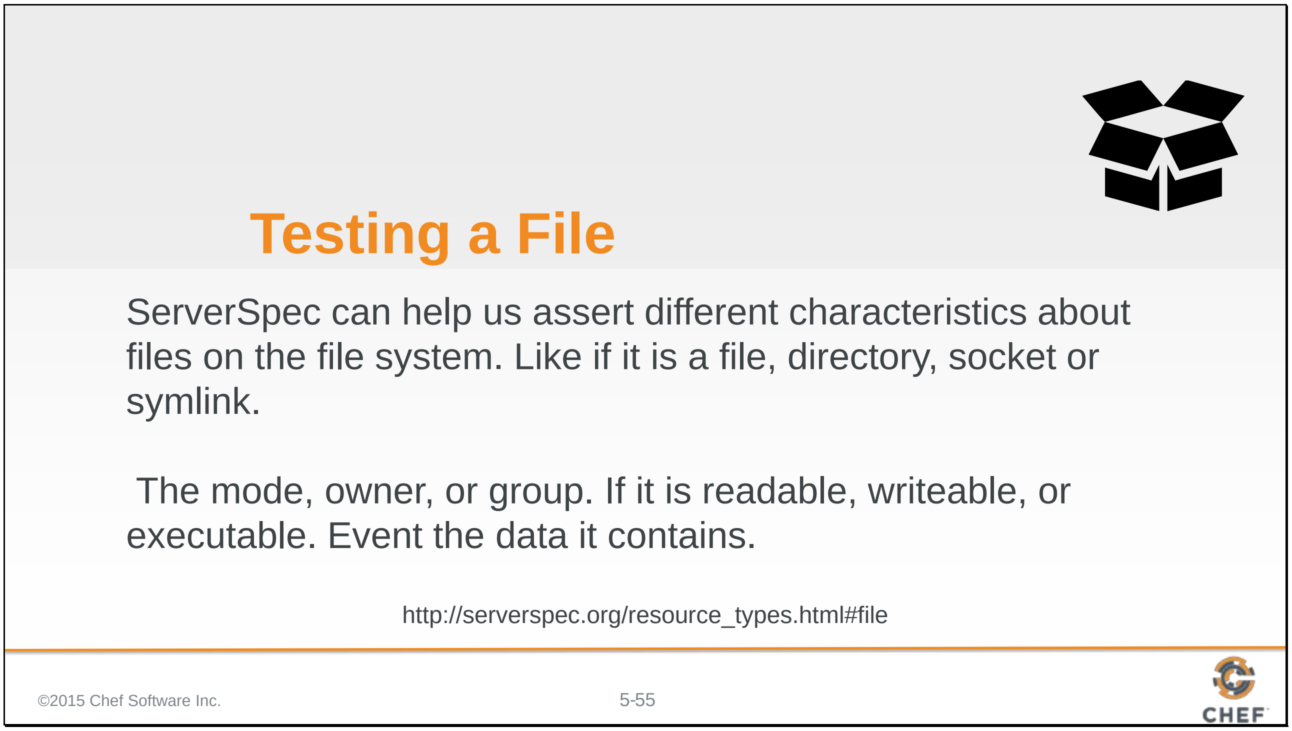
Slide 54



Now that we've explored the basic structure of writing tests to validate our cookbook.

What are other resources within the recipe that we could tests?

Slide 55



ServerSpec provides a large number of helpers to assist us with many different resources on our system. Important to us in testing more of our workstation cookbook's default recipe is the ability to verify if a file was written, what are the permissions of that file, and what are the contents.

Let's look at a few examples...

Slide 56

Example: The File Contains Data

```
describe file("/etc/passwd") do
  it { should be_file }
end
```

I expect the file named "/etc/passwd" to be a file (as opposed to a directory, socket or symlink).

http://serverspec.org/resource_types.html#file

Here we are describing an expectation that the file named "/etc/passwd" is a file.

Slide 57

Example: The File Contains Specific Content

```
describe file("/etc/httpd/conf/httpd.conf") do
  its(:content) { should match /ServerName
www.example.jp/ }
end
```

I expect the file named "/etc/httpd/conf/httpd.conf" to have content that matches "ServerName www.example.jp"

http://serverspec.org/resource_types.html#file

Here we are describing an expectation that the file named "/etc/httpd/conf/httpd.conf" has contents that match the following regular expression. Asserting that somewhere in the file we will find the following bit of text.

Slide 58

Example: The File is Owned by a Particular User

```
describe file("/etc/sudoers") do
  it { should be_owned_by "root" }
end
```

I expect the file named "/etc/sudoers" to be owned by the "root" user.

Here we are describing an expectation that the file named "/etc/sudoers" should be owned by the root user.

Slide 59

Lab: More Tests

- ❑ Add tests that validate that the remaining package resources have been installed (http://serverspec.org/resource_types.html#package)
- ❑ Add tests that validate the file resource (http://serverspec.org/resource_types.html#file)
- ❑ Run kitchen verify to validate the test meets the expectations that you defined
- ❑ Commit your changes

CHEF

As a lab exercise, we want you to define additional tests that validate the remaining resources within our default recipe.

Add tests for the remaining package resources that are converged by the "workstation" cookbook's default recipe.

You may also add tests for the file resource to ensure the file is present, that the contents are correctly defined, that it is owned by a particular user and owned by a particular group.

Slide 60

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require "spec_helper"

describe "workstation::default" do

  describe package("tree") do
    it { should be_installed }
  end

  describe package("git") do
    it { should be_installed }
  end

end
```

The package named "git" is installed.

http://serverspec.org/resource_types.html#package

Let's review the lab.

Here we are verifying that the package git is installed. The structure of the test is very similar to the one we demonstrated earlier. You'll likely have another test that validates the editor you specified is also installed.

Slide 61

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
...

  describe package("git") do
    it { should be_installed }
  end

  describe file("/etc/motd") do
    it { should be_owned_by "root" }
  end

end
```

The file named "/etc/motd" should be owned by "root".

http://serverspec.org/resource_types.html#file

For the file resource, we chose only to verify that the file named "/etc/motd" is owned by the root user. You may have verified that it was a file, that it belonged to a group, and that it contained content you felt important to verify.

Slide 62

Lab: Running the Specification

```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.4.0)
-----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
(skipping)
-----> Chef Omnibus installation detected (install only if missing)
       Transferring files to <default-centos-67>
       Starting Chef Client, version 12.4.1
(skipping)
       Running handlers:
       Running handlers complete
       Chef Client finished, 6/6 resources updated in 64.426896317 seconds
       Finished converging <default-centos-67> (1m9.02s).
-----> Kitchen is finished. (1m9.69s)
```


With more tests created lets verify all of these tests pass when we converged the workstation cookbooks default recipe. Use the `kitchen verify` command to execute the test

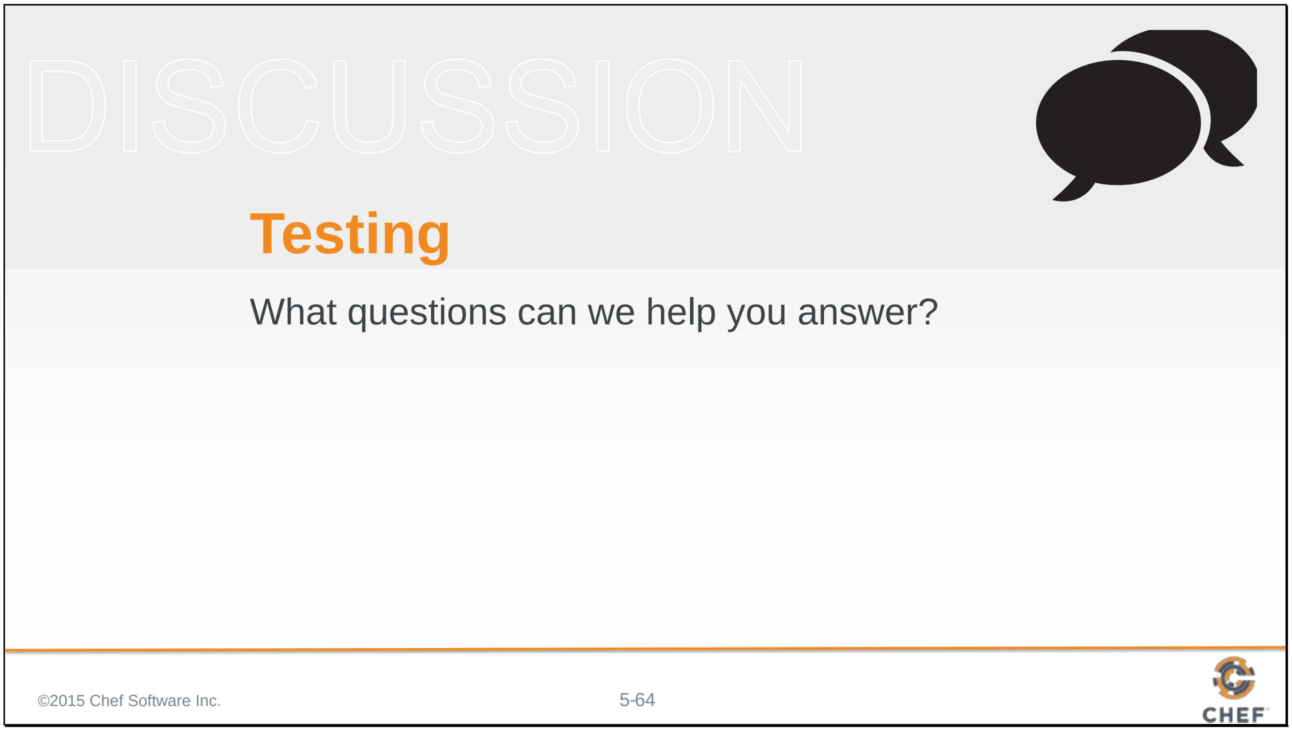
Slide 63

Lab: Commit Your Work

```
$ cd ~/cookbooks/workstation
$ git add .
$ git status
$ git commit -m "Added additional tests for default recipe"
```

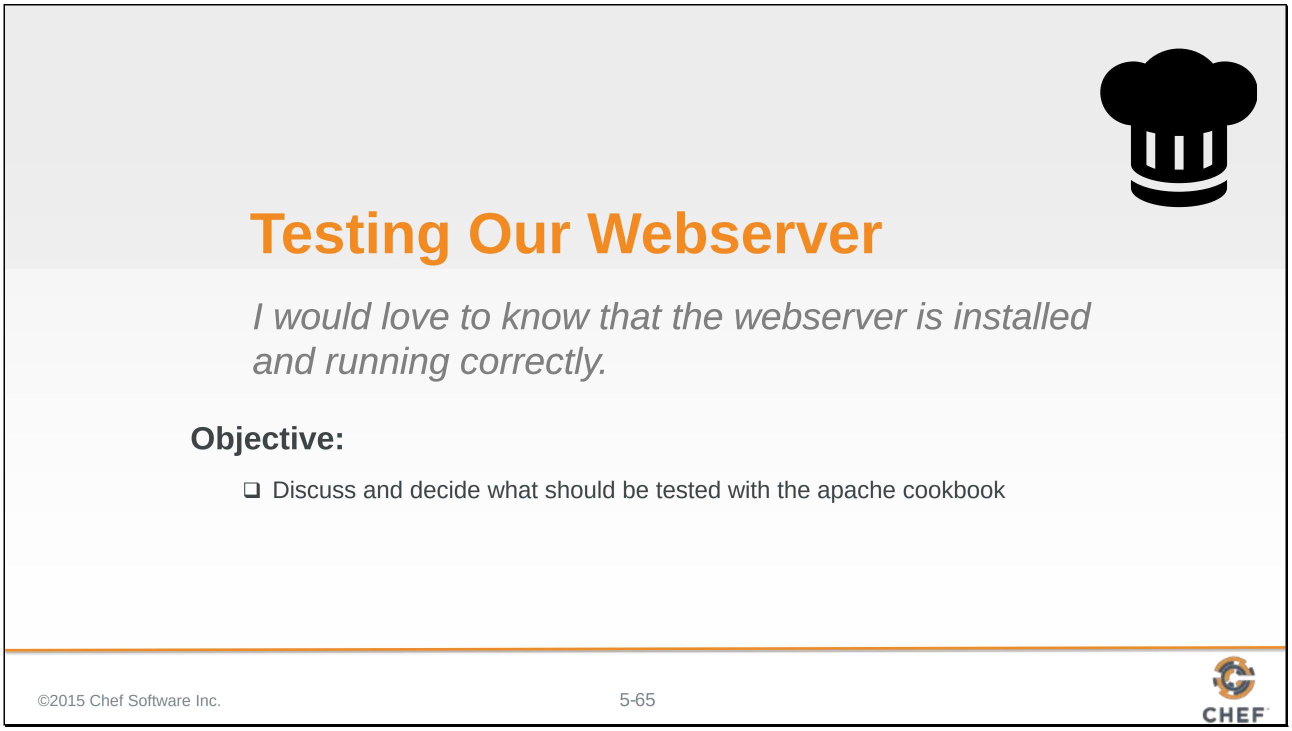

If all the tests that you defined are working then it is time to commit our changes to version control.

Slide 64



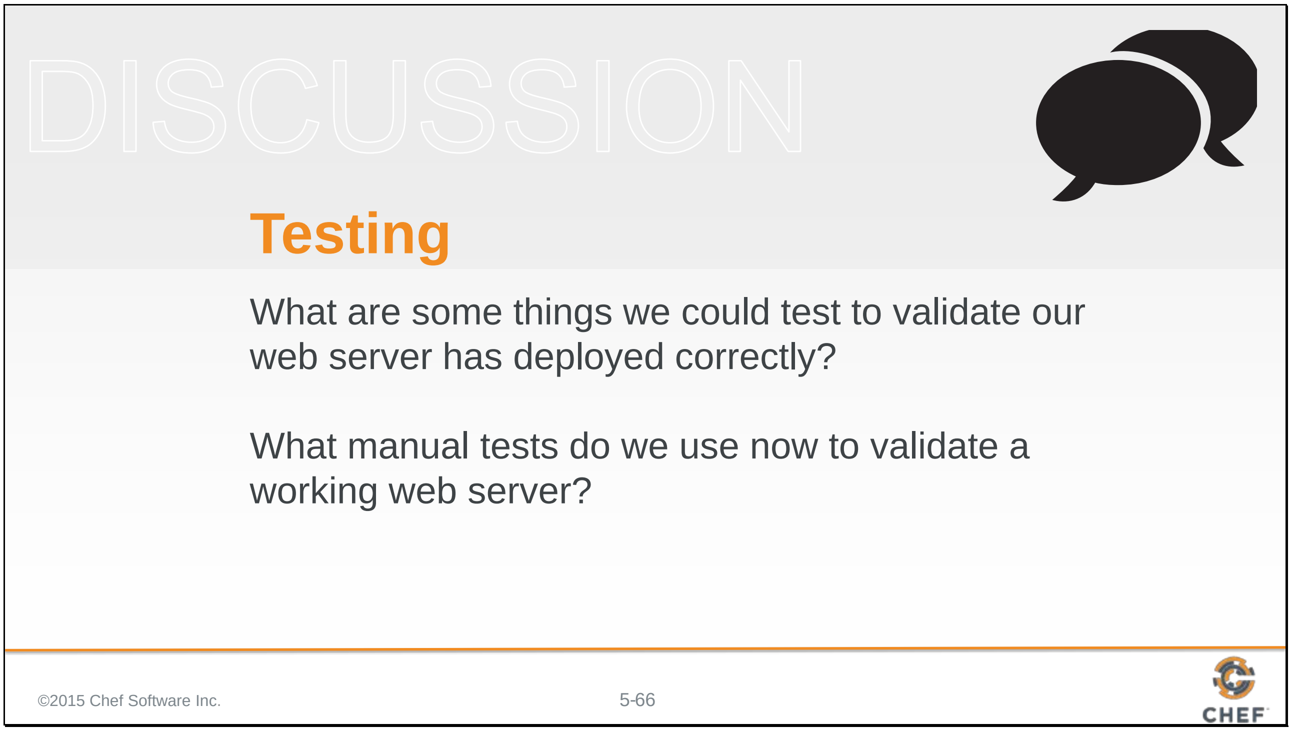
What questions can we help you answer?

Slide 65



Now lets turn our focus towards testing the apache cookbook.

Slide 66



What are some things we could test to validate our web server has deployed correctly?

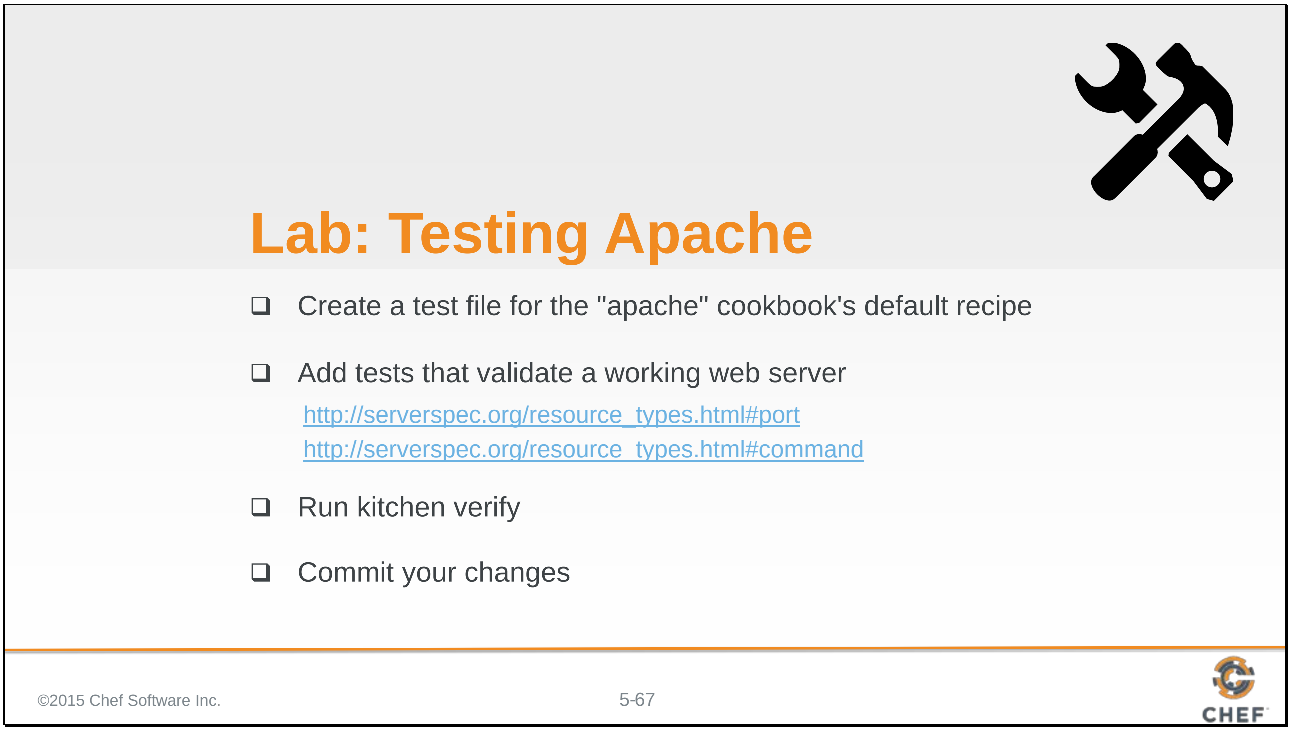
The apache cookbook is similar to the workstation cookbook. It has a package and file which are things that we have already tested. The new thing is the service. We could review the ServerSpec documentation to find examples on how to test the service.

But does testing the package, file and service validate that apache is hosting our static web page and returning the content to visitors of the instance?

What manual tests do we use now to validate a working web server?

After applying the recipes in the past we visited the site through a browser or verified the content through running the command 'curl localhost'.
Is that something that we could test as well? Does ServerSpec provide the way for us to execute a command and verify the results?

Slide 67

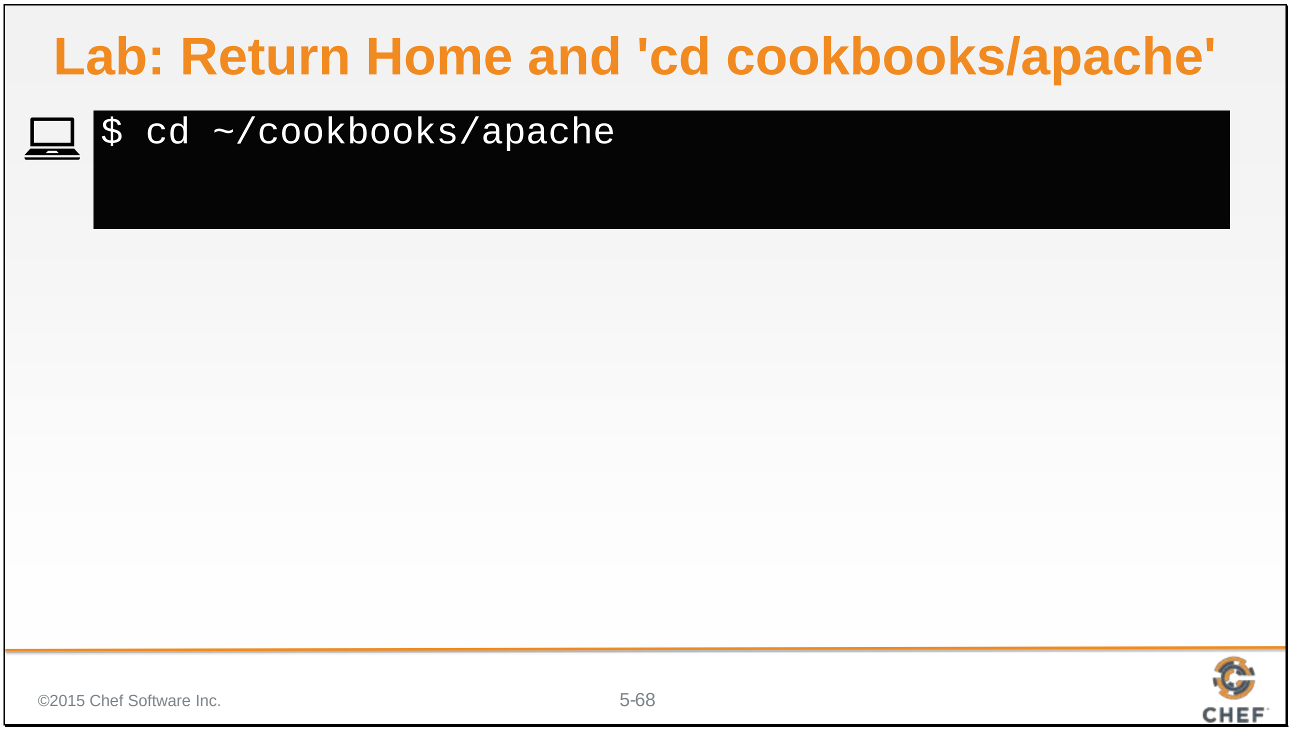


So for this final exercise, you are going to create a test file for the apache cookbook's default recipe.

That test will validate that you have a working web server. This means I want you to add the tests that you feel are necessary that the system is installed and working correctly.

When you are done execute your tests with `kitchen verify`.

Slide 68



Return home and then move into the apache cookbook's directory.

Slide 69

Lab: What Does the Webserver Say?

~/cookbooks/apache/test/integration/default/serverspec/default_spec.rb

```
require "spec_helper"

describe "apache::default" do
  describe port(80) do
    it { should be_listening }
  end

  describe command("curl
http://localhost") do
    its(:stdout) { should match /Hello,
world!/ }
  end
end
```

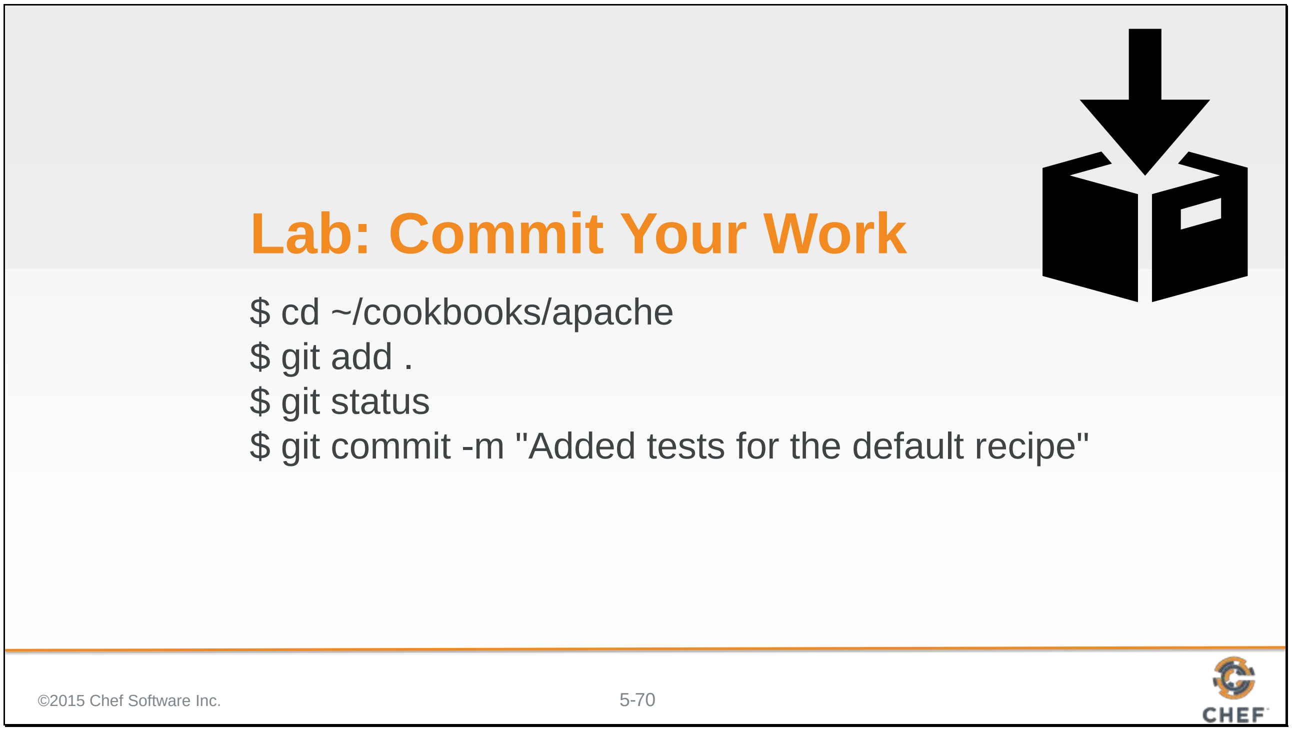
Port 80 should be listening.

The standard out from the command "curl http://localhost" should match "Hello, world!"

Here we chose to validate that port 80 should be listening for incoming connections.

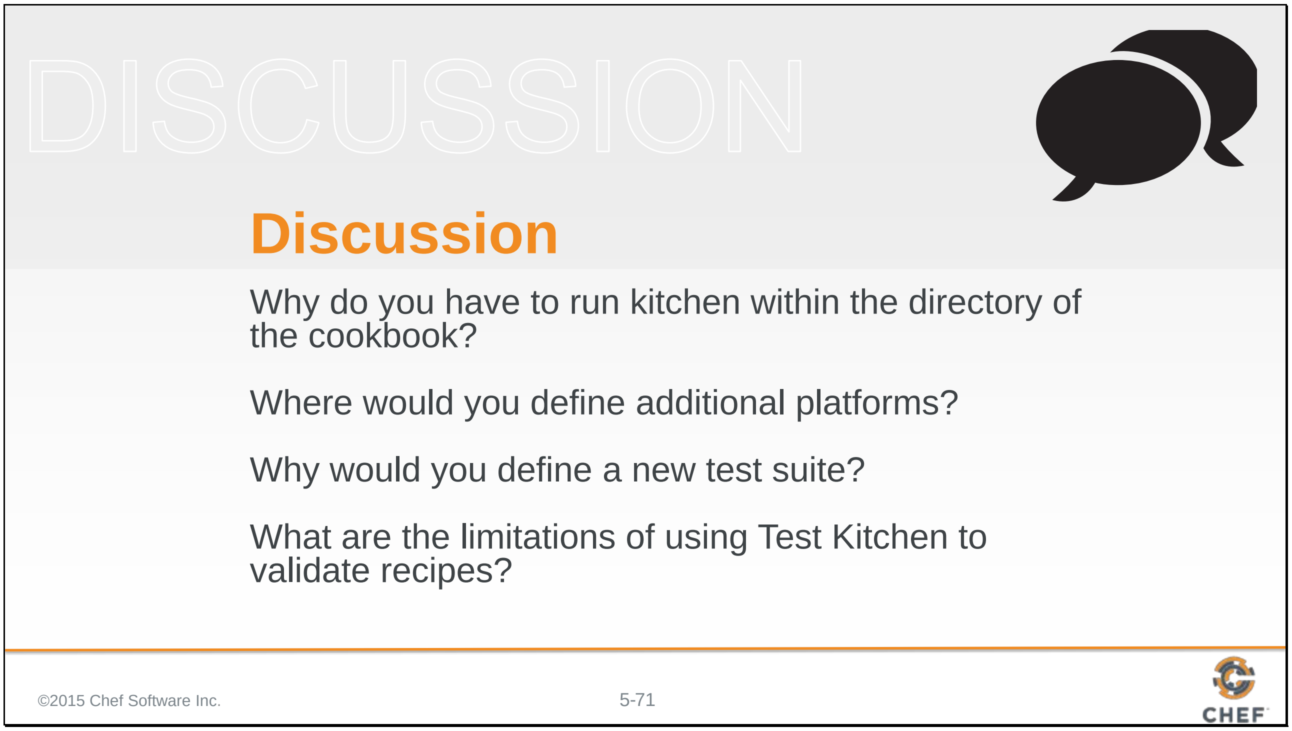
And we also validated that the standard out from the command "curl http://localhost" should match "Hello, world!".

Slide 70



Again, lets commit the work.

Slide 71



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 72

What questions can we help you answer?

Generally or specifically about chef-client, local mode, run lists, and include_recipe.

Slide 73

CHEF™

6: Details About the System

Slide 2

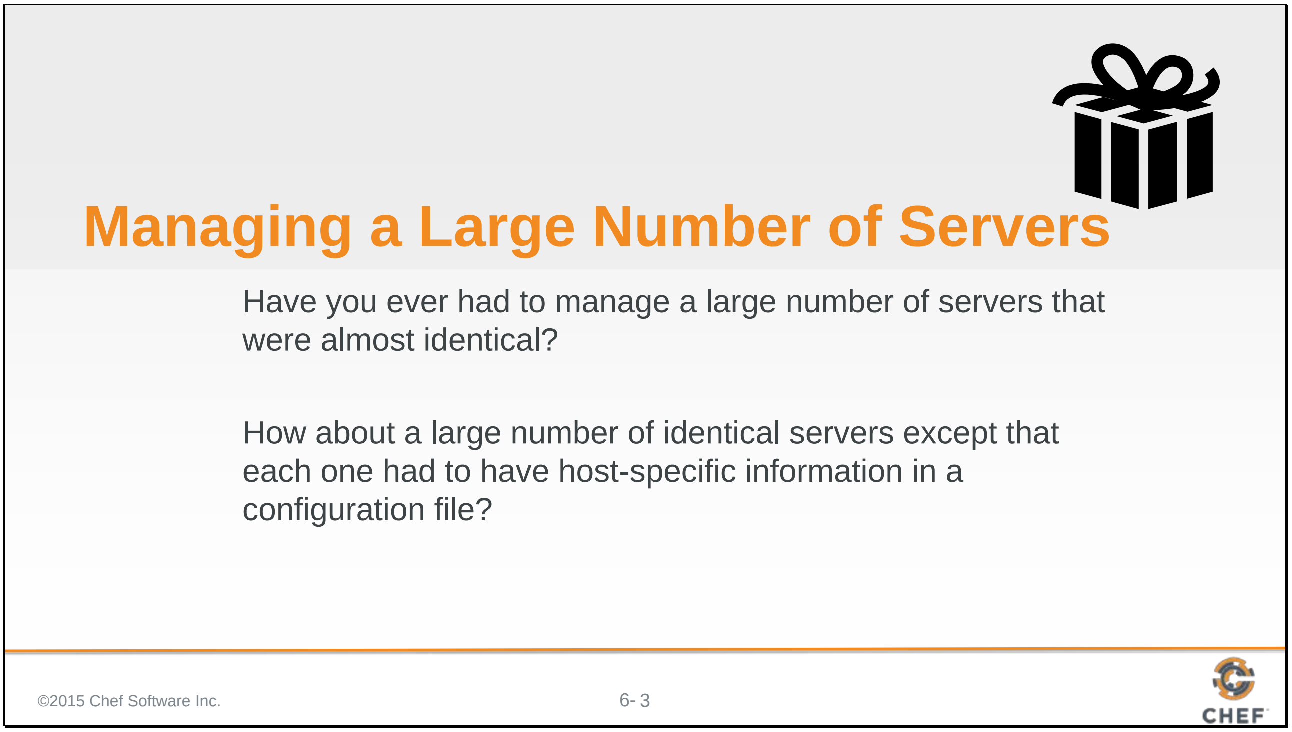
Objectives

After completing this module, you should be able to

- Capture details about a system
- Use the node object within a recipe
- Use Ruby's string interpolation
- Update the version of a cookbook

In this module you will learn how to capture details about a system, use the node object within a recipe, use Ruby's string interpolation, and update the version of a cookbook.

Slide 3



Have you ever had to manage a large number of servers that were almost identical?

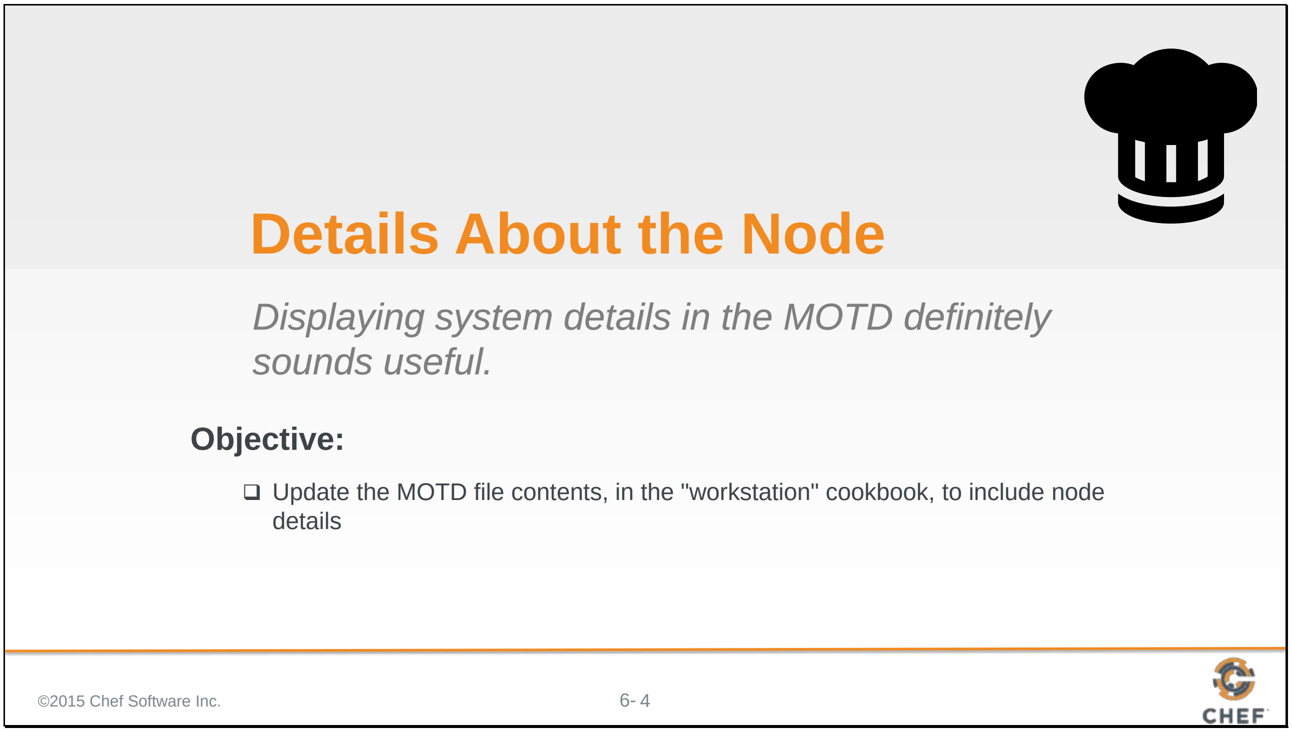
How about a large number of identical servers except that each one had to have host-specific information in a configuration file?

The file needed to have the hostname or the IP address of the system.

Maybe you needed to allocate two-thirds of available system memory into HugePages for a database. Perhaps you needed to set your thread max to number of CPUs minus one.

The uniqueness of each system required you to define custom configuration files. Custom configurations that you need to manage by hand.

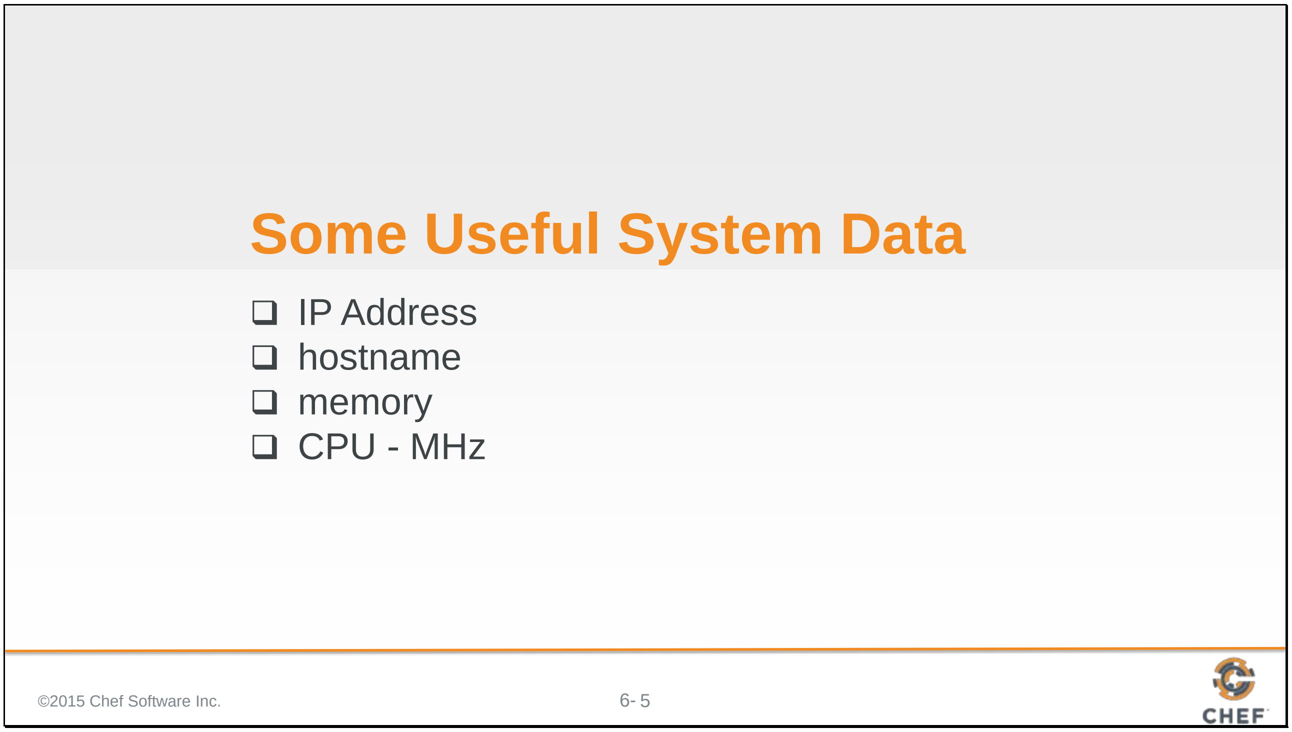
Slide 4



Here we've been given the simple request of providing some additional details about our node in both our Message of the Day and our default index page that we deploy with our web server.

We'll start first with our message of the day.

Slide 5

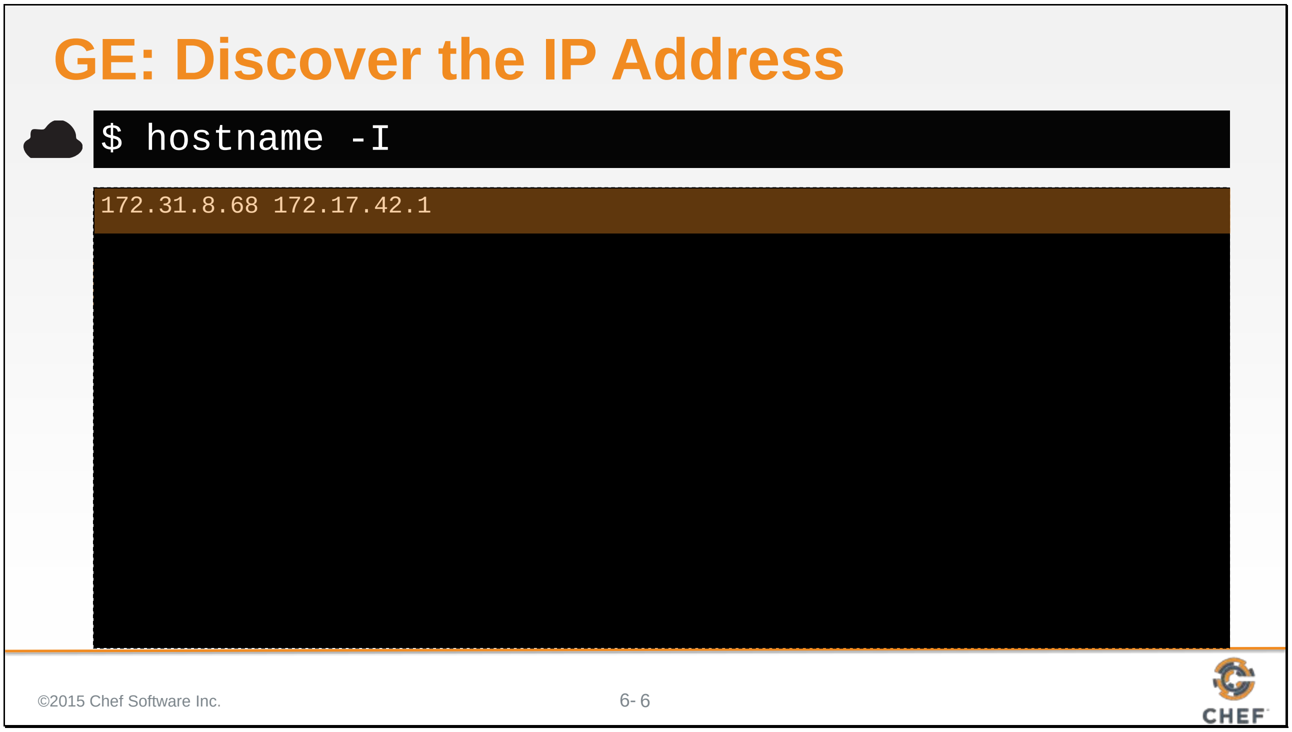


Thinking about some of the scenarios that we mentioned at the start of the session makes us think that it would be useful to capture:

The IP address, hostname, memory, and CPU megahertz of our current system.

We'll walk through capturing that information using various system commands starting with the IP address.

Slide 6



To discover the IP address of the node, we can issue the command

`hostname -I`

Slide 7

GE: Discover the IP Address

```
$ ifconfig
```

```
docker0   Link encap:Ethernet  HWaddr 56:84:7A:FE:97:99
          inet addr:172.17.42.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::5484:7aff:fefe:9799/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25870 errors:0 dropped:0 overruns:0 frame:0
          TX packets:128971 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1459392 (1.3 MiB)  TX bytes:190819384 (181.9 MiB)

eth0      Link encap:Ethernet  HWaddr 0A:4D:03:F7:91:D7
          inet addr:172.31.8.68  Bcast:172.31.15.255  Mask:255.255.240.0
          inet6 addr: fe80::84d:3ff:fef7:91d7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```


CHEF

Or you can dig it out of the results of running `ifconfig`.

Slide 8

GE: Add the IP Address

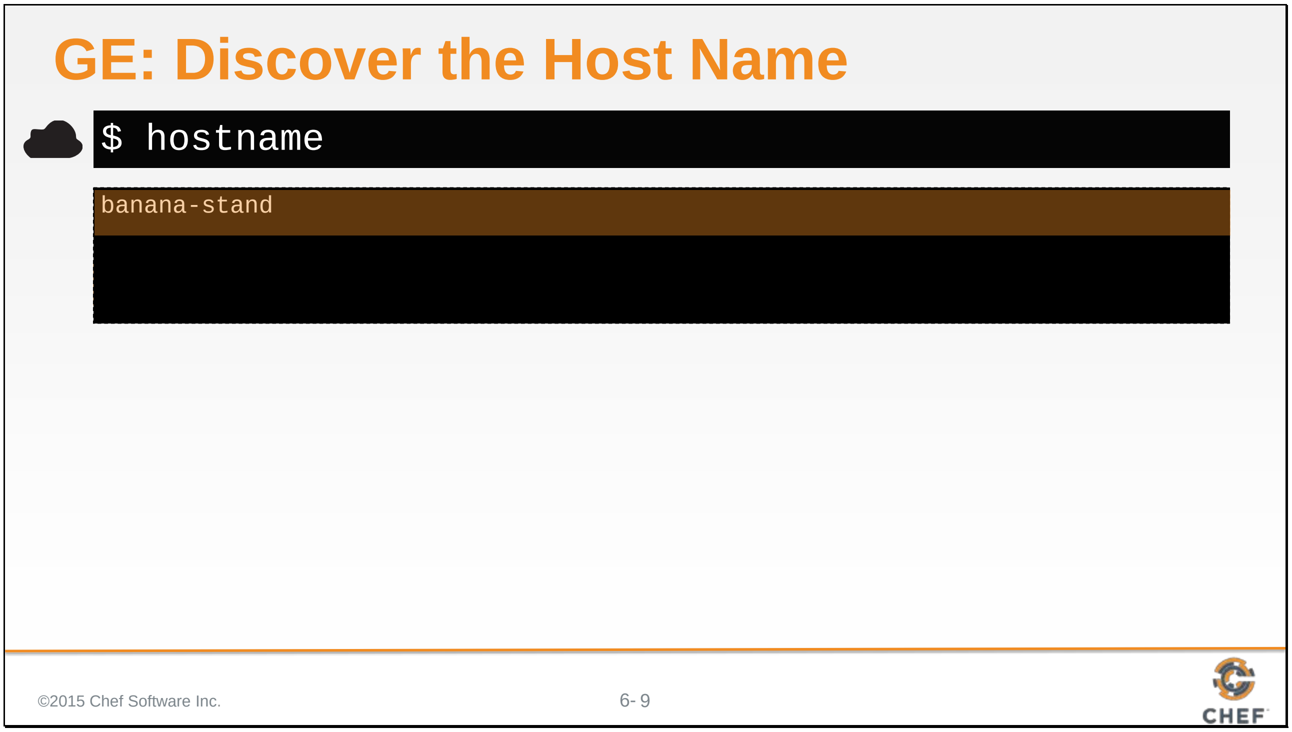
~/cookbooks/workstation/recipes/setup.rb

```
file "/etc/motd" do
  content "Property of ...

  IPADDRESS: 104.236.192.102

"
  mode "0644"
  owner "root"
  group "root"
end
```


You can include this information in our "/etc/motd by updating the contents of the file resource's content attribute.

Within the existing string value we've inserted a number of new lines for formatting and placed our IP address along with its value.

Slide 9



Next is the machine's hostname. This is easily retrievable with the `hostname` command.

Note: The host name of your virtual workstation may simply be an IP address. For example, "ip-172-31-2-14x".

Slide 10

GE: Adding the Host Name

~/cookbooks/workstation/recipes/setup.rb

```
file "/etc/motd" do
  content "Property of ...

  IPADDRESS: 104.236.192.102
  HOSTNAME: banana-stand

"
  mode "0644"
  owner "root"
  group "root"
end
```

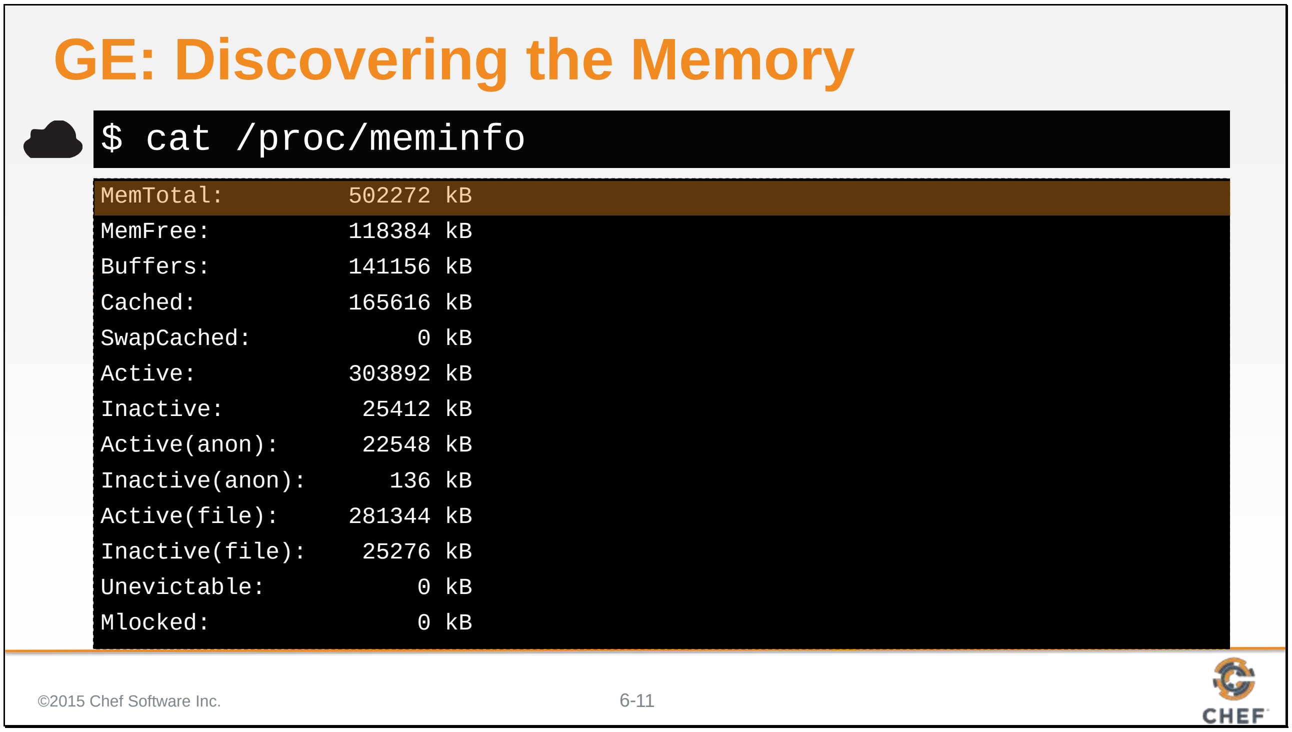

CHEF

We can also include this information in the file resource's attribute on a new line below our IP address.

Your setup.rb file's might end up looking like the following depending on how the host name was defined on the virtual workstation:

IPADDRESS: 104.236.192.102
HOSTNAME : ip-104-236-192-102

Slide 11



One way to gather the memory of our system is to `cat` the contents of the /proc/meminfo. There we can select the total memory available on the system.

Slide 12

GE: Adding the Memory

~/cookbooks/workstation/recipes/setup.rb

```
file "/etc/motd" do
  content "Property of ...

  IPADDRESS: 104.236.192.102
  HOSTNAME : banana-stand
  MEMORY   : 502272 kB

"
  mode "0644"
  owner "root"
  group "root"
end
```


CHEF

And again, add it in the file resource's attribute below your hostname.

Slide 13

GE: Discover the CPU - MHz

```
$ cat /proc/cpuinfo
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40GHz
stepping        : 4
cpu MHz         : 2399.998
cache size      : 15360 KB
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36
```


Discovering information about the system's CPU is very similar. We can `cat` the contents of /proc/cpuinfo and select the CPU megahertz from the results.

Slide 14

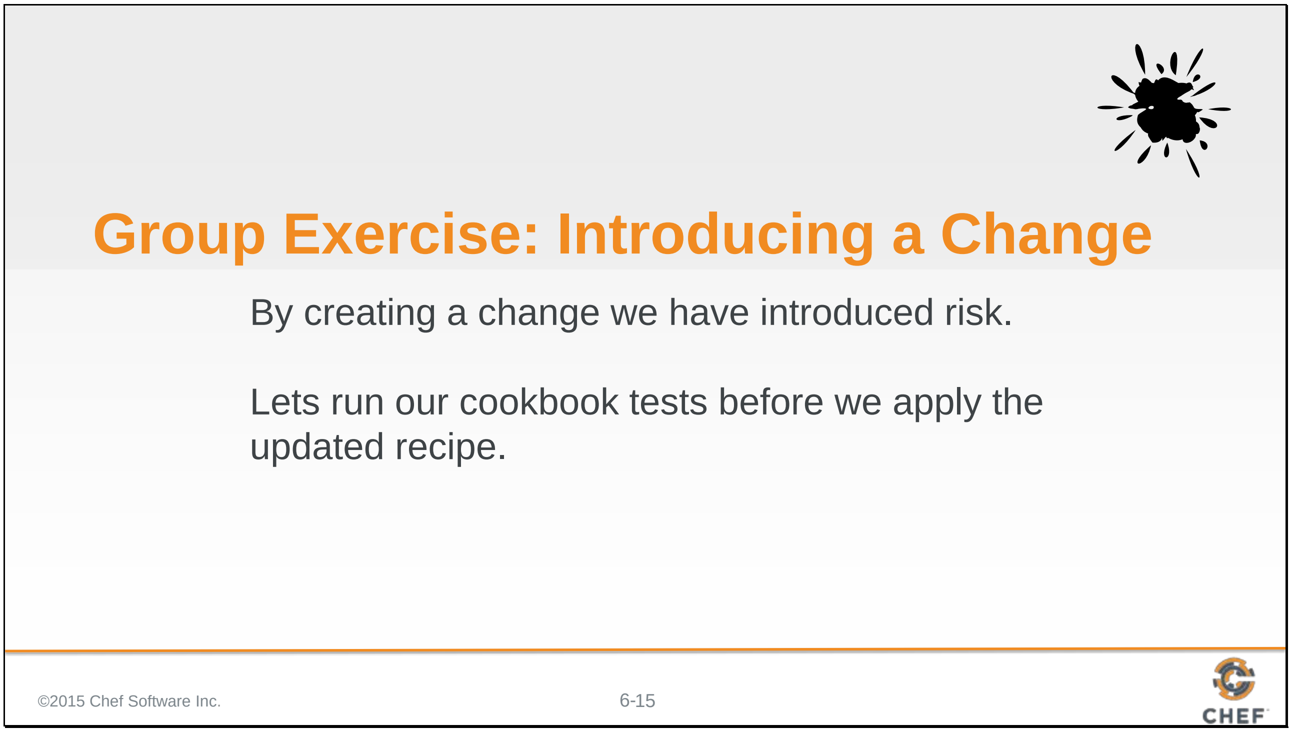
GE: Adding the CPU

~/cookbooks/workstation/recipes/setup.rb

```
file "/etc/motd" do
  content "Property of ...

  IPADDRESS: 104.236.192.102
  HOSTNAME : banana-stand
  MEMORY   : 502272 kB
  CPU      : 2399.998 MHz
"
  mode "0644"
  owner "root"
  group "root"
end
```


Add the CPU information to the file resource's content attribute.

Slide 15



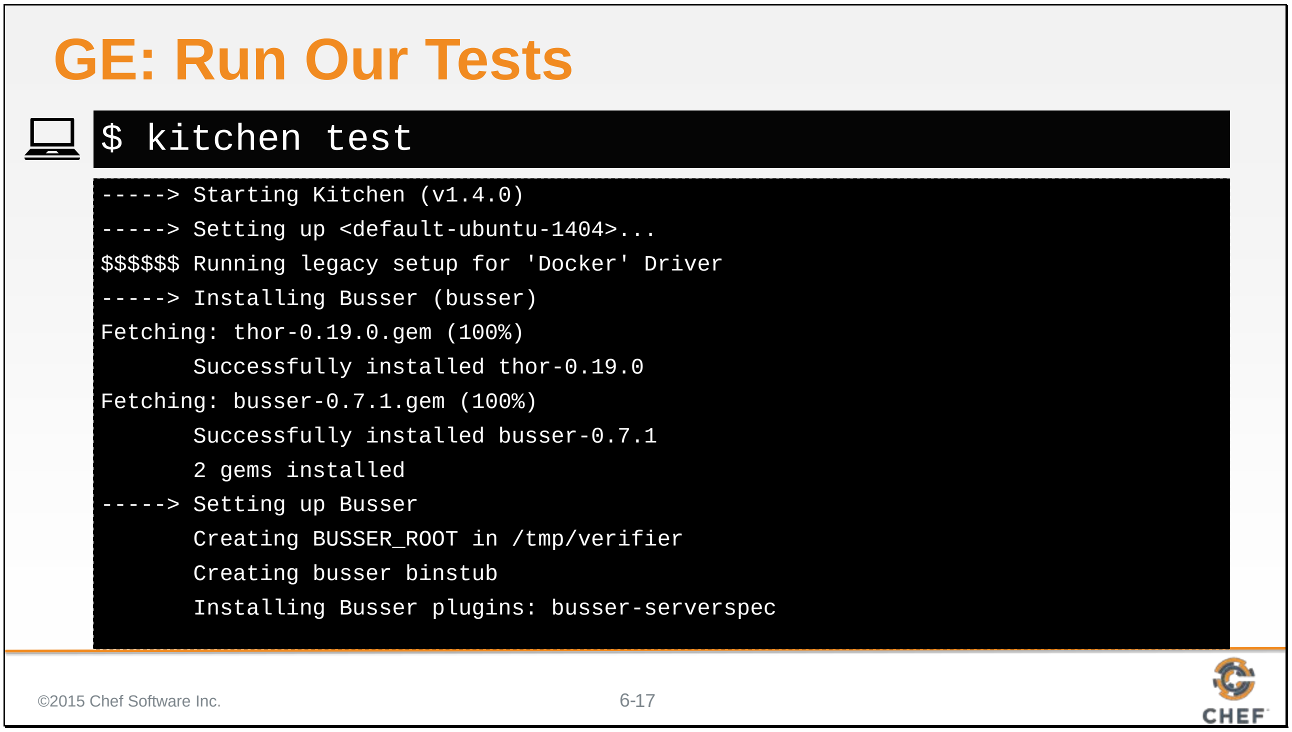
By updating the file resource we have introduced a change to the cookbook and introduced a risk. This change may not work. It could be a typo when transcribed from the slide, or the code that we have provided you may be out-of-date, or very possibly, incorrect.

Before we apply the updated recipe we can use testing to ensure the recipe is correctly defined.

Slide 16

Remember, we are testing a specific cookbook with kitchen so we need to be within the directory of the cookbook. So change directory into the workstation cookbook's directory.

Slide 17



We have not defined any new tests related to the content changes of the /etc/motd. So running the tests will tell us if we have accidentally broken any of the existing functionality but there is nothing testing the new functionality that we added.

Slide 18

GE: Return Home and Apply workstation Cookbook

```
$ cd ~
$ sudo chef-client --local-mode -r "recipe[workstation]"
```

```
resolving cookbooks for run list: ["workstation"]
Synchronizing Cookbooks:
  - workstation
Compiling Cookbooks...
Converging 6 resources
Recipe: workstation::setup
  * yum_package[nano] action install (up to date)
  * yum_package[vim] action install (up to date)
  * yum_package[emacs] action install (up to date)
  * yum_package[tree] action install (up to date)
  * yum_package[git] action install (up to date)
```


If everything looks good, then we want to use `chef-client`. `chef-client` is not run on a specific cookbook--it is a tool that allows us to apply recipes for multiple cookbooks that are stored within a cookbooks directory.

1. So we need to return home to the parent directory of all our cookbooks.

2. Then use `chef-client` to locally apply the run list defined as: the workstation cookbook's default recipe.

Slide 19

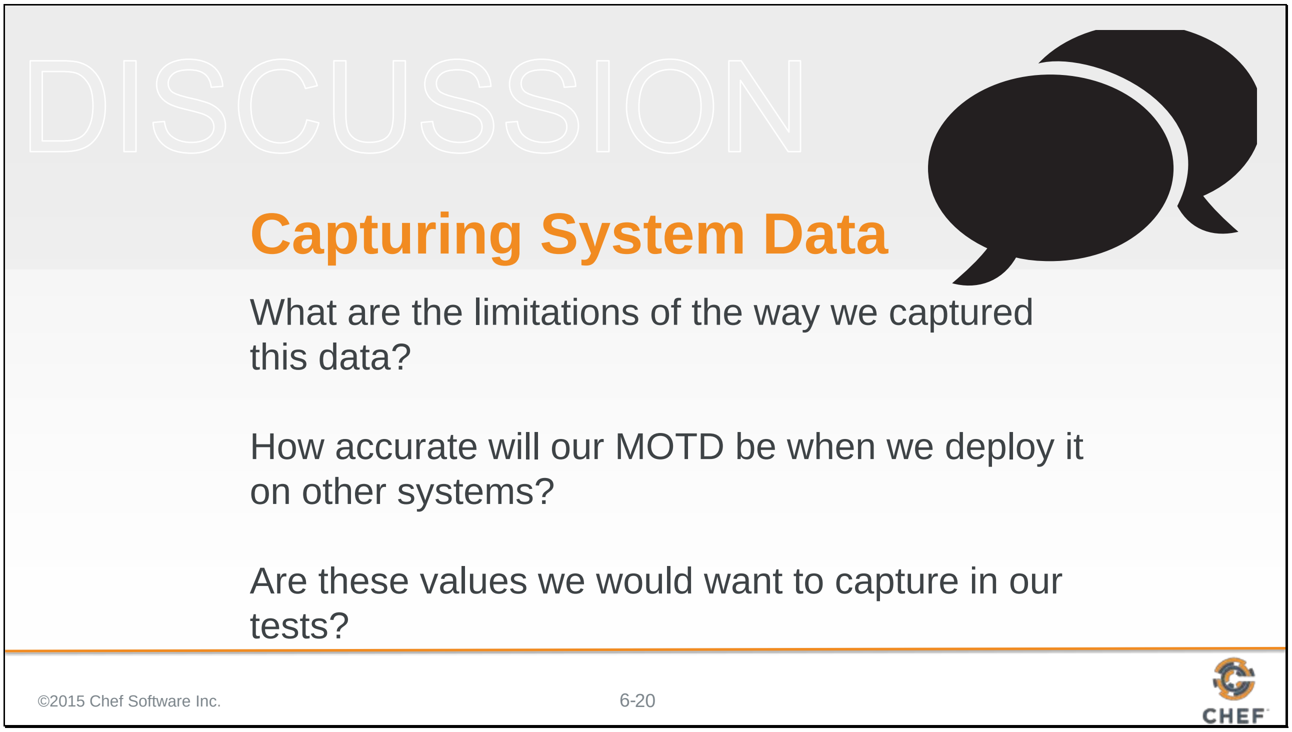
GE: Verify that the /etc/motd Has Been Updated

```
$ cat /etc/motd
```

```
Property of ...

  IPADDRESS: 172.31.8.68
  HOSTNAME : ip-172-31-8-68
  MEMORY   : 605048 kB
  CPU      : 1795.672
```


Verify that your /etc/motd had been updated with our values.

Slide 20



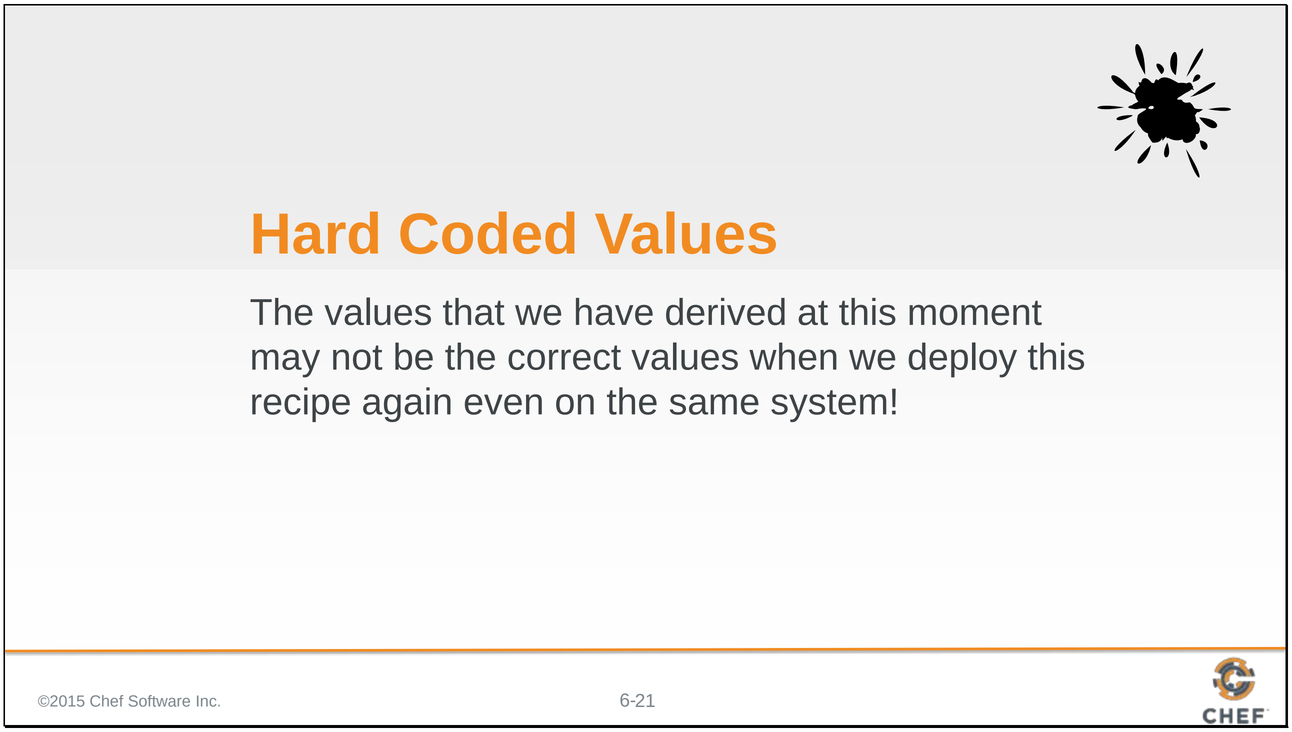
Now that we've defined these values, lets reflect:

What are the limitations of the way we captured this data?

How accurate will our MOTD be when we deploy it on other systems?

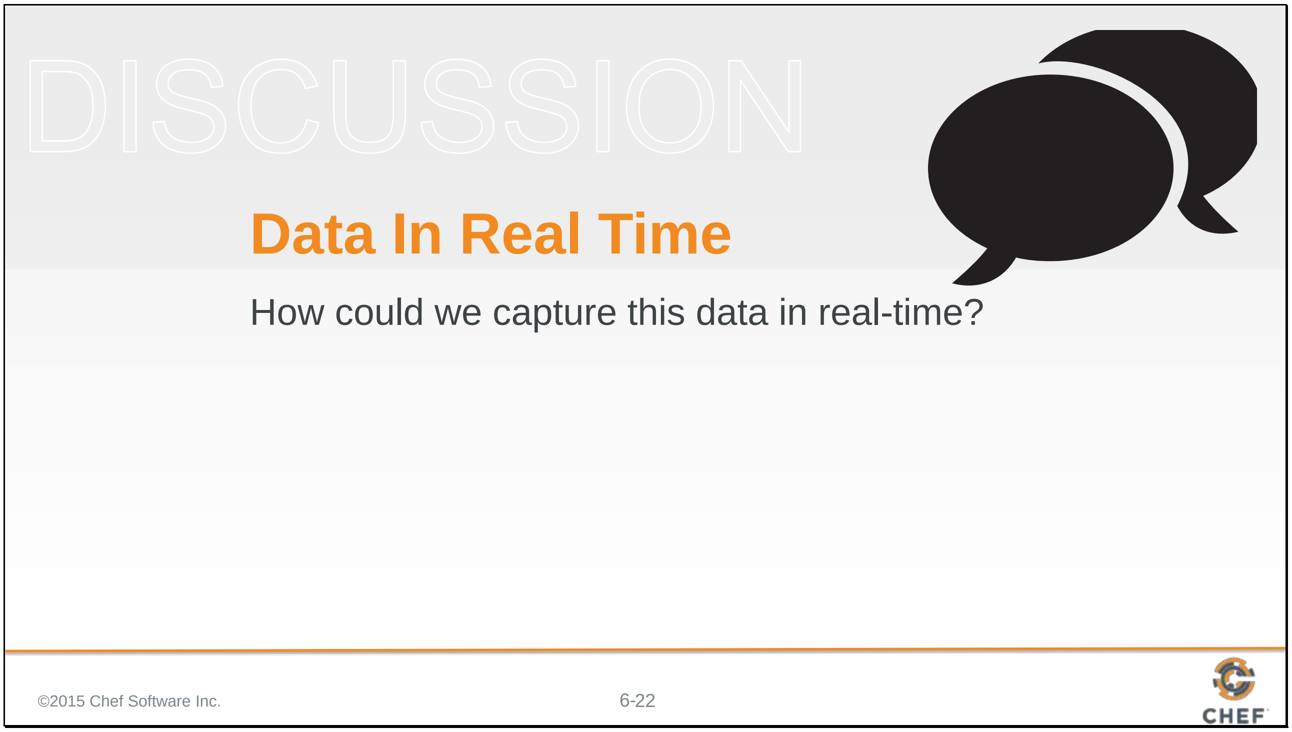
Are these values we would want to capture in our tests?

Slide 21



If you have worked with systems for a while, the general feeling is that hard-coding the values in our file resource's attribute probably is not sustainable because the results are tied specifically to this system at this moment in time.

Slide 22



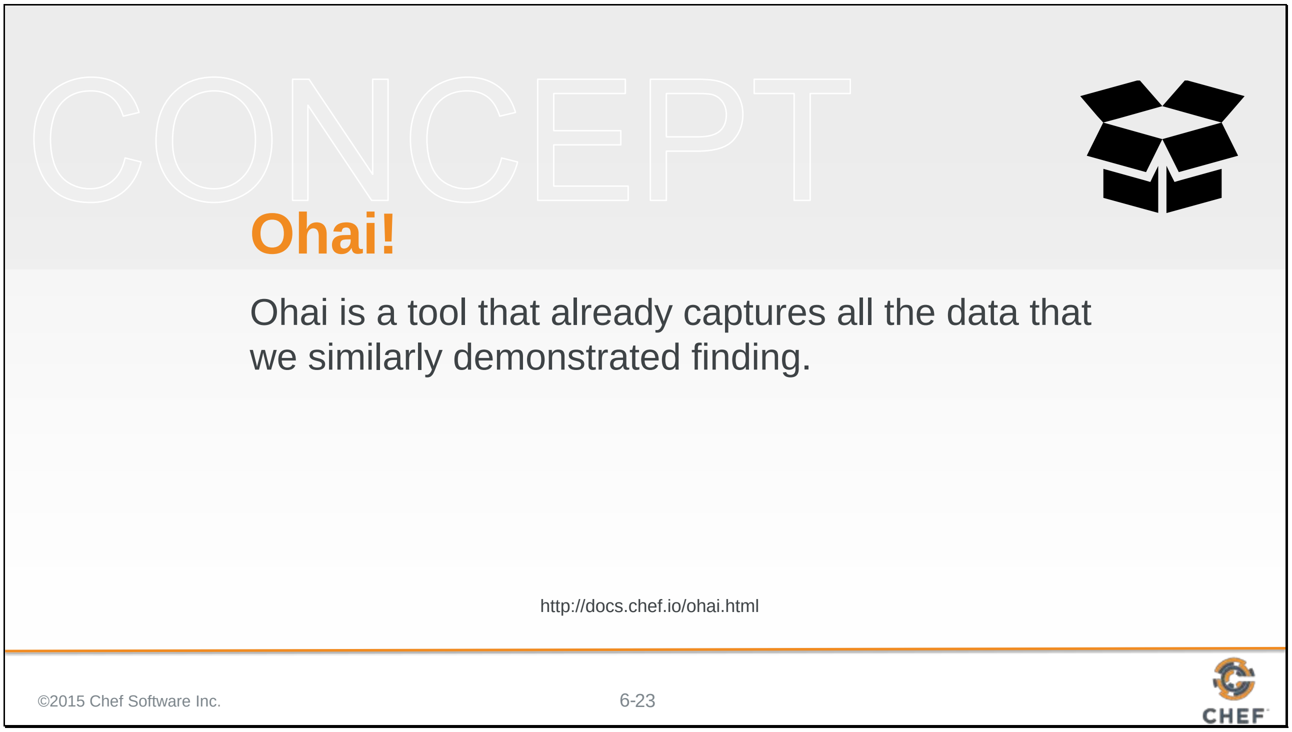
So how can we capture this data in real-time?

Capturing the data in real-time on each system is definitely possible. One way would be to execute each of these commands, parse the results, and then insert the dynamic values within the file resource's content attribute.

We could also figure out a way to run system commands within our recipes.

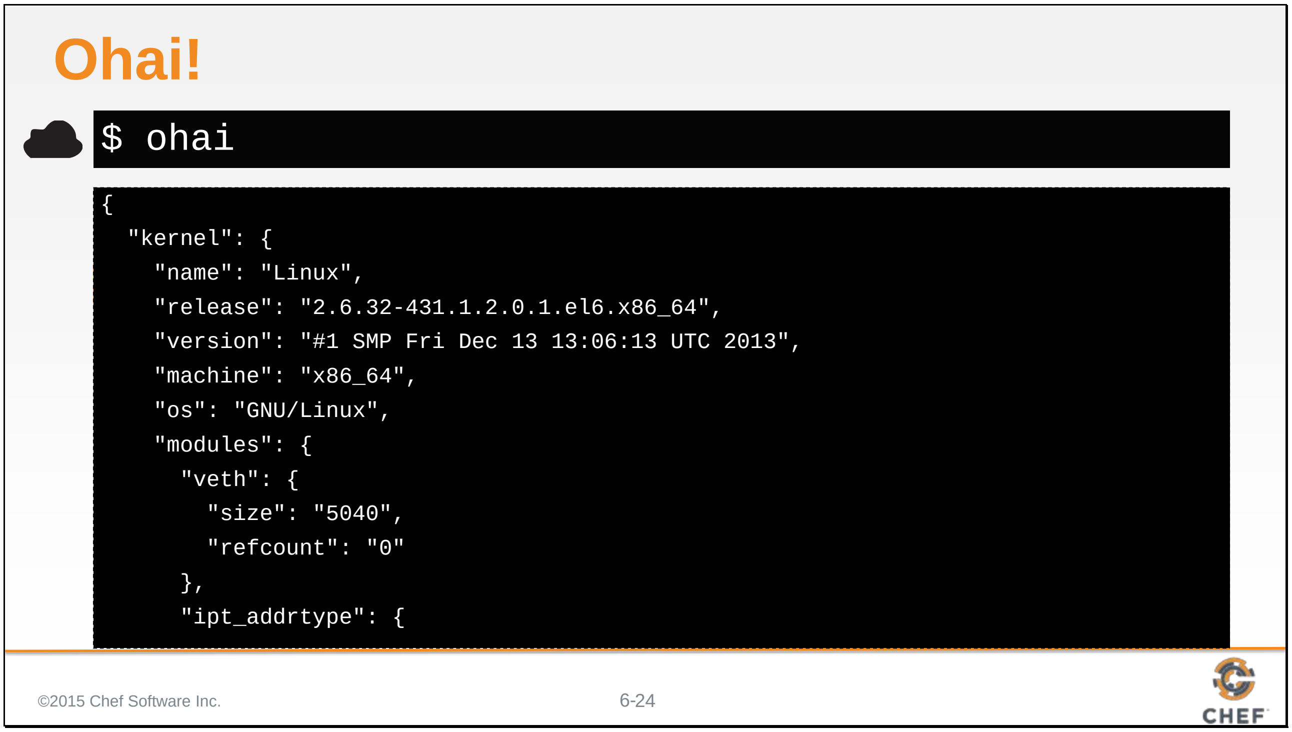
Before we start down this path, we'd like to introduce you to Ohai.

Slide 23



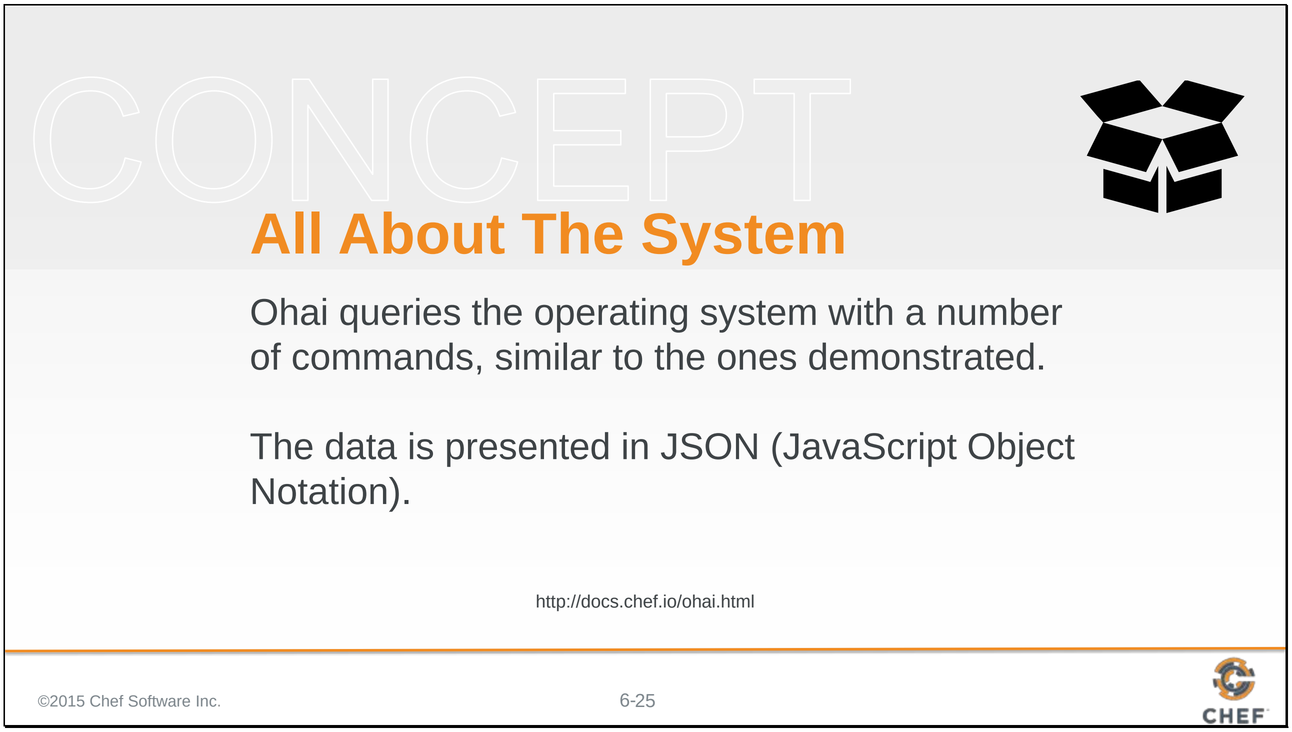
Ohai is a tool that detects and captures attributes about our system. Attributes like the ones we spent our time capturing already.

Slide 24



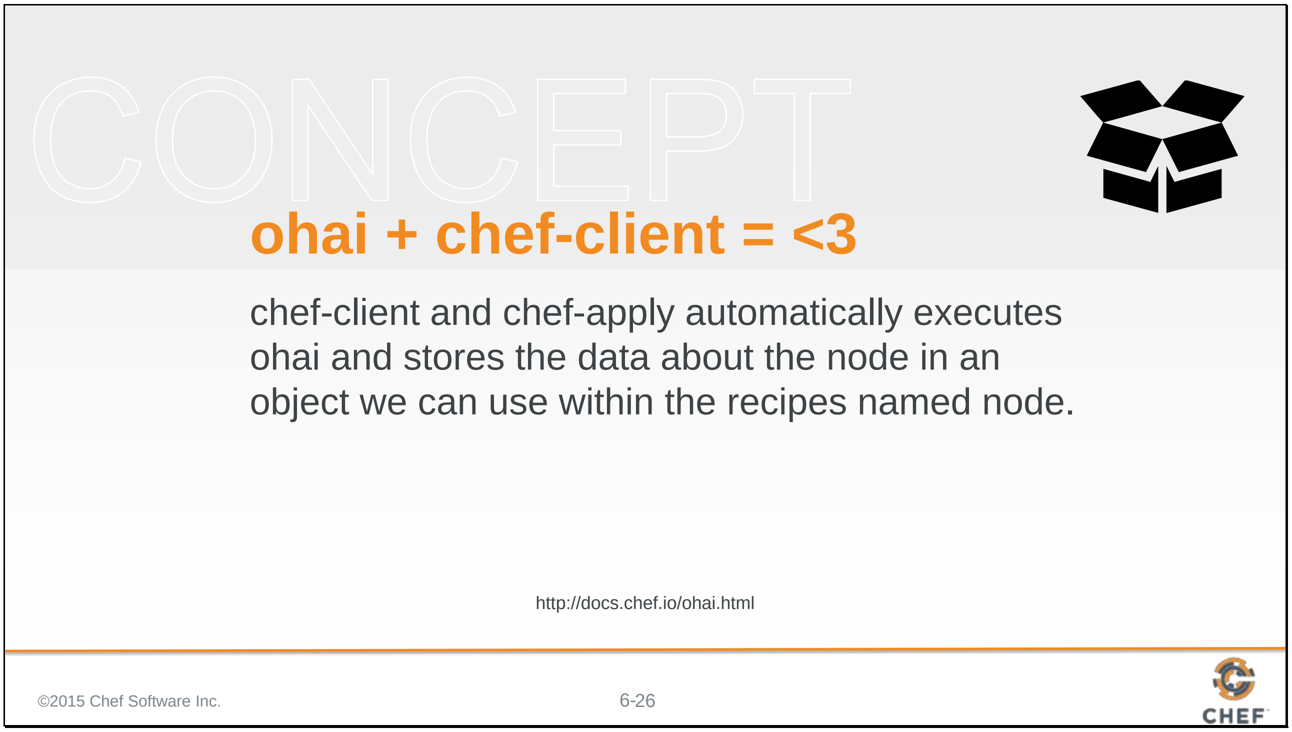
Ohai is also a command-line application that is part of the ChefDK.

Slide 25



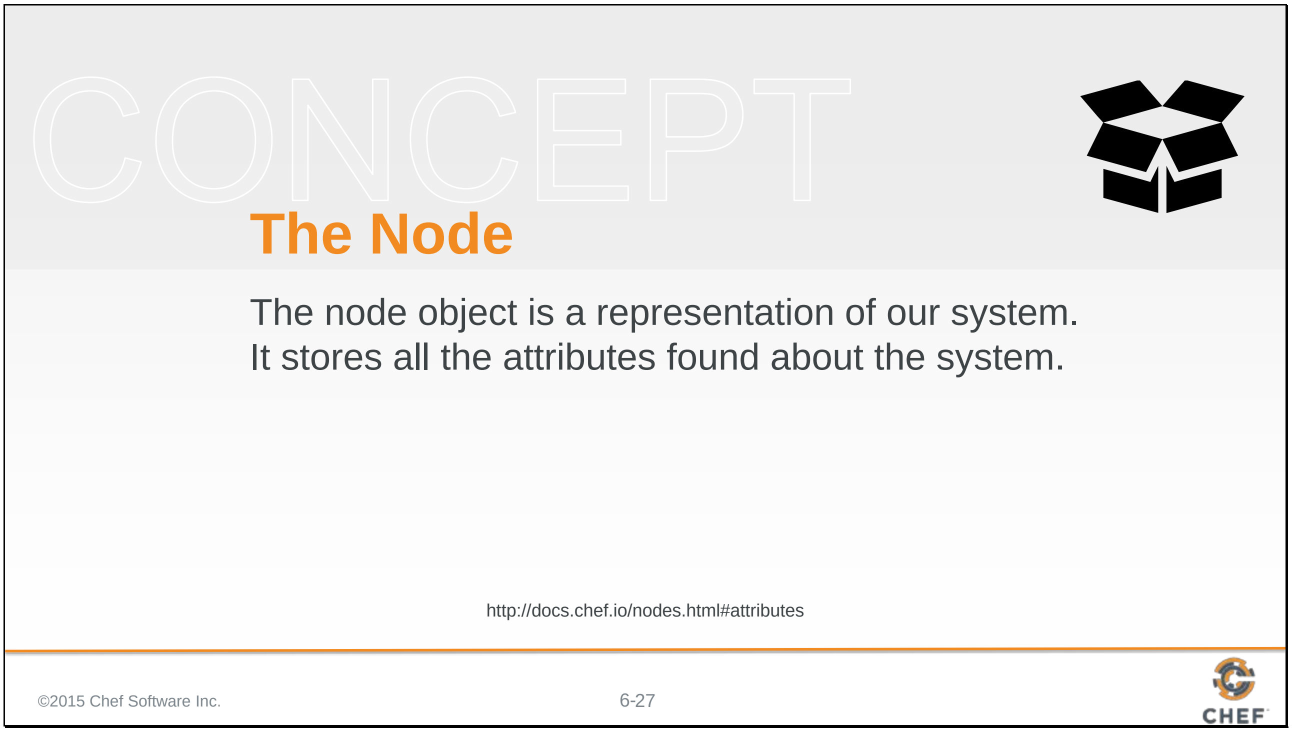
Ohai, the command-line application, will output all the system details represented in JavaScript Object Notation (JSON).

Slide 26



As we mentioned before, these values are available in our recipes because `chef-client` and `chef-apply` automatically execute Ohai. This information is stored within a variable we call 'the node object'

Slide 27

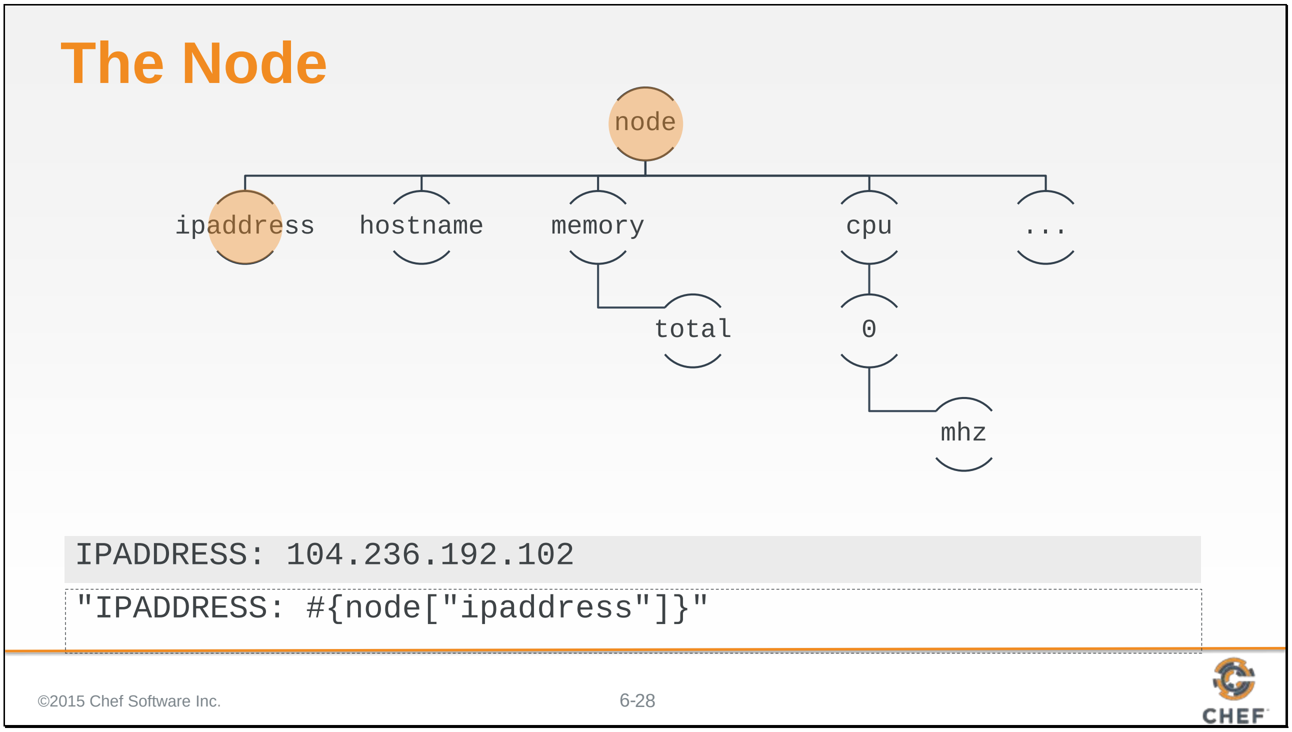


The node object is a representation of our system. It stores all these attributes found about the system. It is available within all the recipes that we write to assist us with solving the similar problems we outlined at the start.

An attribute is a specific detail about a node, such as an IP address, a host name, a list of loaded kernel modules, the version(s) of available programming languages that are available, and so on.

Lets look at using the node object to retrieve the ipaddress, hostname, total memory, and cpu megahertz.

Slide 28



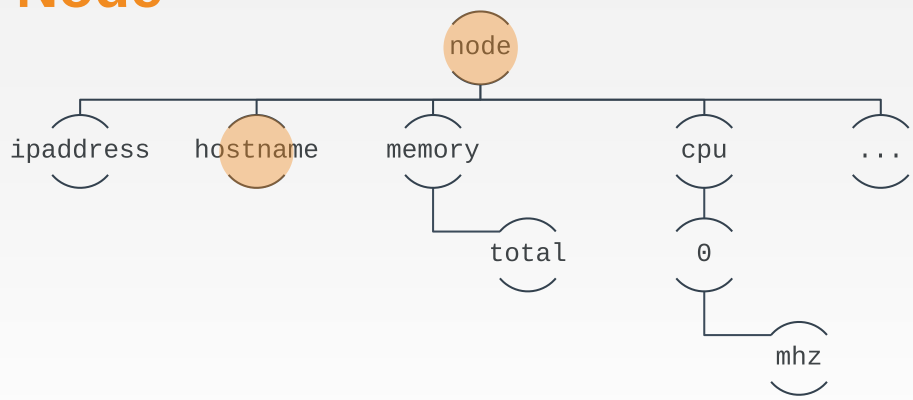
This is the visualization of the node attributes as a tree. That is done here to illustrate that the node maintains a tree of attributes that we can request from it.

The shaded text near the bottom of this slide is the hard-coded value we currently have in the file resource's content attribute.

At the very bottom is an example of how we could use the node's dynamic value within a string instead of the hard-coded one.

Slide 29

The Node

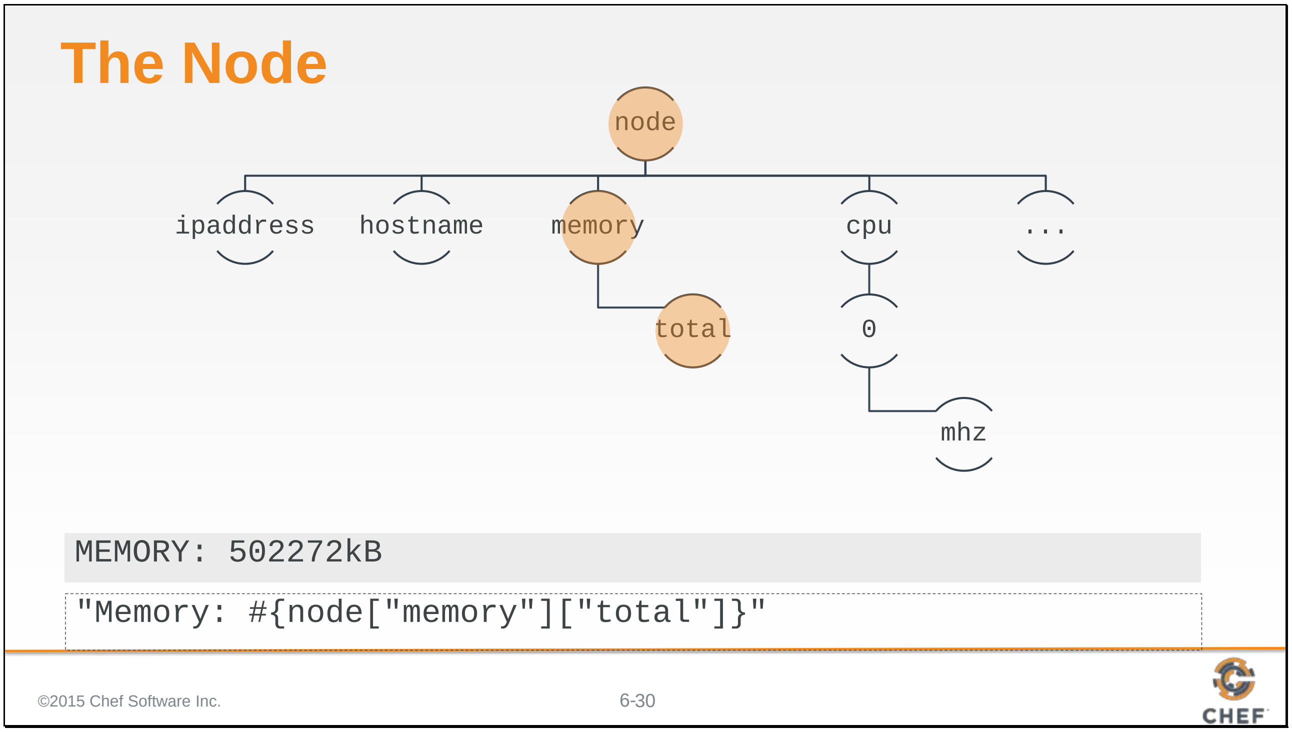



```
HOSTNAME: banana-stand
```

```
"HOSTNAME: #{node["hostname"]}"
```


The node maintains a hostname attribute. This is how we retrieve and display it.

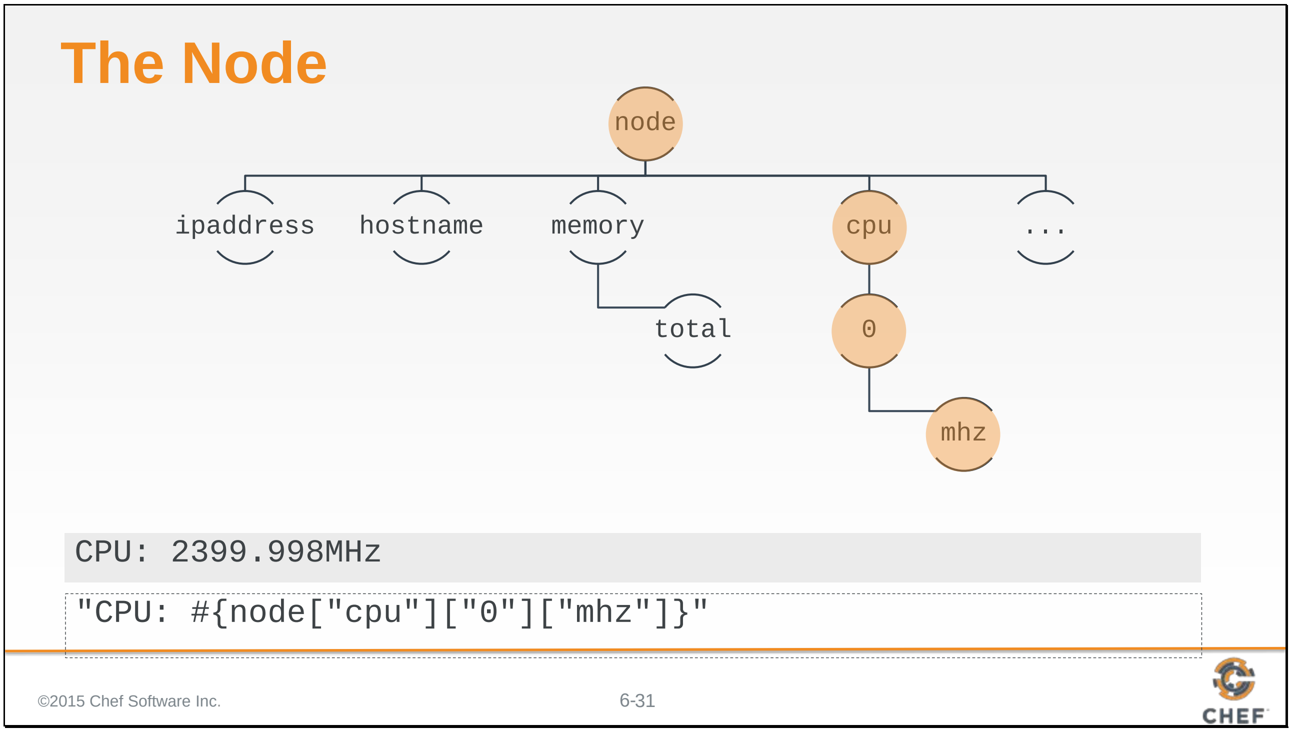
Slide 30



The node contains a top-level value memory which has a number of child elements. One of those child elements is the total amount of system memory.

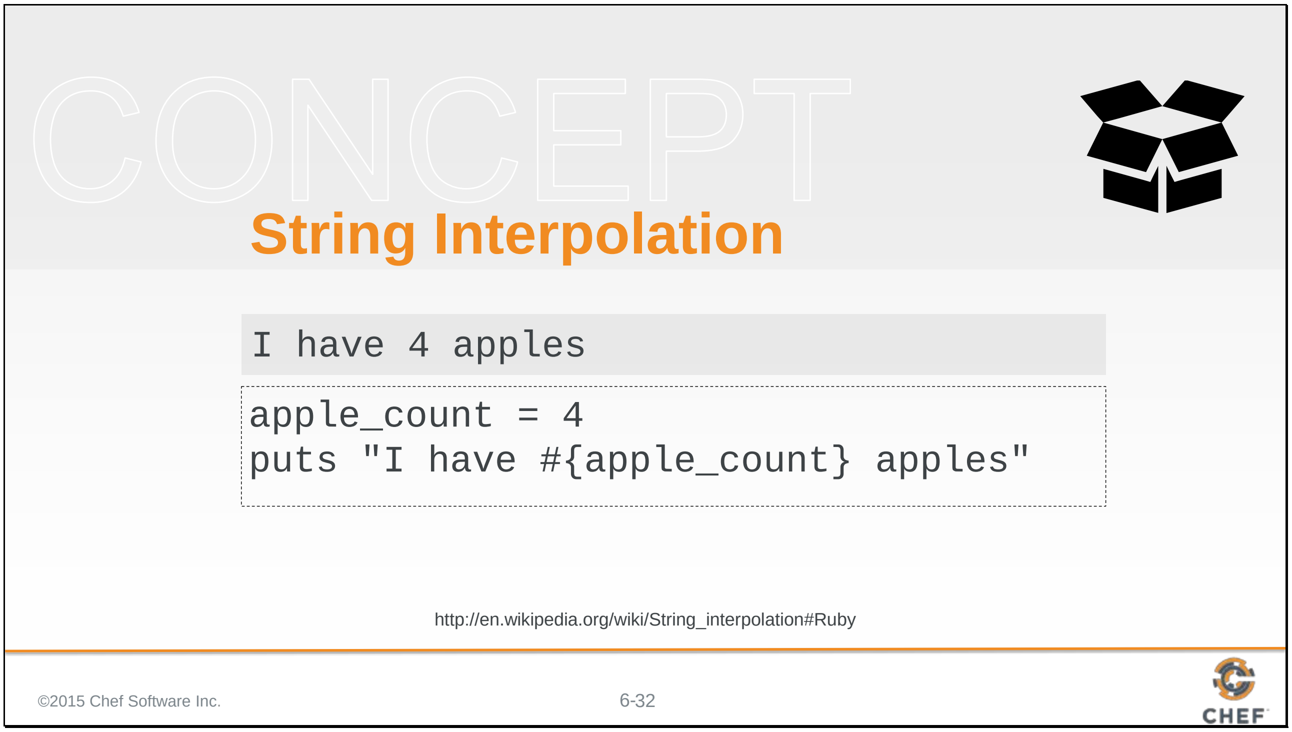
Accessing the node information is different. We retrieve the first value "memory", returning a the subset of keys and values at that level, and then immediately select to return the total value.

Slide 31



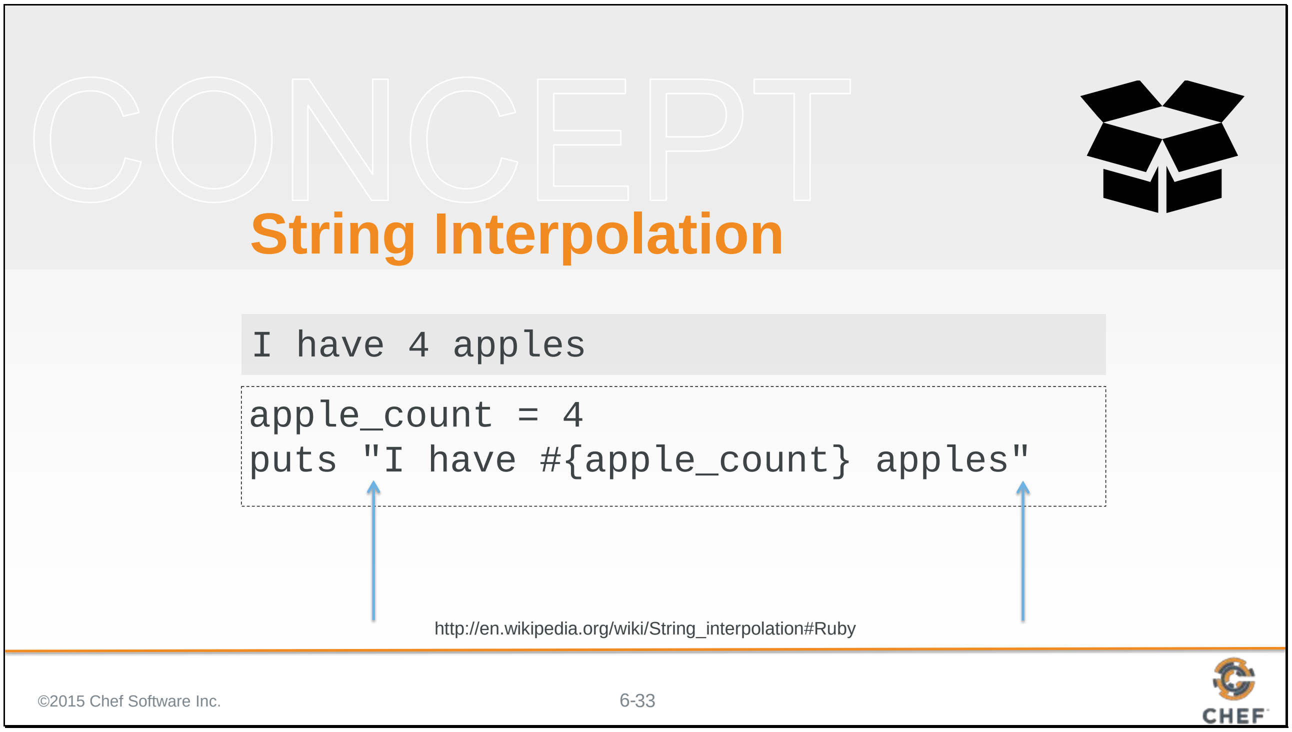
And finally, here we return the megahertz of the first CPU.

Slide 32



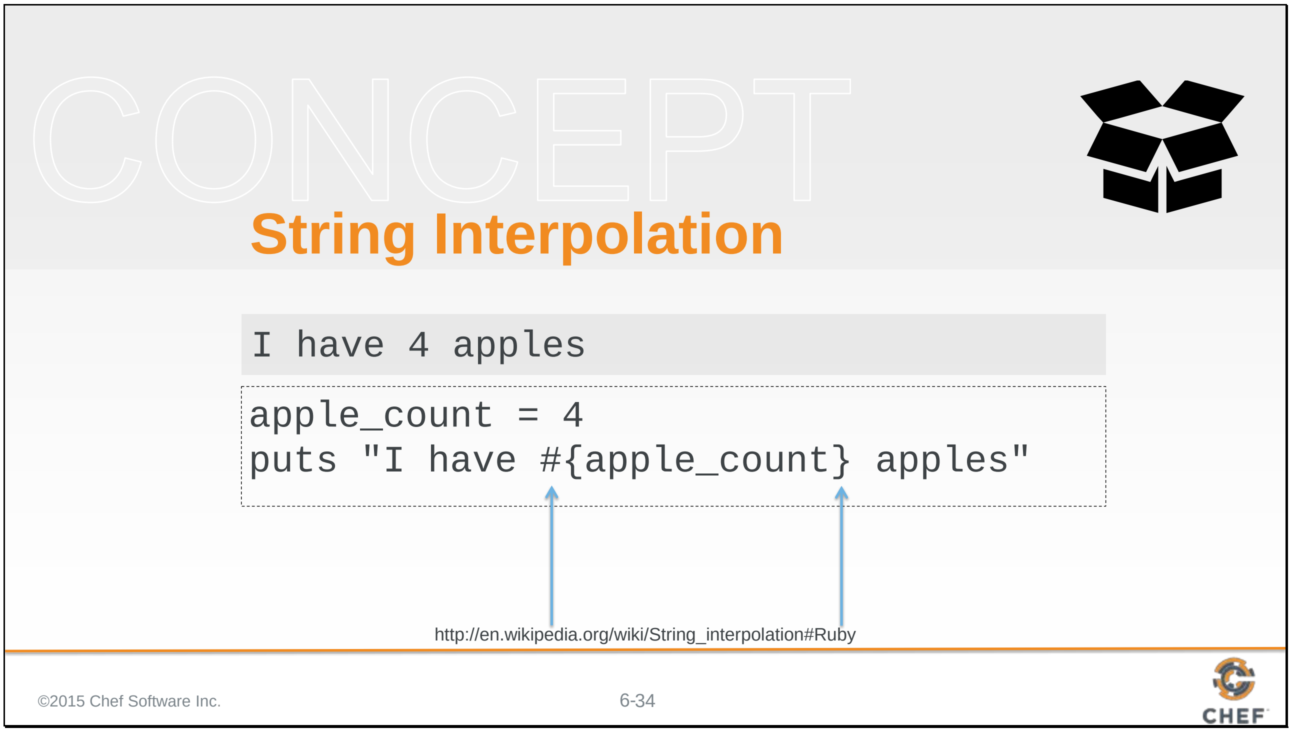
In all of the previous examples we demonstrated retrieving the values and displaying them within a string using a ruby language convention called string interpolation.

Slide 33



String interpolation is only possible with strings that start and end with double-quotes.

Slide 34



To escape out to display a ruby variable or ruby code you use the following sequence: number sign, left curly brace, the ruby variables or ruby code, and then a right curly brace.

Slide 35

GE: Using the Node's Attributes

~/cookbooks/workstation/recipes/setup.rb

```
file "/etc/motd" do
  content "Property of ...

  IPADDRESS: #{node["ipaddress"]}
  HOSTNAME : #{node["hostname"]}
  MEMORY   : #{node["memory"]["total"]}
  CPU      : #{node["cpu"]["0"]["mhz"]}
"
  mode "0644"
  owner "root"
  group "root"
end
```


In this group exercise, instead of using hard-coded values, use string interpolation within the file resource's content attribute to allow the system to access the node object's attribute for:

- IP address
- Hostname
- Total memory
- Megahertz of the first CPU.

Slide 36

GE: Verify the Changes

- ❑ Change directory into the "workstation" cookbook's directory
- ❑ Run kitchen test for the "workstation" cookbook
- ❑ Change directory into the home directory
- ❑ Run chef-client locally to verify the "workstation" cookbook's default recipe.

Again we have created a change.

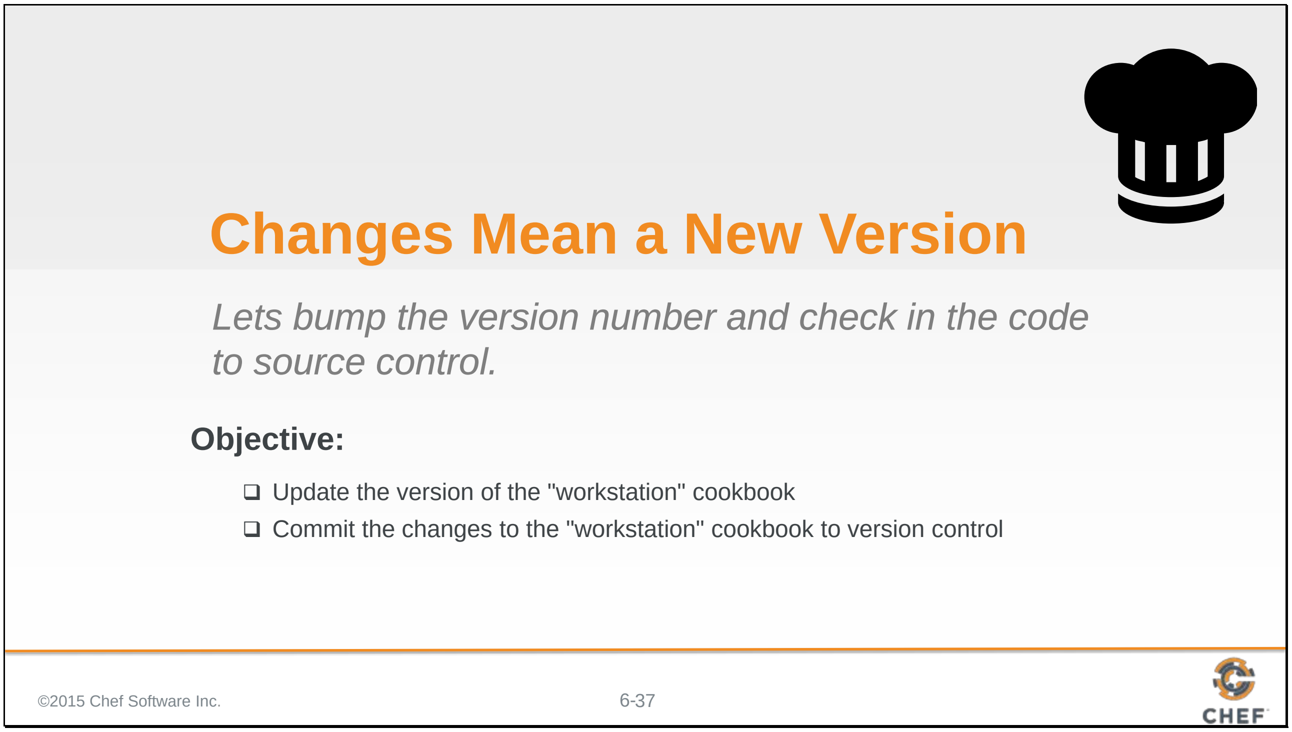
1. Move into the workstation cookbook's directory.

2. Verify the changes we made to the workstation cookbook's default recipe with kitchen.

3. Return to the home directory.

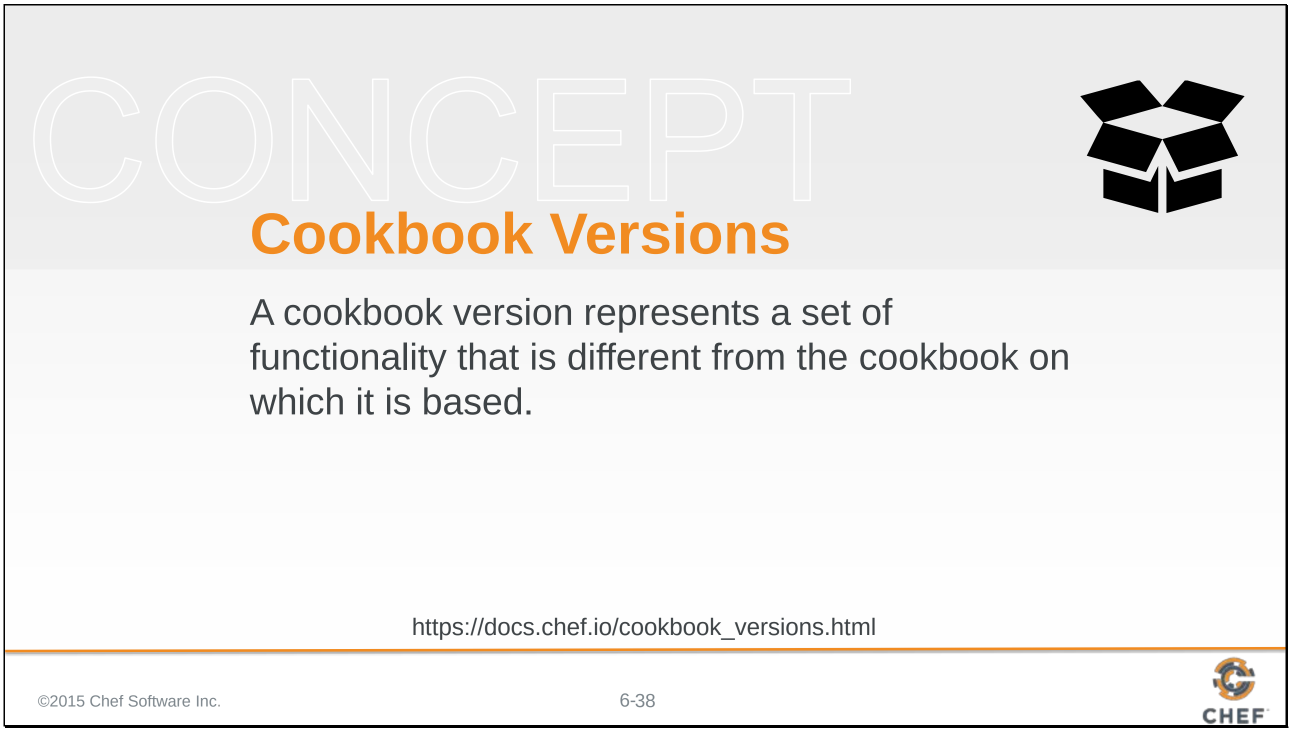
4. Use 'chef-client' to locally apply the workstation cookbook's default recipe.

Slide 37



Now that we've made these significant changes and verified that they work, its time we bumped the version of the cookbook and commit the changes.

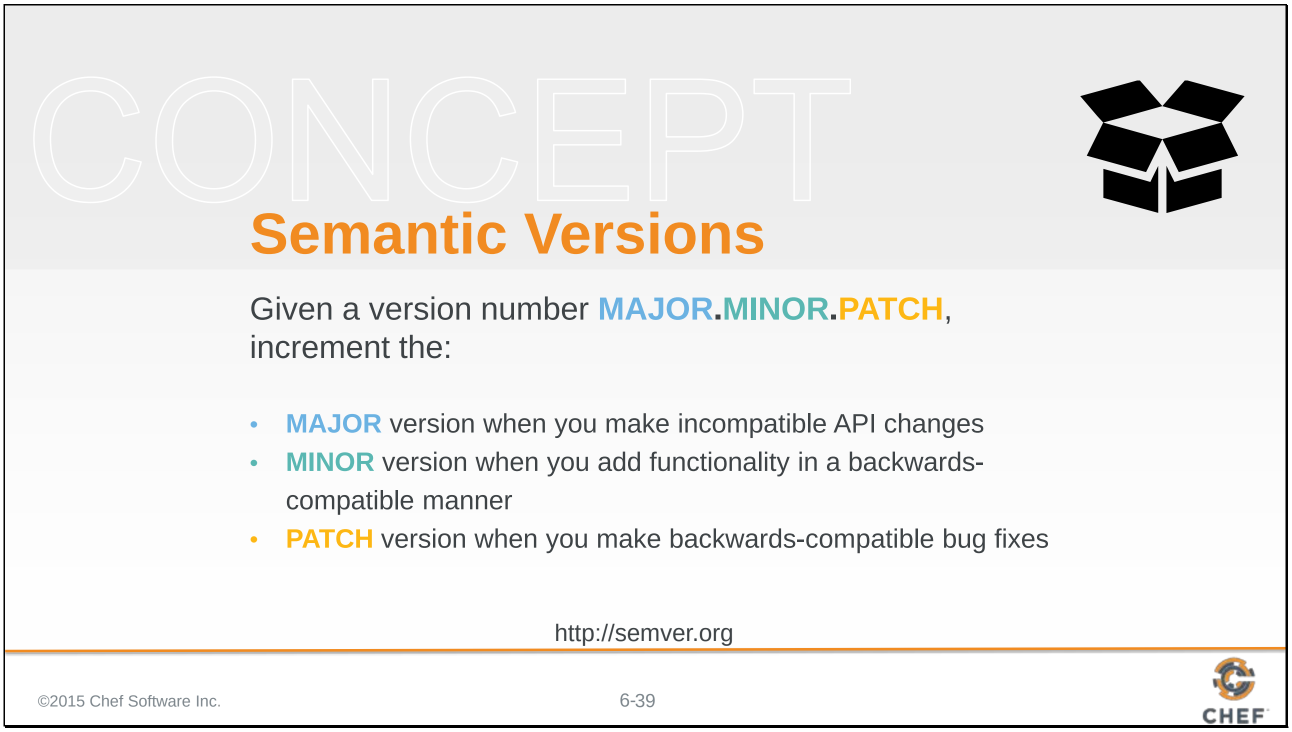
Slide 38



A version may exist for many reasons, such as ensuring the correct use of a third-party component, updating a bug fix, or adding an improvement.

The first version of the cookbook displayed a simple property message in the /etc/motd. The changes that we finished are new features of the cookbook.

Slide 39



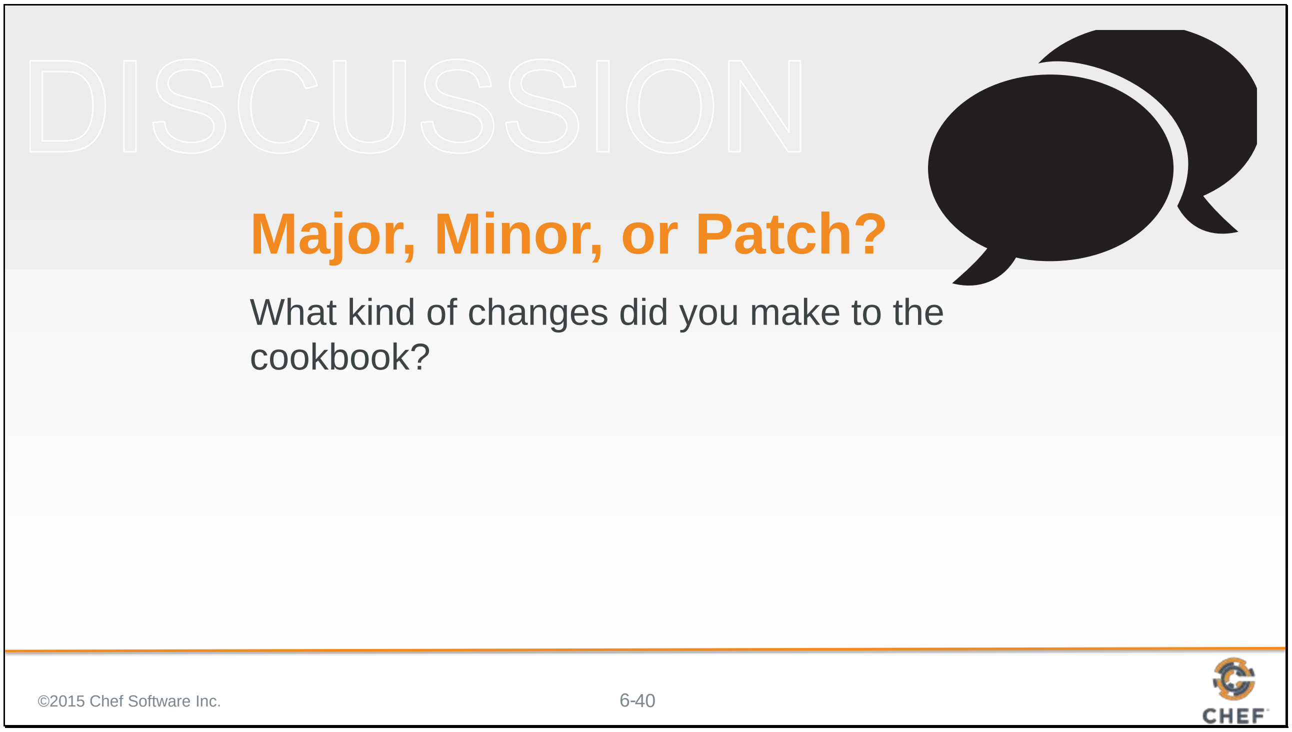
Cookbooks use semantic version. The version number helps represent the state or feature set of the cookbook. Semantic versioning allows us three fields to describe our changes: major; minor; and patch.

Major versions are often large rewrites or large changes that have the potential to not be backwards compatible with previous versions. This might mean adding support for a new platform or a fundamental change to what the cookbook accomplishes.

Minor versions represent smaller changes that are still compatible with previous versions. This could be new features that extend the existing functionality without breaking any of the existing features.

And finally Patch versions describe changes like bug fixes or minor adjustments to the existing documentation.

Slide 40



So what kind of changes did you make to the cookbook? How could we best represent that in an updated version?

Slide 41

GE: Update the Cookbook Version

~/cookbooks/workstation/metadata.rb

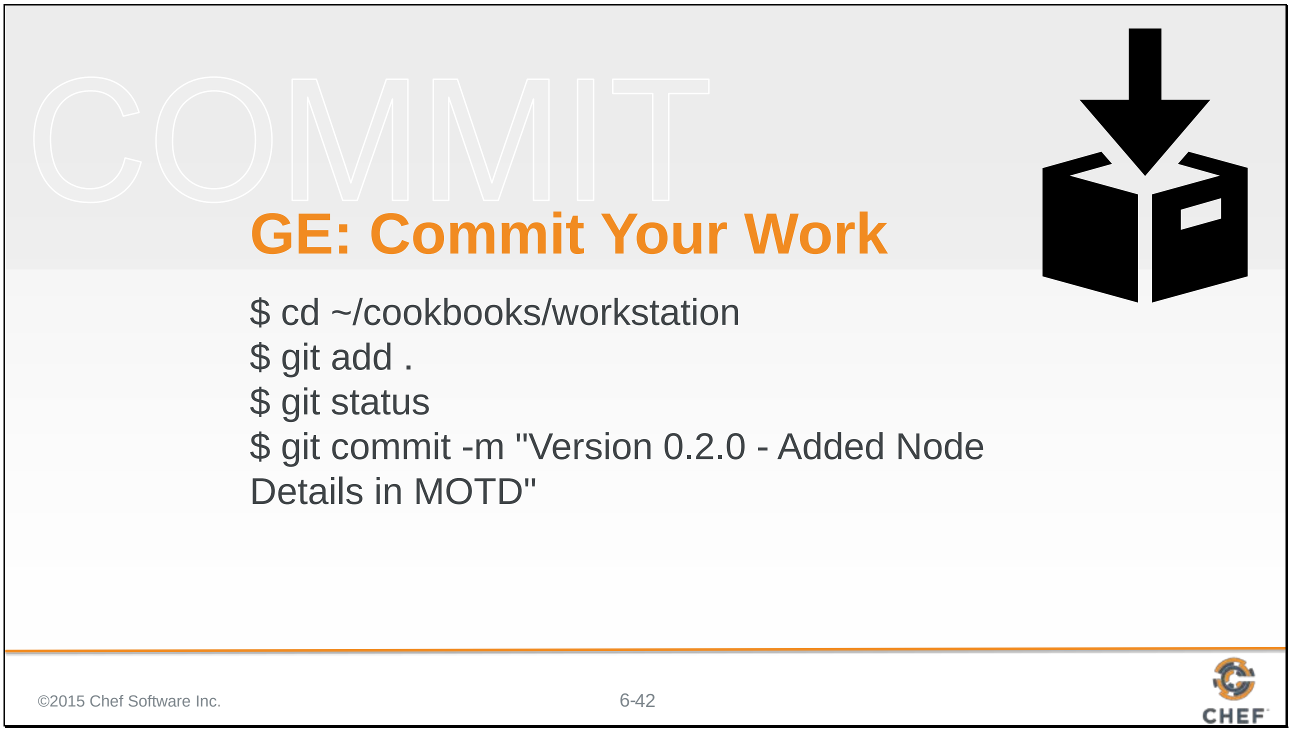
```
name             'workstation'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures workstation'
long_description 'Installs/Configures workstation'
version          '0.2.0'
```


6-41

Changing the contents of an existing resource--by adding the attributes of the node doesn't seem like a bug fix and it doesn't seem like a major rewrite. It is like a new set of features while remaining backwards compatible.

Edit ~/cookbooks/workstation/metadata.rb and update the version's minor number to 0.2.0.

Slide 42



The last thing to do is commit our changes to source control. Change into the directory, add all the changed files, and then commit them.

Slide 43

Lab: Node Details in the Webserver

In this lab, the file resource named "/var/www/html/index.html" is created with the content that includes the node details:

- ipaddress
- hostname

- ❑ Run kitchen test for the "apache" cookbook
- ❑ Run chef-client to locally apply the "apache" cookbook's default recipe.
- ❑ Update the version of the "apache" cookbook
- ❑ Commit the changes

Now it's time to add similar functionality to the apache cookbook. You should try to follow the high-level steps in this slide to complete this lab.

Slide 44

Lab: Apache Recipe

~/cookbooks/apache/recipes/server.rb

```
...

file "/var/www/html/index.html" do
  content "<h1>Hello, world!</h1>
<h2>ipaddress: #{node["ipaddress"]}</h2>
<h2>hostname: #{node["hostname"]}</h2>
"
end

...
```


Update the file resource, named "/var/www/html/index.html, to be created with the content that includes the node's IP address and its host name.

Slide 45

Lab: Test the Apache Cookbook's Default Recipe

```
$ cd cookbooks/apache
$ kitchen test
```

```
-----> Starting Kitchen (v1.4.0)
-----> Cleaning up any prior instances of <default-centos-67>
-----> Destroying <default-centos-67>...
       Finished destroying <default-centos-67> (0m0.00s).
-----> Testing <default-centos-67>
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  2.56 kB
       Sending build context to Docker daemon
       Step 0 : FROM centos:centos6
        ---> 72703a0520b7
       Step 1 : RUN yum clean all
```


Change into the apache cookbook's directory and then run `kitchen test` to verify that the changes we introduced did not cause a regression.

Slide 46

Lab: Run chef-client to Apply the Apache Cookbook

```
$ cd ~
$ sudo chef-client --local-mode -r "recipe[apache]"
```

```
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
(skipping)
* service[httpd] action enable (up to date)
* service[httpd] action start (up to date)
Running handlers:
Running handlers complete
Chef Client finished, 1/4 resources updated in 29.019528692 seconds
```


If everything passes and you feel confident that it will also work on the current workstation, change to the home directory and then run `chef-client` to apply the apache cookbook locally to the system.

Slide 47

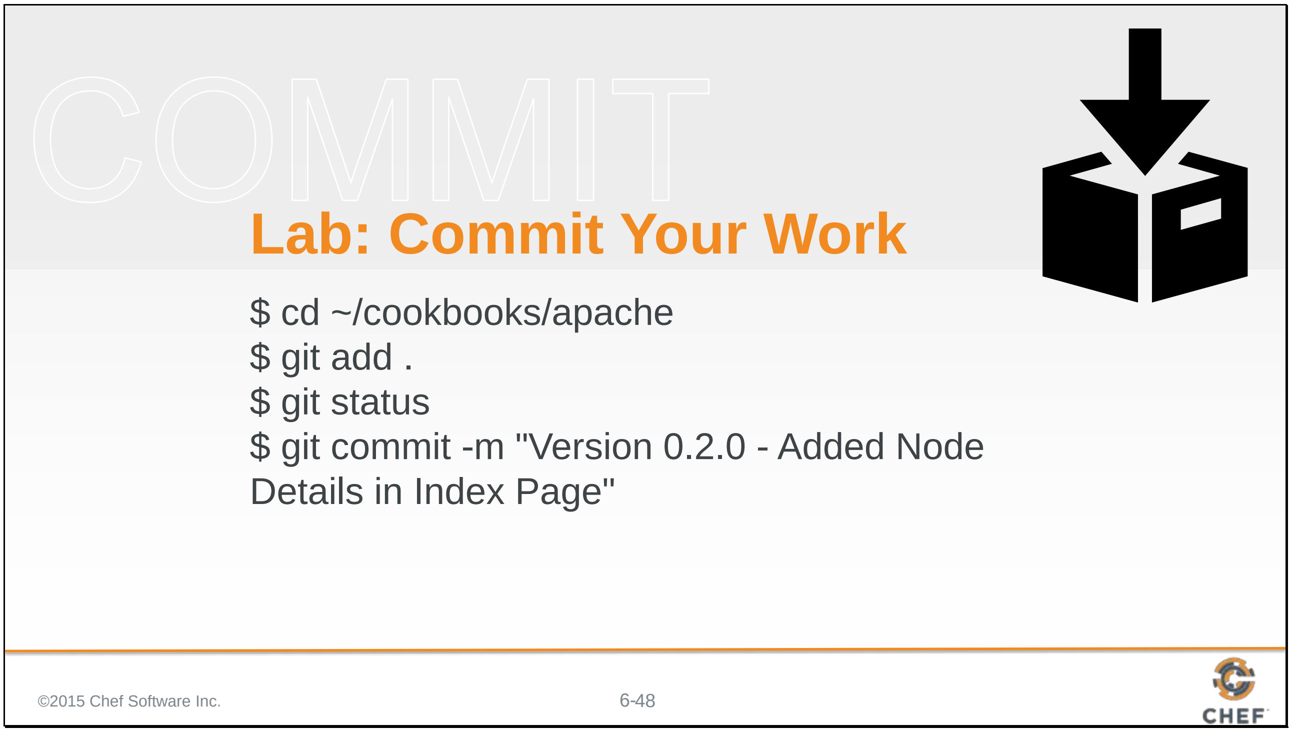
Lab: Update the Cookbook Version

~/cookbooks/apache/metadata.rb

```
name             'apache'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures apache'
long_description 'Installs/Configures apache'
version          '0.2.0'
```

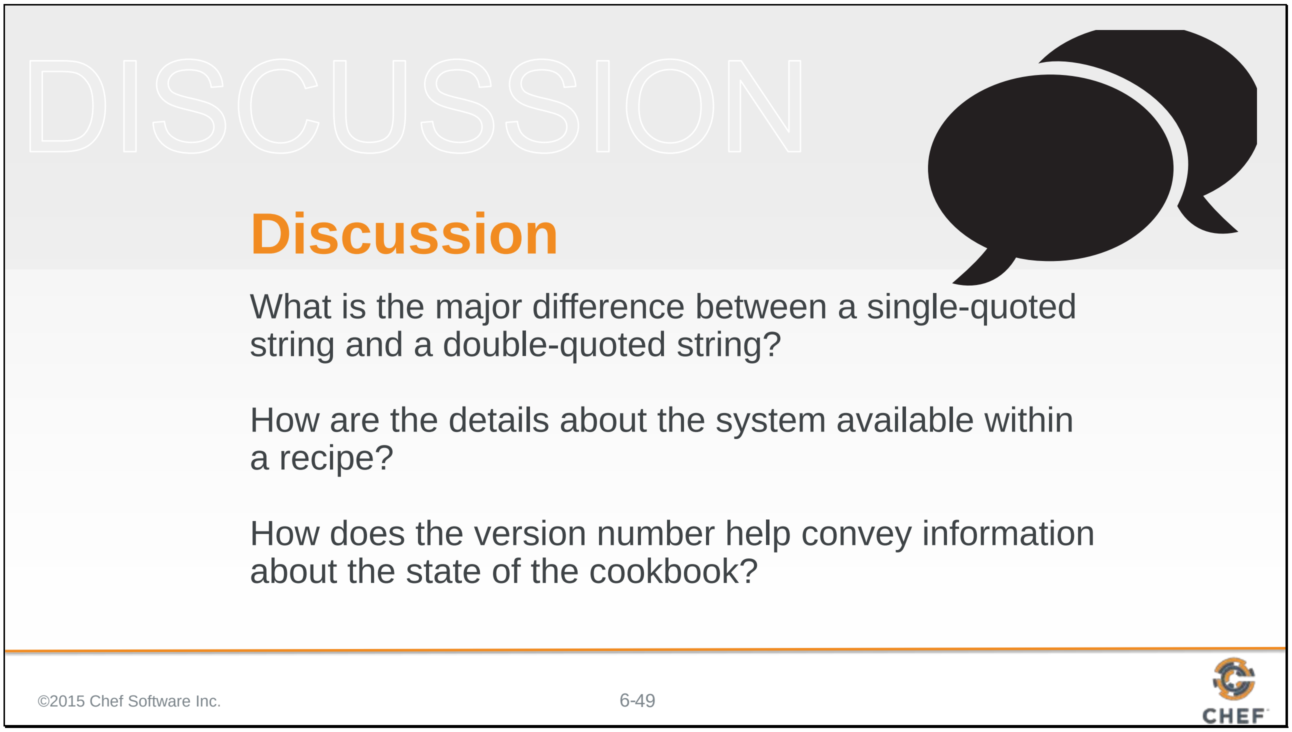

Showing these two attributes in the index html page seems very similar to the feature we added for the workstation cookbook. So update the version of the apache cookbook to 0.2.0 as well.

Slide 48



And finally, commit your changes to git.

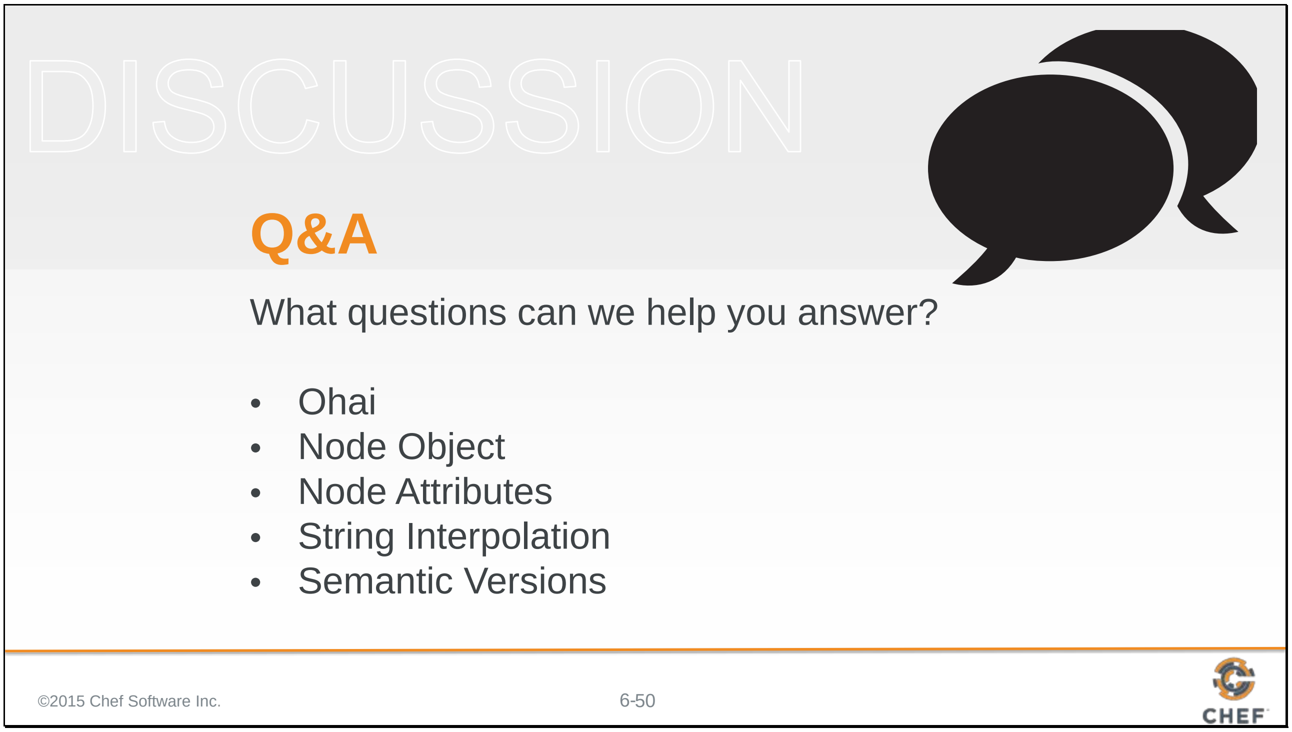
Slide 49



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 50



With that we have added all of the requested features.

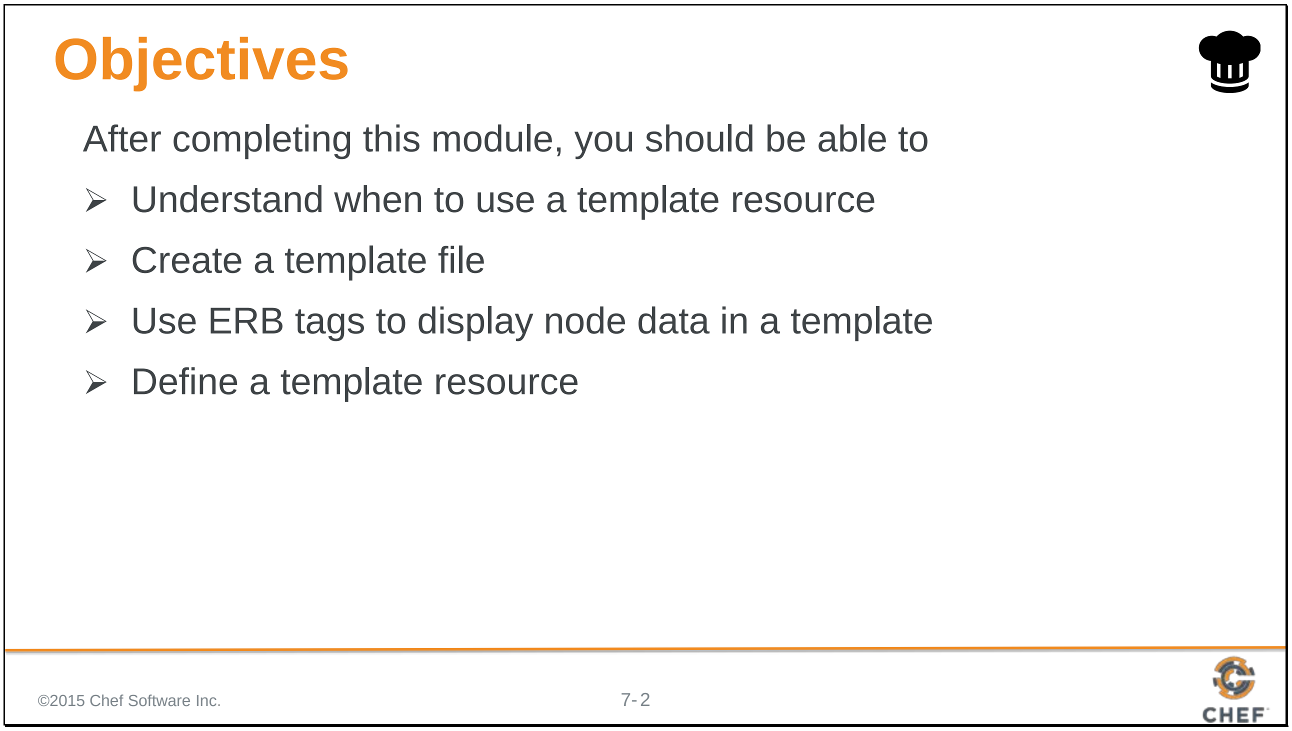
What questions can we help you answer?

In general or about specifically about ohai, the node object, node attributes, string interpolation, or semantic versioning.

Slide 51

7: Desired State and Data

Slide 2



In this module you will learn how to understand when to use a template resource, create a template file, use ERB tags to display node data in a template, define a template resource.

Slide 3

Cleaner Recipes

In the last section we updated our two cookbooks to display information about our node.

We added this content to the file resource in their respective recipes.

Slide 4

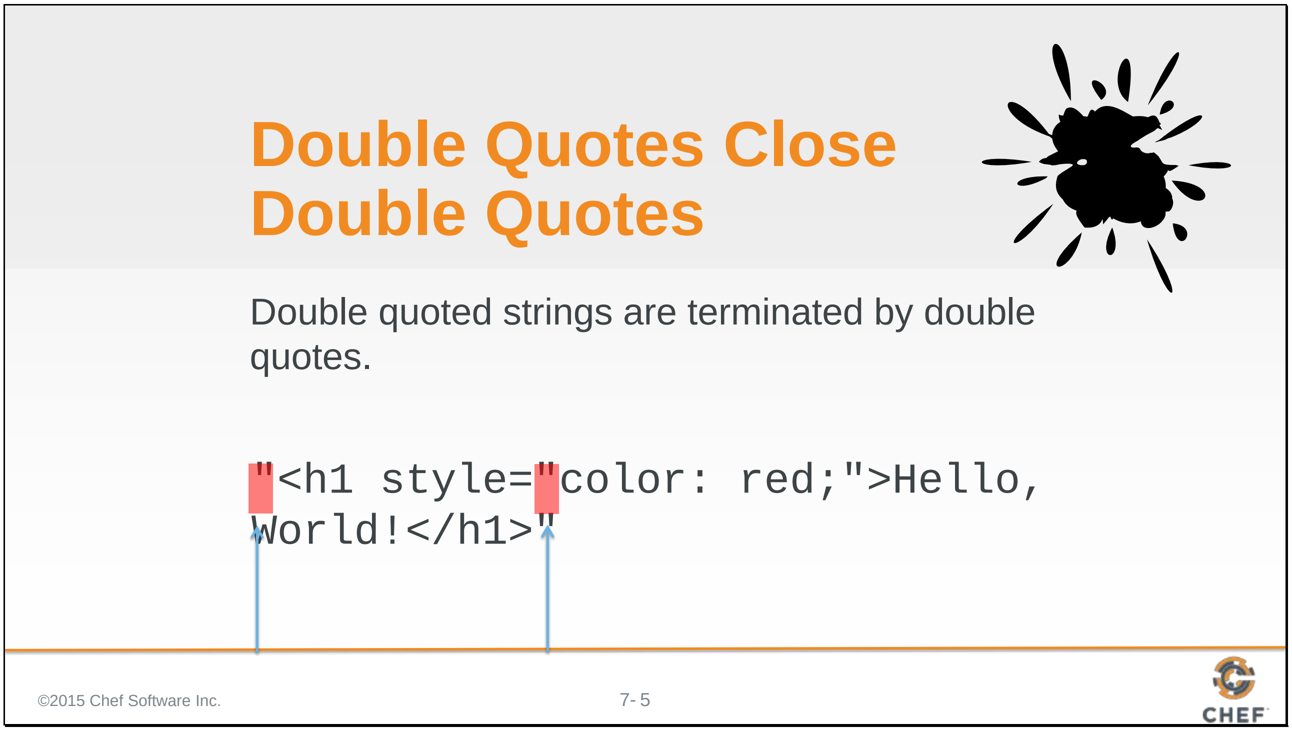
Apache Recipe

~/cookbooks/apache/recipes/server.rb

```
package "apache2"

file "/var/www/html/index.html" do
  content "<h1>Hello, world!</h1>
<h2>ipaddress: #{node["ipaddress"]}</h2>
<h2>hostname: #{node["hostname"]}</h2>
"
end

service "apache2" do
  action [ :enable, :start ]
end
```


What if new changes are given to us for the website splash page? For each new addition we would need to return to this recipe and carefully paste the contents of the new HTML into the string value of the content attribute.

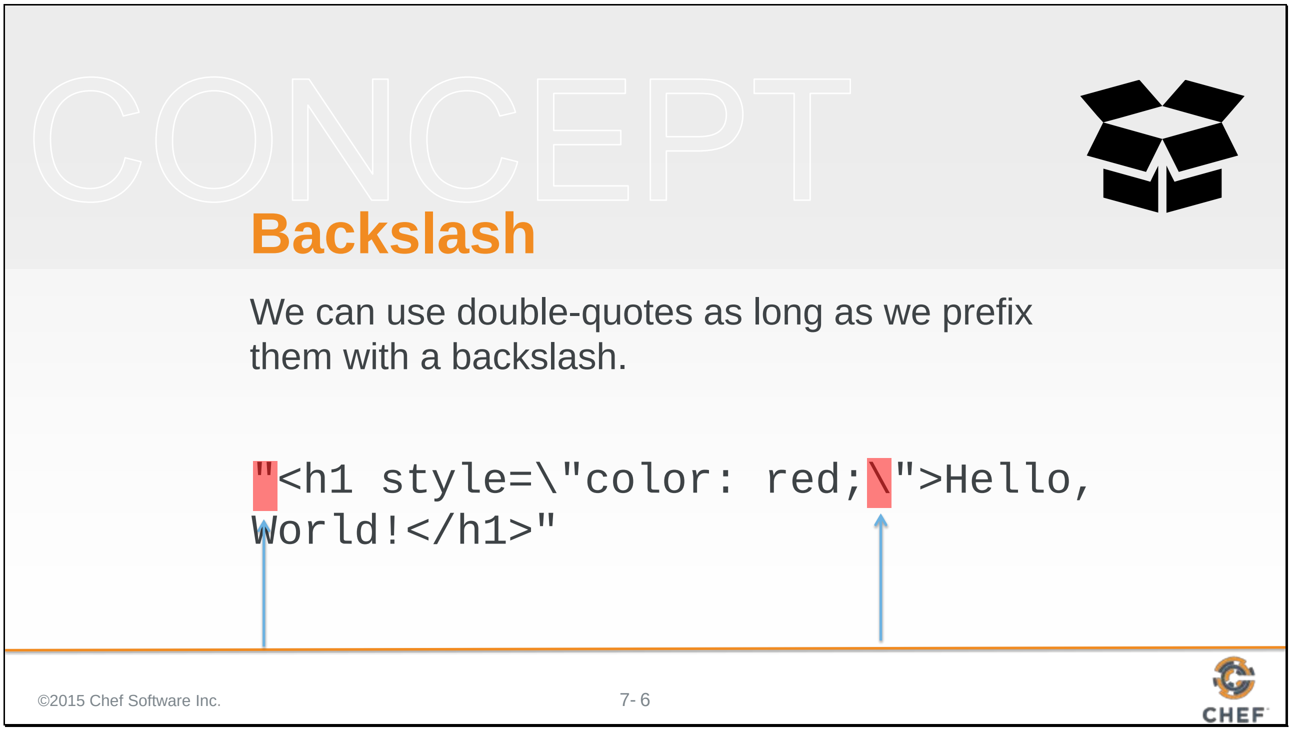
Slide 5



There are some things that you need to be careful of when working with double-quoted strings in Ruby:

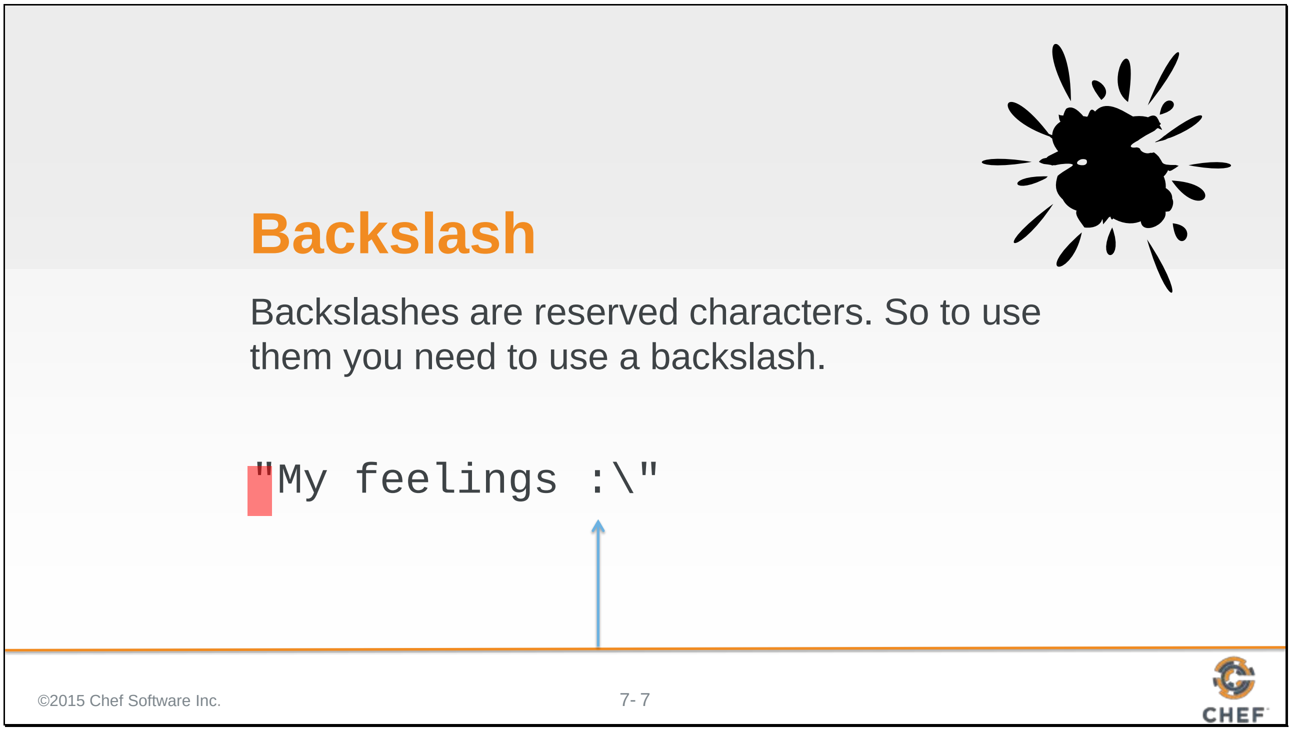
Double-quoted strings are terminated by double-quotes so if any of the text that we paste into this content field has double quotes it is going to have to be escaped.

Slide 6



With Ruby strings you can use the backslash character as an escape character. In this case, if you wanted to have a double-quote inside a double-quoted string, you would need to place a backslash before the double-quote.

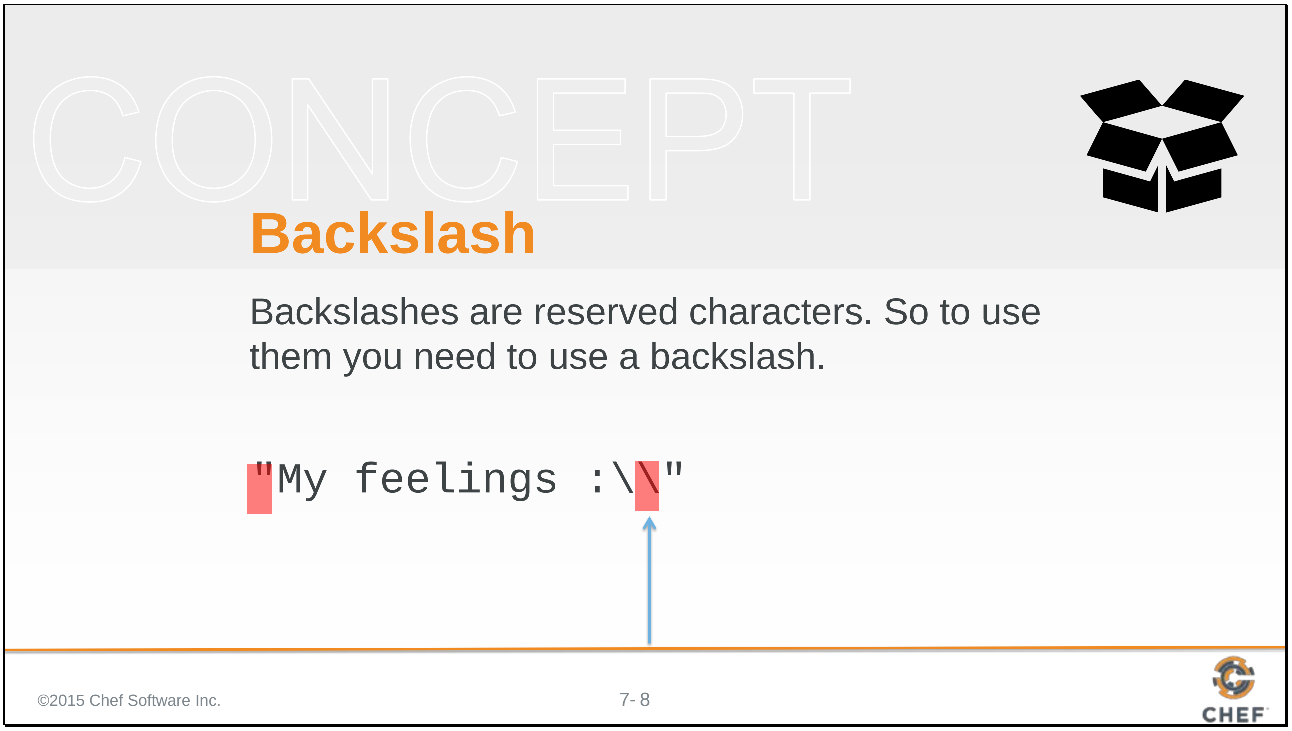
Slide 7



That also brings up an issue with continually-pasting text. You will also need to keep an eye out for backslash characters because backslash characters are now the escape character.

If you want to literally represent a backslash you'll need to use two-backslashes.

Slide 8



So every time text is pasted into the string value of the content attribute, you will need to find and replace all backslashes with double-backslashes and then replace all double-quotes with backslash double-quotes.

Slide 9

Unexpected Formatting

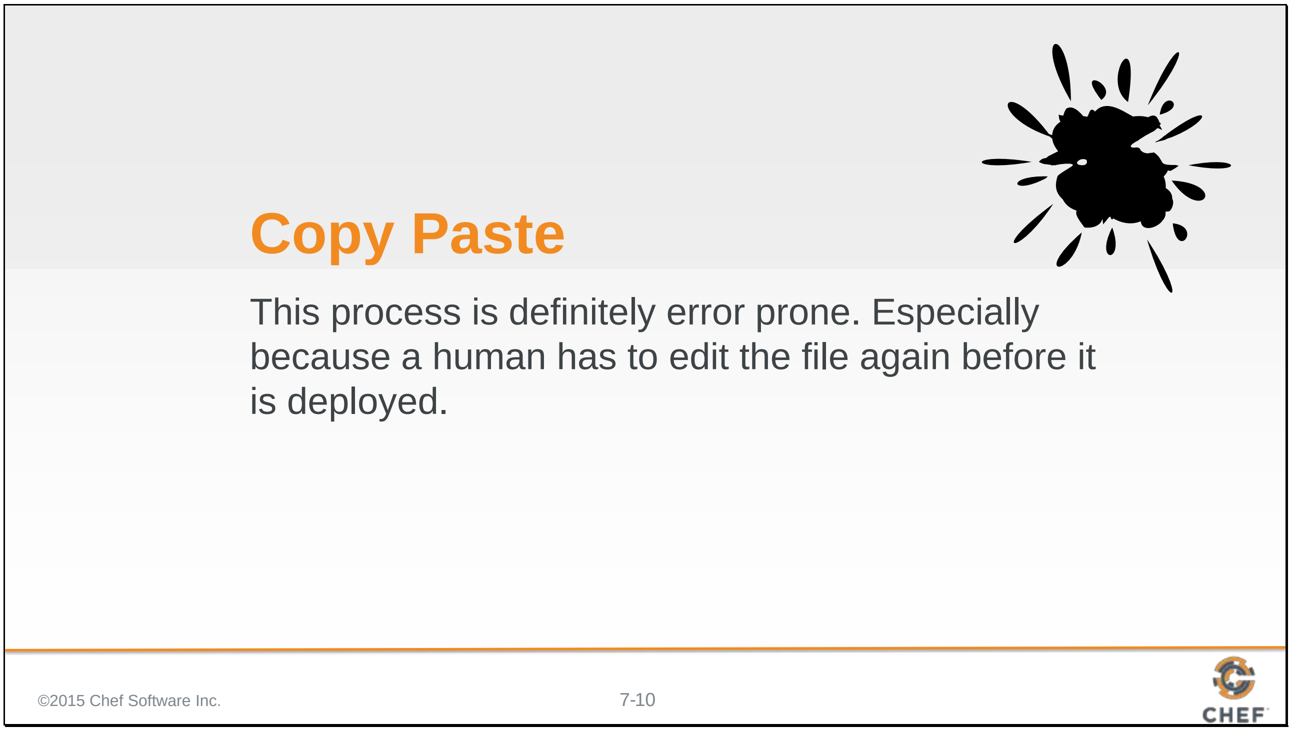
```
file "/etc/motd" do
  content "This is the first line of the file.
           This is the second line. If I try and line it up...

Don't even think about pasting ASCII ART in here!
"
end
```

```
This is the first line of the file.
           This is the second line. If I try and line
it up...

Don't even think about pasting ASCII ART in here!
```


It is important to note that the file content may have some important formatting that might be easily overlooked when working with the content in a recipe file.

Besides that, if the size of the string value of the content field grows, it will consume the recipe--making it difficult to understand what is desired state and what is data.

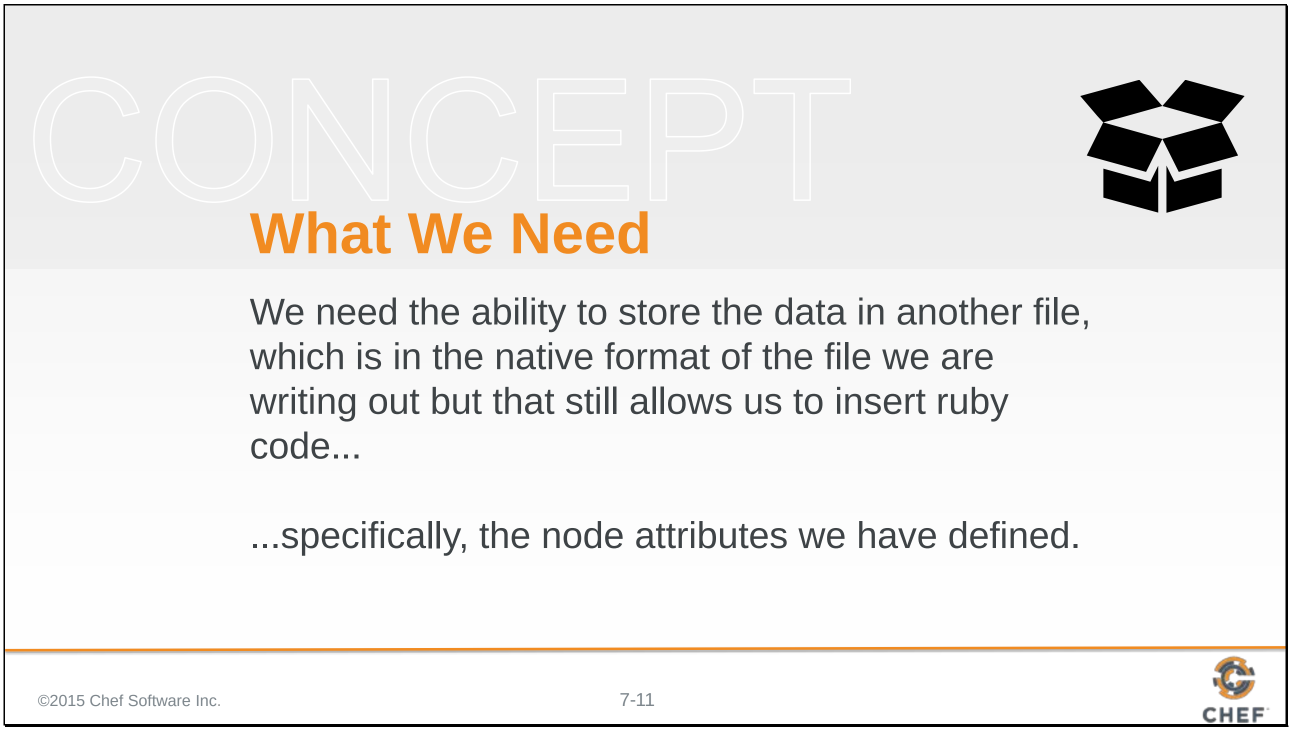
Slide 10



This could sound like a bug waiting to happen.

Any process that requires you to manually copy and paste values and then remember to escape out characters in a particular order, is likely going to lead to issues later when you deploy this recipe to production.

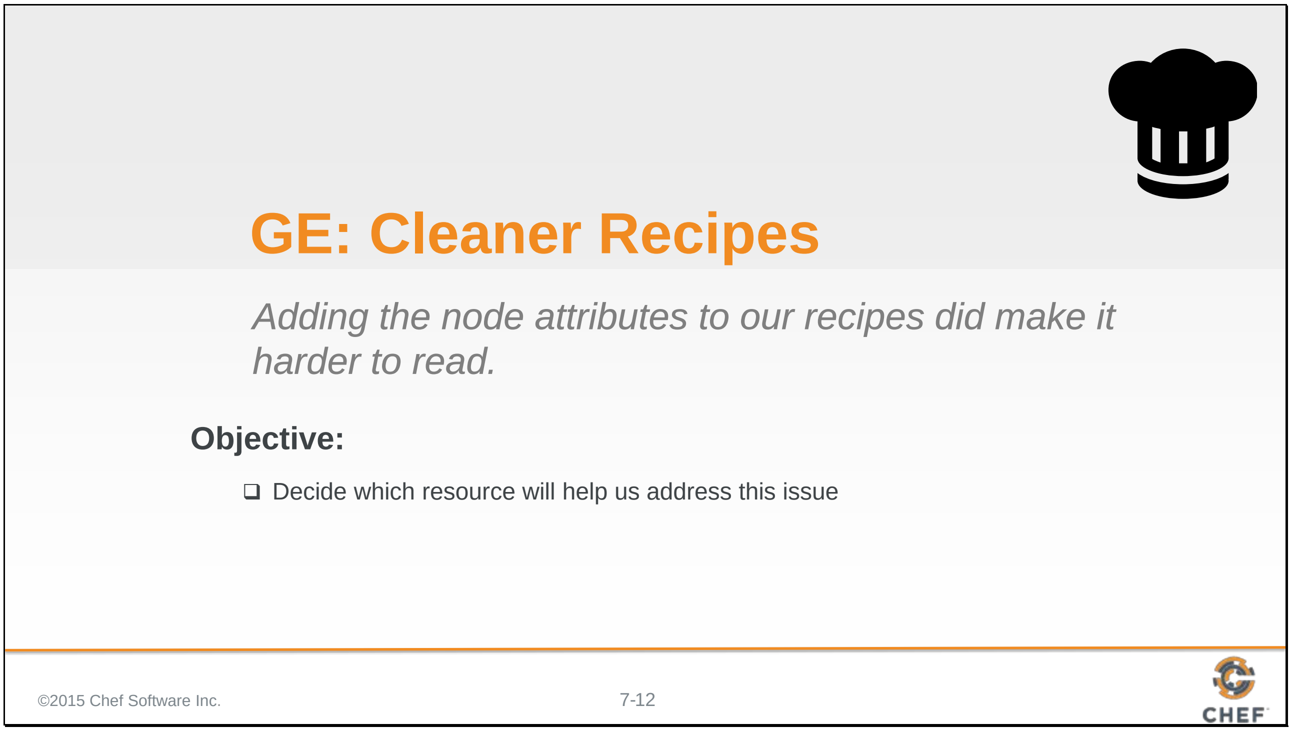
Slide 11



It is better to store this data in another file. The file would be native to whatever format is required so it you wouldn't need to escape any common characters.

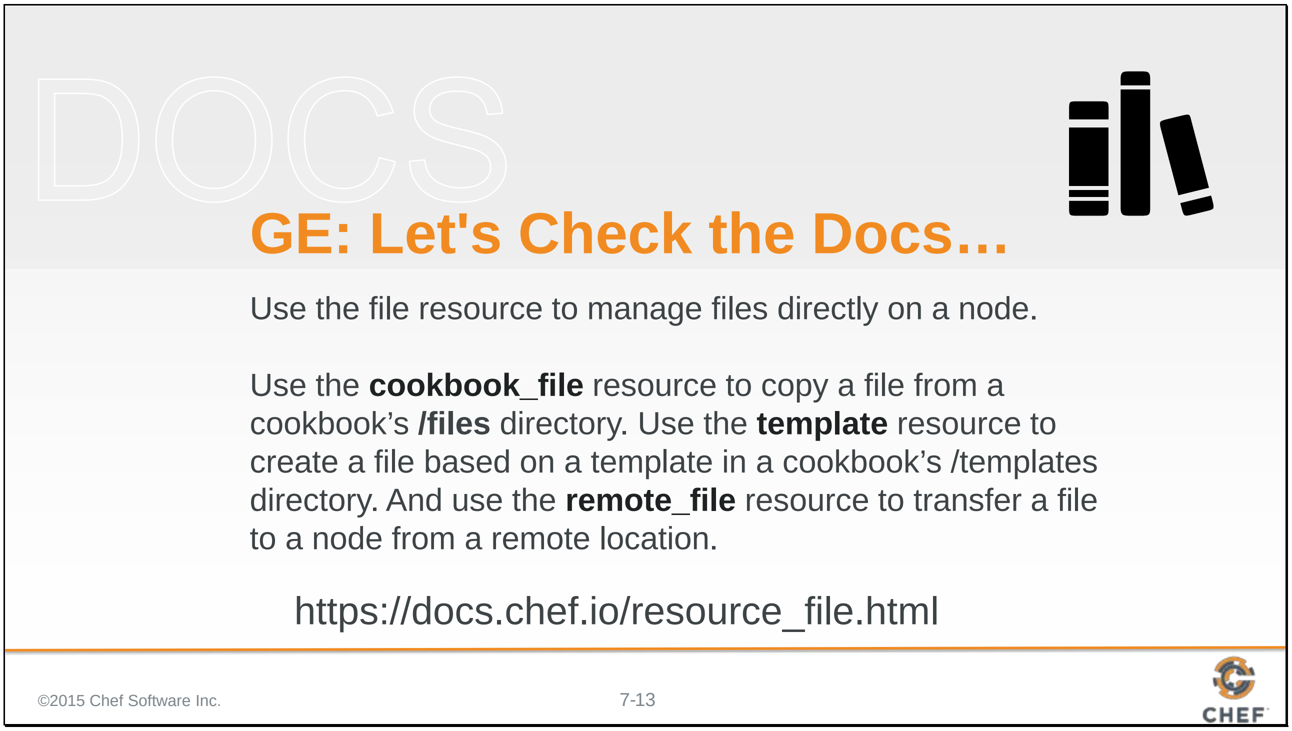
But you still need a way to insert node attributes. So you really need a native file format that allows us to escape out to ruby.

Slide 12



To solve this problem, we need to read up on the file resource more or see if Chef provides alternatives.

Slide 13

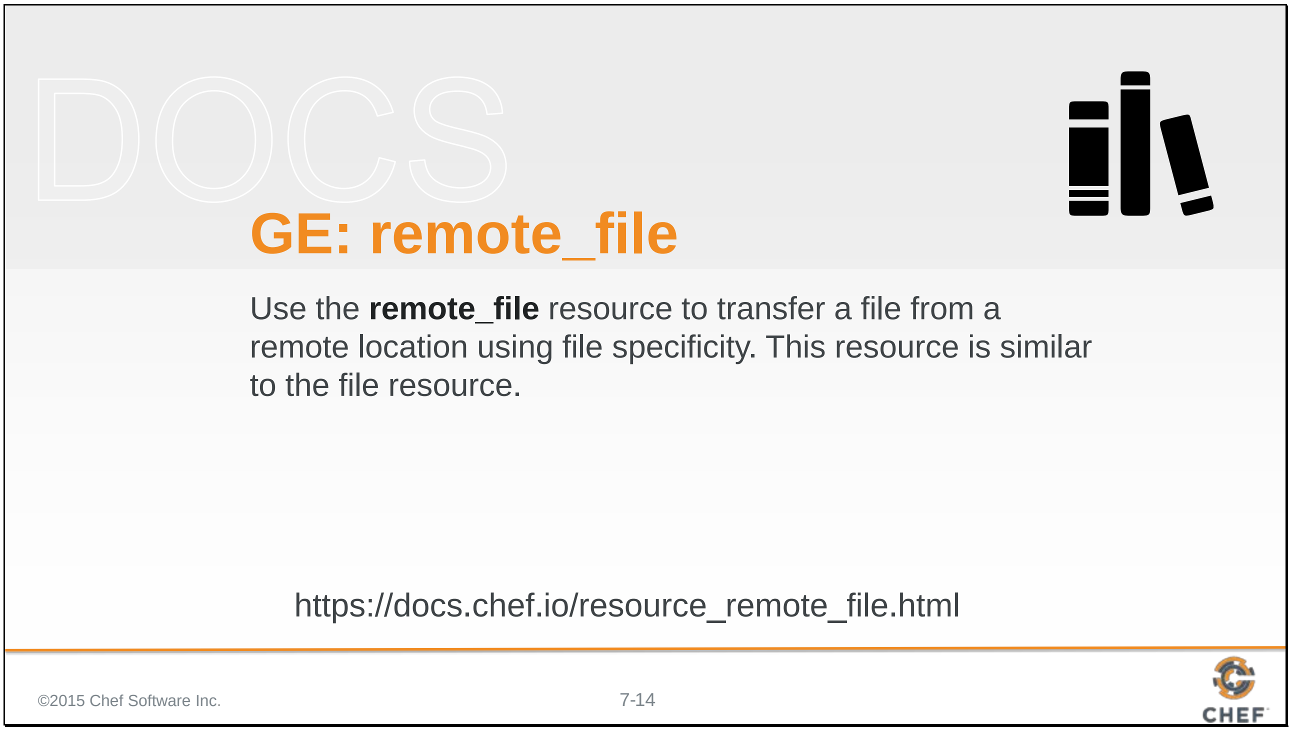


Let's start from what we know--the file resource. Open the documentation and see what it says and see if it gives us an clue to finding alternatives.

The file resource documentation suggests a couple of alternatives to using the file resource: cookbook_file resource; template resource; and remote_file resource.

Lets start with the remote_file resource.

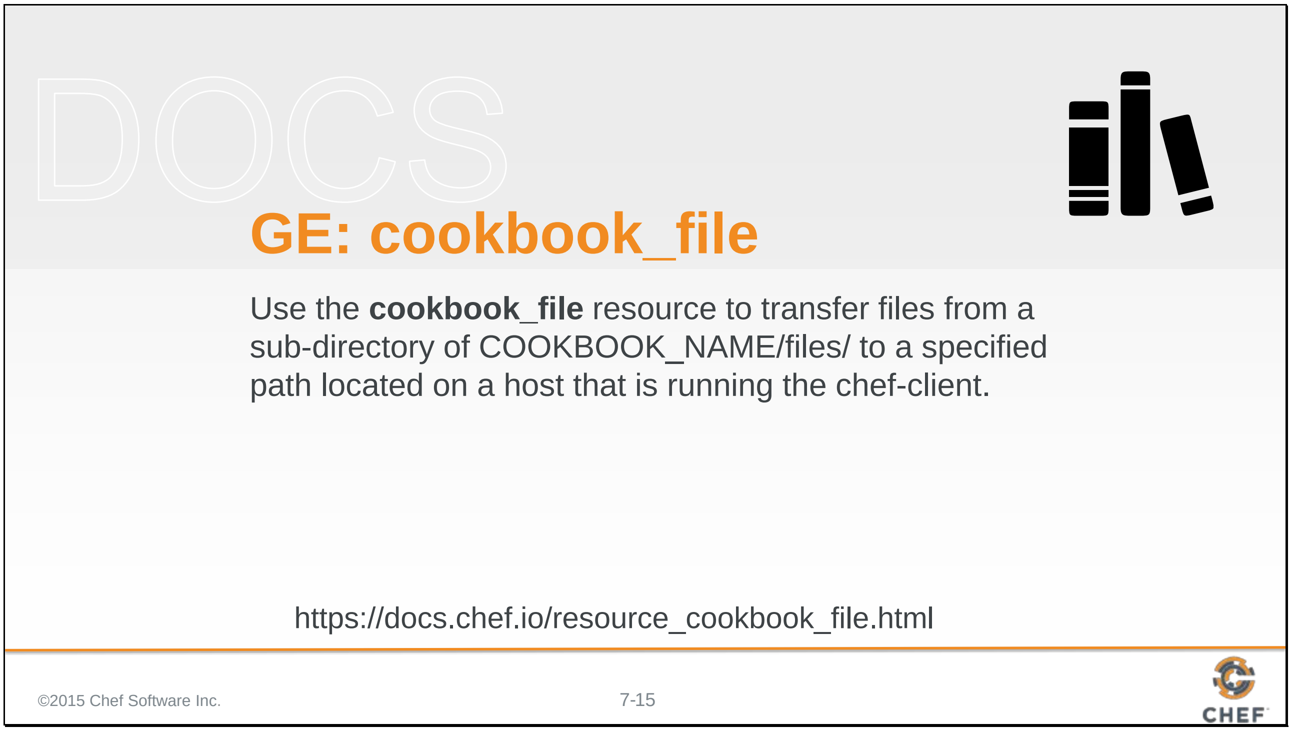
Slide 14



Reading the documentation for remote_file, it seems that remote_file is similar to file. Except remote_file is used to specify a file at a remote location that is copied to a specified file path on the system.

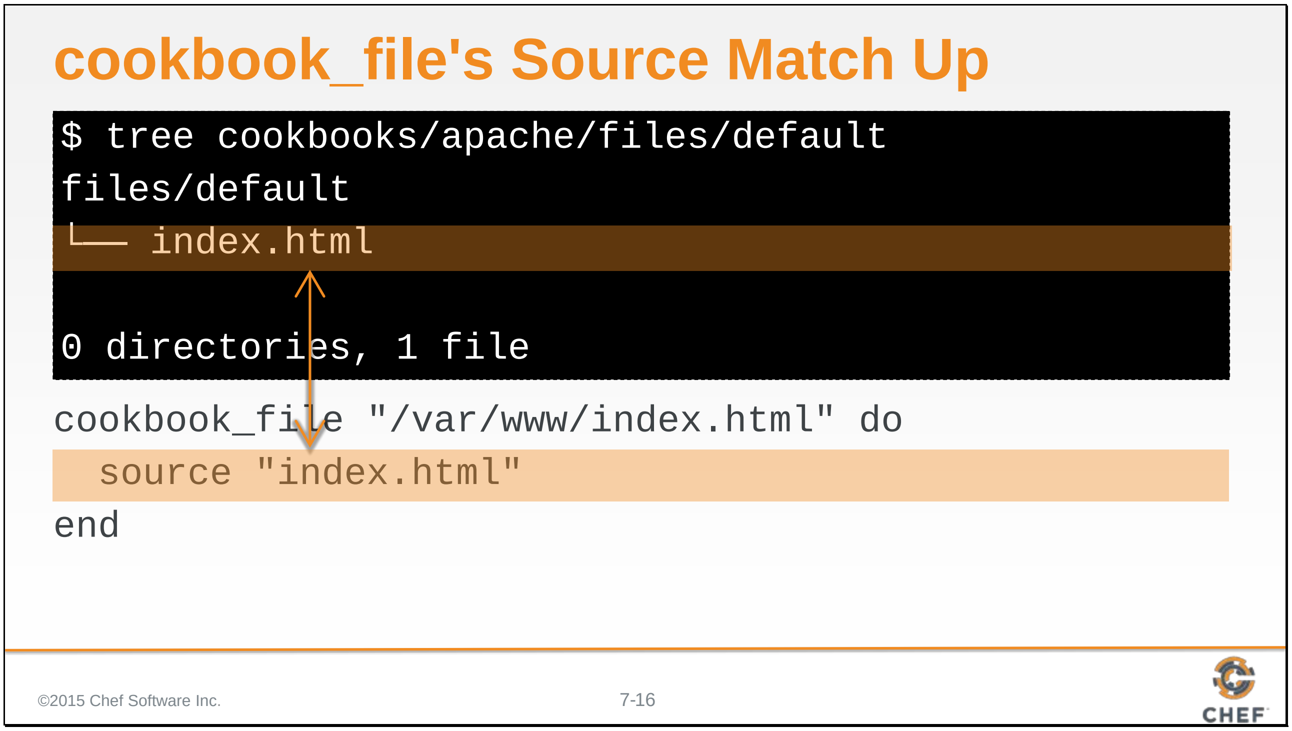
So we could define our index file or message-of-the-day file on a remote system. But that does not allow us to insert attributes about the node we are currently on.

Slide 15



Reading the documentation for cookbook_file, after the boiler-plate resource definition, it sounds as though a cookbook file is capable of...

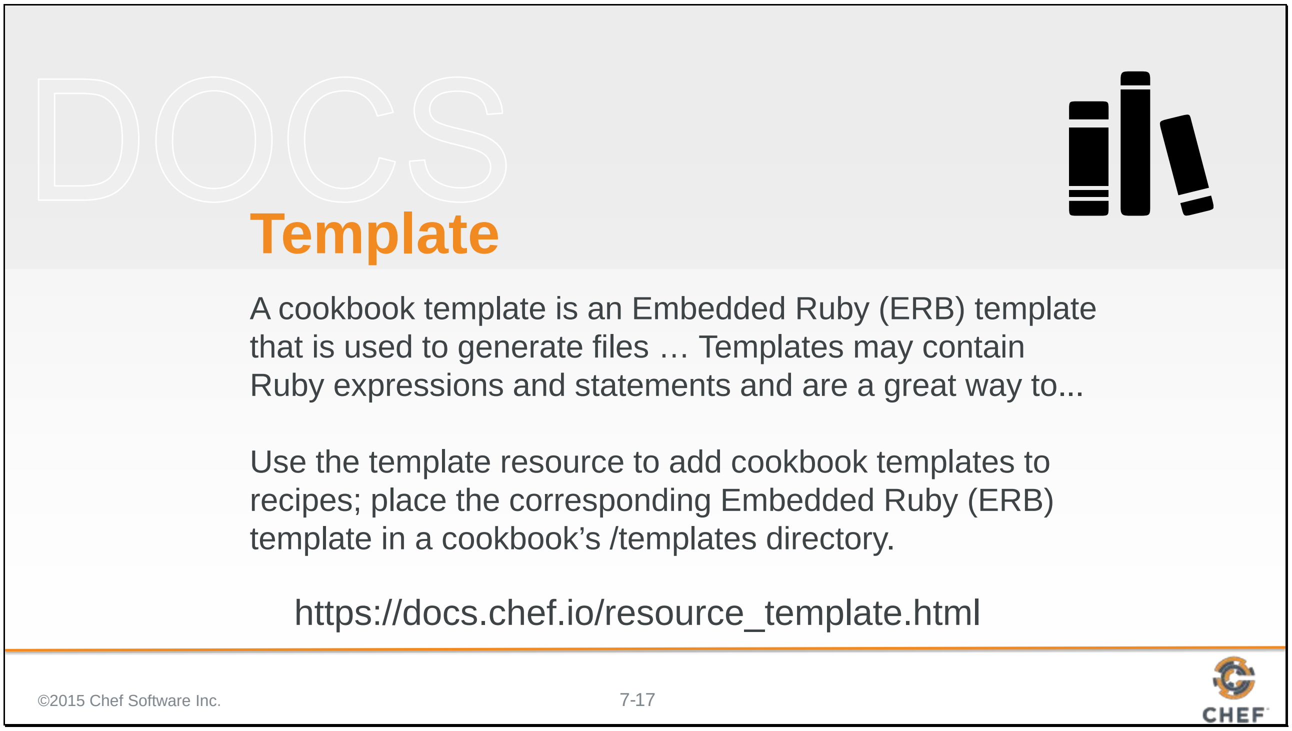
Slide 16



...allowing us to store a file within our cookbook and then have that file transferred to a specified file path on the system.

While it sounds like it allows us to write a file in its native format, it does not sound as though the ability exists to escape out to access the node object and dynamically populate data.

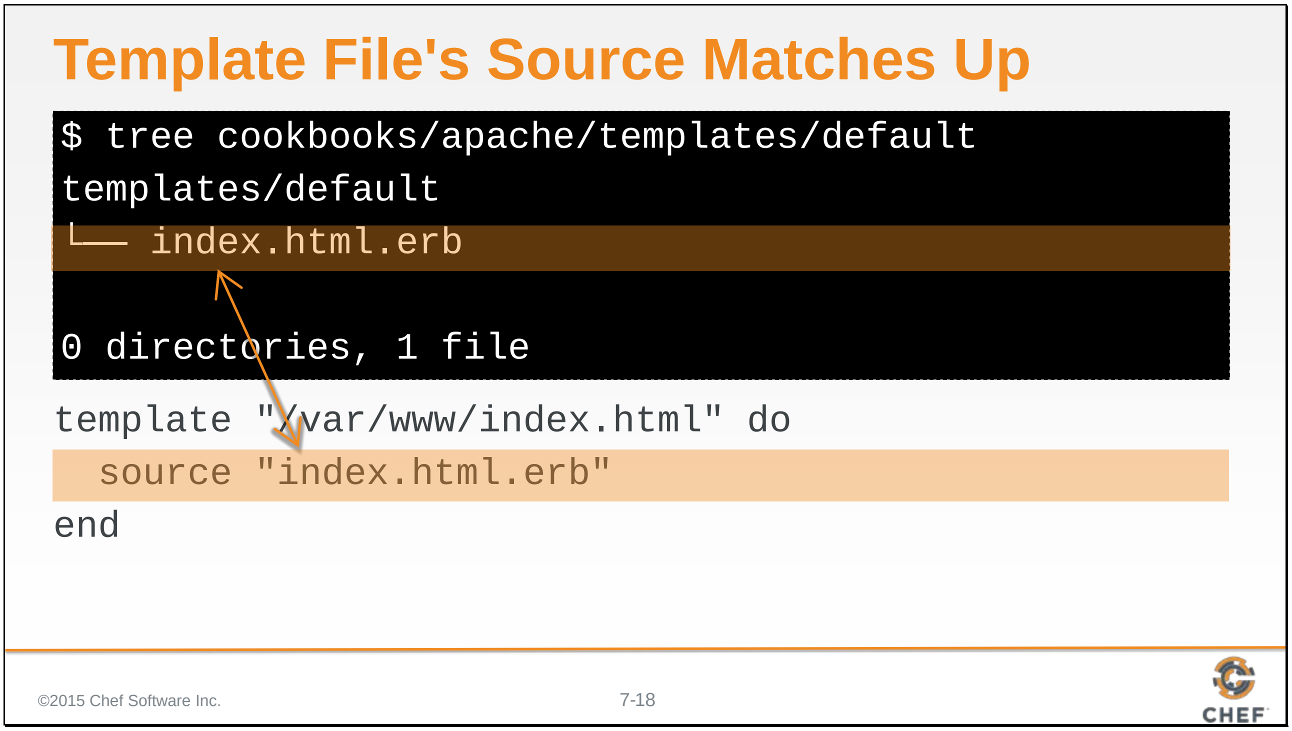
Slide 17



Lets explore templates.

Reviewing the documentation, it seems as though it shares some similarities to the cookbook_file resource.

Slide 18



A template can be placed in a particular directory within the cookbook and it will be delivered to a specified file path on the system.

The biggest difference is that it says templates can contain ruby expressions and statements. This sounds like what we wanted: A native file format with the ability to insert information about our node.

Slide 19

DOCS

Template

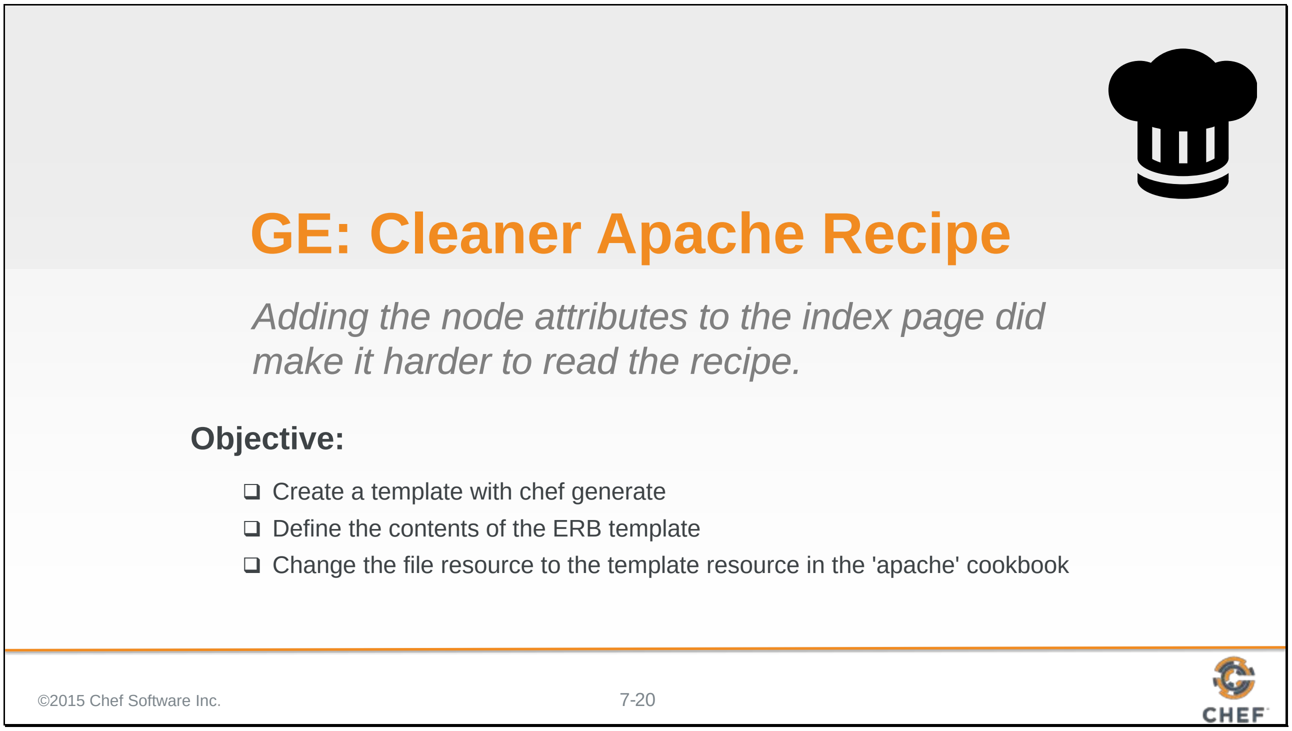
To use a template, two things must happen:

1. A template resource must be added to a recipe
2. An Embedded Ruby (ERB) template must be added to a cookbook

https://docs.chef.io/resource_template.html#using-templates

And if we look at the bottom section about "Using Templates", we'll see more information about what is required and how we can use them to escape out to execute ruby code.

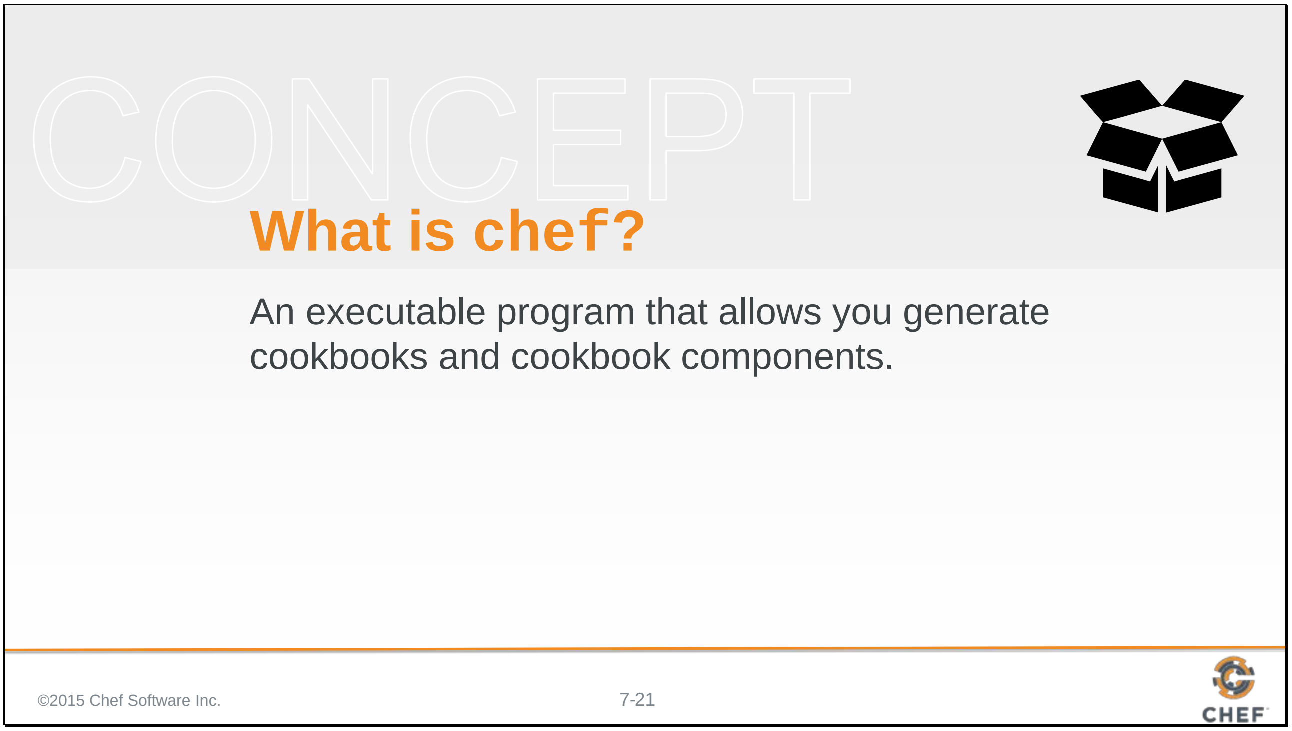
Slide 20



So our objective is clear. We need to use a template resource and create a template and then link them together.

Lets start by creating the actual template file and then we will update the recipe.

Slide 21



Remember that application Chef--the one that generated our cookbooks. Well it is able to generate cookbook components as well.

Templates and files (for cookbook_files) are a few of the other things it can generate for us.

Slide 22

GE: What Can chef Do?

```
$ chef --help
```

```
Usage:
    chef -h/--help
    chef -v/--version
    chef command [arguments...] [options...]


Available Commands:
    exec        Runs the command in context of the embedded ruby
    gem         Runs the `gem` command in context of the embedded ruby
    generate    Generate a new app, cookbook, or component
    shell-init  Initialize your shell to use ChefDK as your primary ruby
    install     Install cookbooks from a Policyfile and generate a locked
cookbook set
```


Lets use help to review the command again.

Slide 23

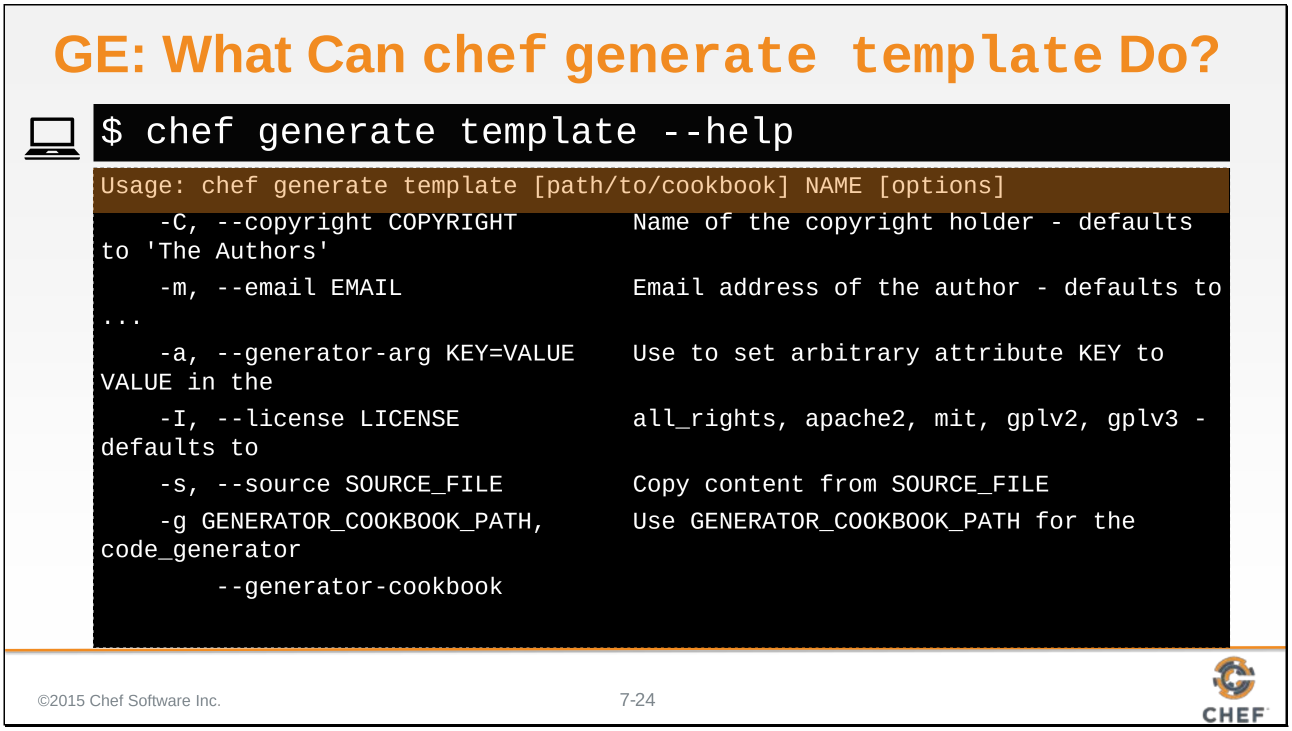
GE: What Can chef generate Do?

```
$ chef generate --help
```

```
Usage: chef generate GENERATOR [options]

Available generators:
  app         Generate an application repo
  cookbook    Generate a single cookbook
  recipe      Generate a new recipe
  attribute   Generate an attributes file
  template    Generate a file template
  file        Generate a cookbook file
  lwrp        Generate a lightweight resource/provider
  repo        Generate a Chef policy repository
  policyfile  Generate a Policyfile for use with the install/push commands
(experimental)
```


And lets ask for help about the 'generate' subcommand.

Slide 24



Finally lets ask for help for generating templates.

The command requires two parameters--the path to where the cookbook is located and the name of the template to generate. There are some other additional options but these two seem like the most important.

Slide 25

GE: Use chef to Generate a Template

```
$ cd ~
$ chef generate template cookbooks/apache index.html
```

```
Compiling Cookbooks...
Recipe: code_generator::template
  * directory[cookbooks/apache/templates/default] action create
    - create new directory cookbooks/apache/templates/default
  * template[cookbooks/apache/templates/default/index.html.erb] action create
    - create new file cookbooks/apache/templates/default/index.html.erb
    - update content in file cookbooks/apache/templates/default/index.html.erb
from none to e3b0c4
    (diff output suppressed by config)
```


Use '**chef generate template'** to create a template in the apache cookbook found in the cookbooks/apache directory and the file we want to create is named index.html.

Slide 26

GE: Lets Look at the Template File

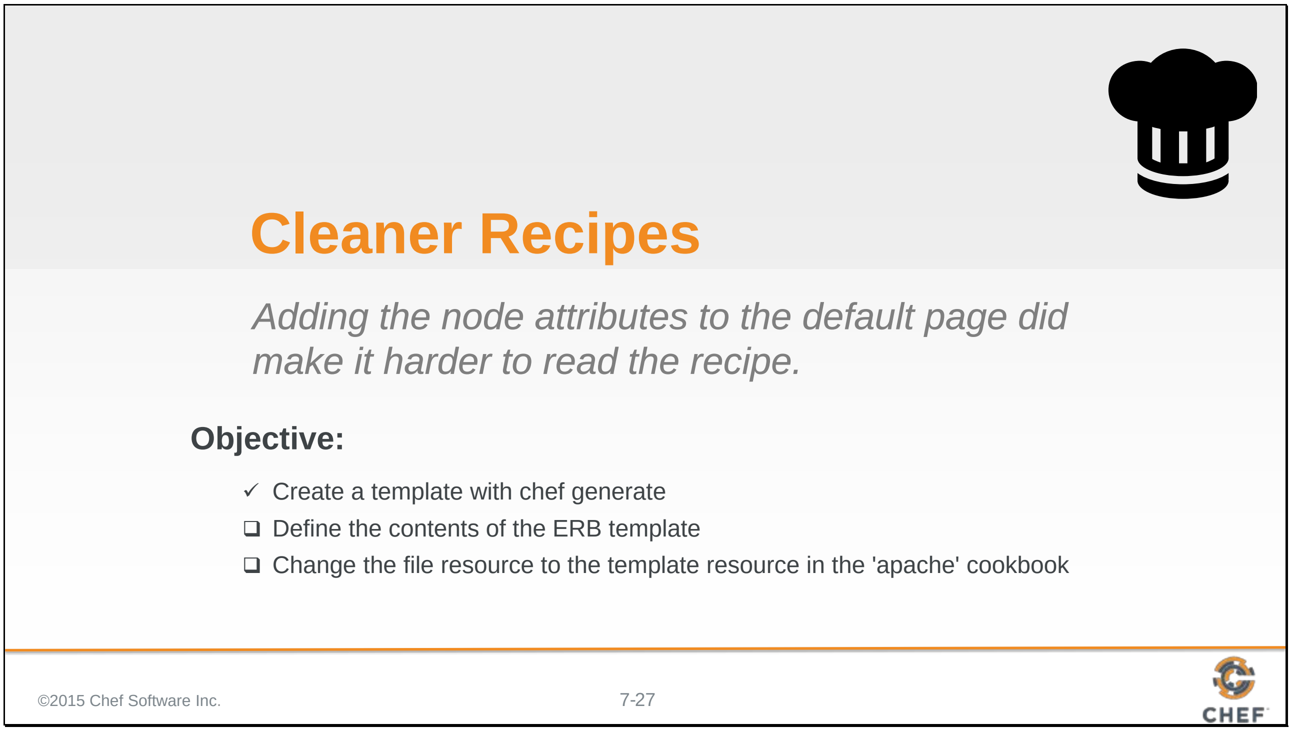
```
$ tree cookbooks/apache/templates
```

```
cookbooks/apache/templates/
└── default
    └── index.html.erb

1 directory, 1 file
```


CHEF

That is the first step. Now that the template exists, we are ready to define the content within the template file.

Slide 27



Now we need to understand what ERB means.

Slide 28

CONCEPT

ERB

An Embedded Ruby (ERB) template allows Ruby code to be embedded inside a text file within specially formatted tags.

Ruby code can be embedded using expressions and statements.

https://docs.chef.io/templates.html#variables

CHEF

ERB template files are special files because they are the native file format we want to deploy but we are allowed to include special tags to execute ruby code to insert values or logically build the contents.

Slide 29

Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

Each ERB tag has a beginning tag and a matched ending tag.

Here is an example of a text file that has several ERB tags defined in it.

Slide 30

Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

Each ERB tag has a beginning tag and a matched ending tag.

CHEF

Each ERB tag has a beginning tag and an ending tag.

Slide 31

Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

Each ERB tag has a beginning tag and a matched ending tag.

The beginning tag is a less-than sign followed by a percent sign. The closing tag is a percent sign followed by a greater-than sign.

Slide 32

Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

Executes the ruby code within the brackets and do not display the result.

These tags are used to execute ruby but the results are not displayed.

Slide 33

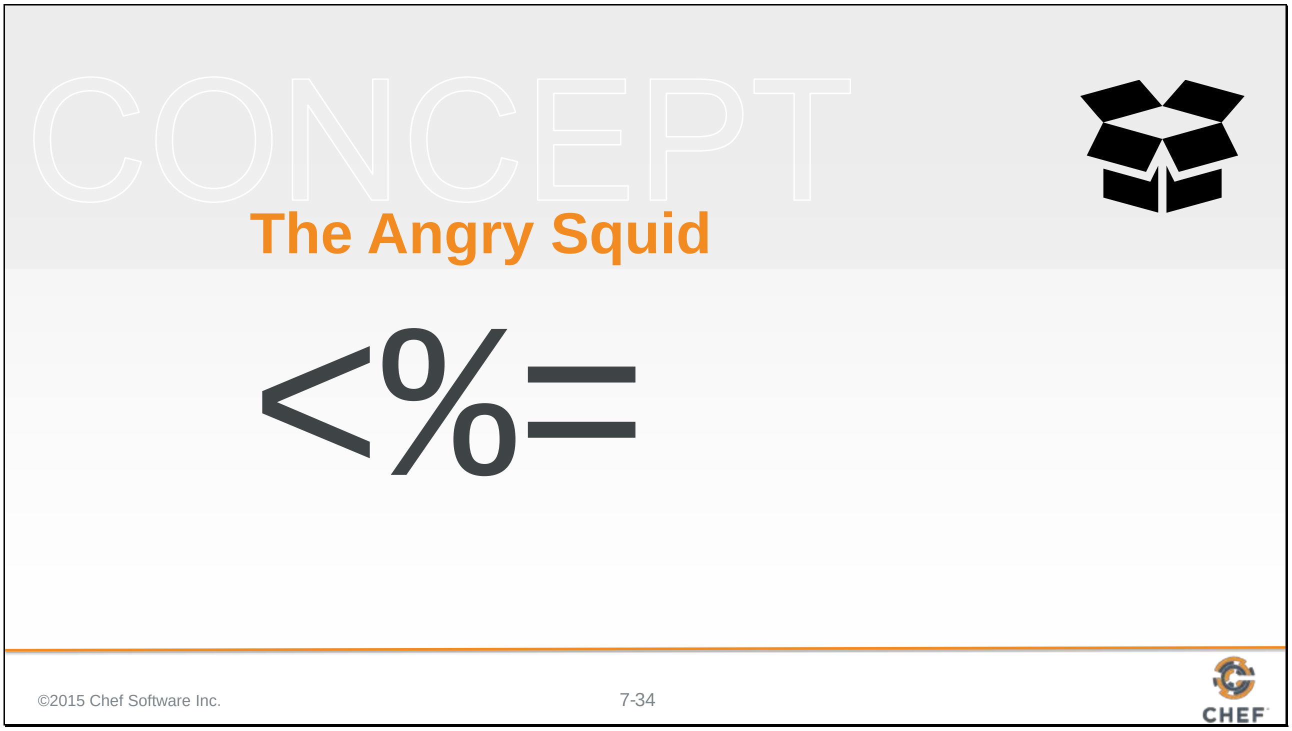
Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

Executes the ruby code within the brackets and display the results.

ERB supports additional tags, one of those is one that allows you to output some variable or some ruby code. Here the example is going to display that 50 plus 50 equals the result of ruby calculating 50 plus 50 and then displaying the result.

Slide 34



The starting tag is different. It has an equals sign. This means show the value stored in a variable or the result of some calculation.

We often refer to this opening tag that outputs the content as the Angry Squid. The less-than is its head, the percent sign as its eyes, and the equals sign its tentacles shooting away after blasting some ink.

Slide 35

GE: Move Our Source to the Template

~/cookbooks/apache/templates/default/index.html.erb

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: #{node["ipaddress"]}</h2>
    <h2>hostname: #{node["hostname"]}</h2>
</body>
</html>
```


With that in mind lets update the template with the current value of the file resource's content field.

Copying this literally into the file does not work because we no longer have the ability to use string interpolation within this html file. String interpolation only works within a ruby file between a double-quoted string.

Slide 36

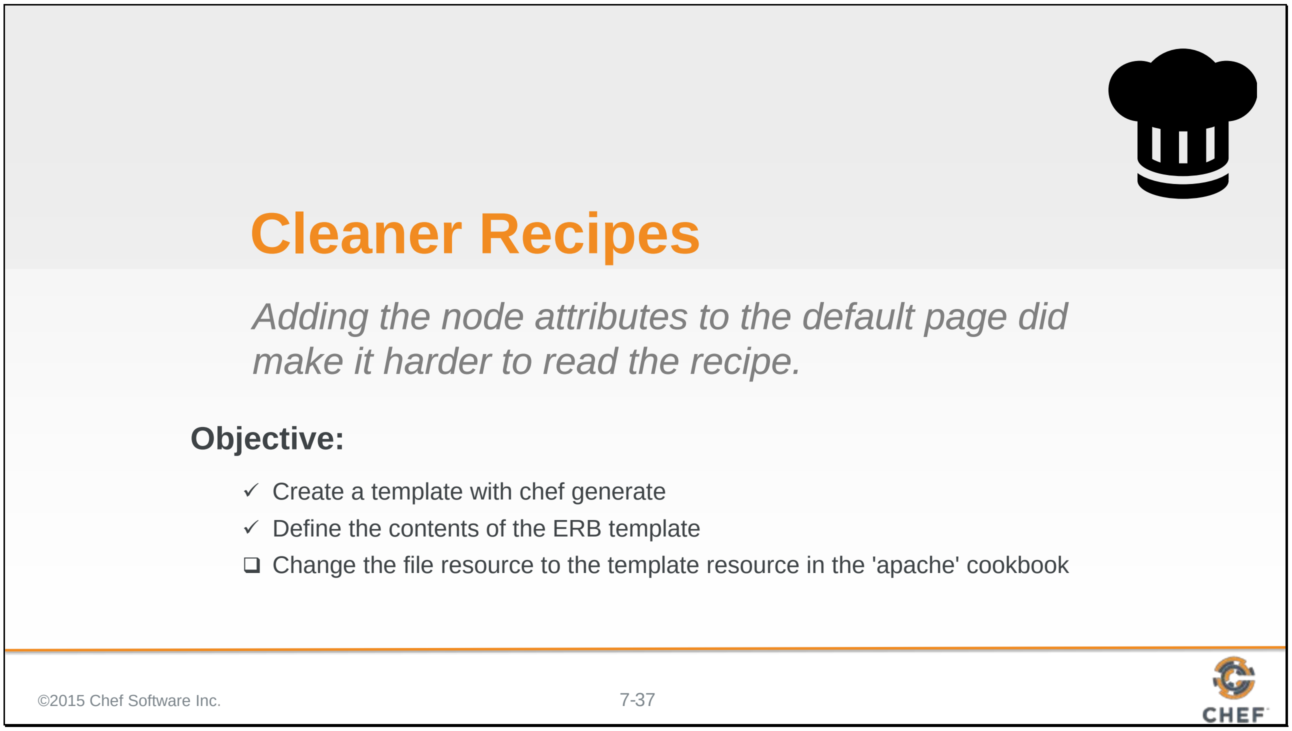
GE: Replace String Interpolation with ERB

~/cookbooks/apache/templates/default/index.html.erb

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: <%= node["ipaddress"] %></h2>
    <h2>hostname: <%= node["hostname"] %></h2>
</body>
</html>
```


We are going to need to change string interpolation sequence with the ERB template syntax. And it seems for this content we want to display the output so we want to make sure that we are using ERB's angry squid opening tag.

Slide 37



The template is created and the contents are correctly defined. It is time to update the recipe.

Slide 38

GE: Remove the Existing Content Attribute

`~/cookbooks/apache/recipes/server.rb`

```
file "/var/www/html/index.html" do
  content "<h1>Hello, world!</h1>
<h2>IPADDRESS: #{node["ipaddress"]}</h2>
<h2>HOSTNAME : #{node["hostname"]}</h2>
"
end
```


Lets open the apache cookbook's recipe named 'server'.

We will want to remove the content attribute from the file resource. Because that content is now in the template. But only if we use a template resource.

Slide 39

GE: Change the File Resource to a Template

~/cookbooks/apache/recipes/server.rb

```
template "/var/www/html/index.html" do

end
```


So its time to change the file resource to a template resource so that it can use the template file that we have defined.

Slide 40

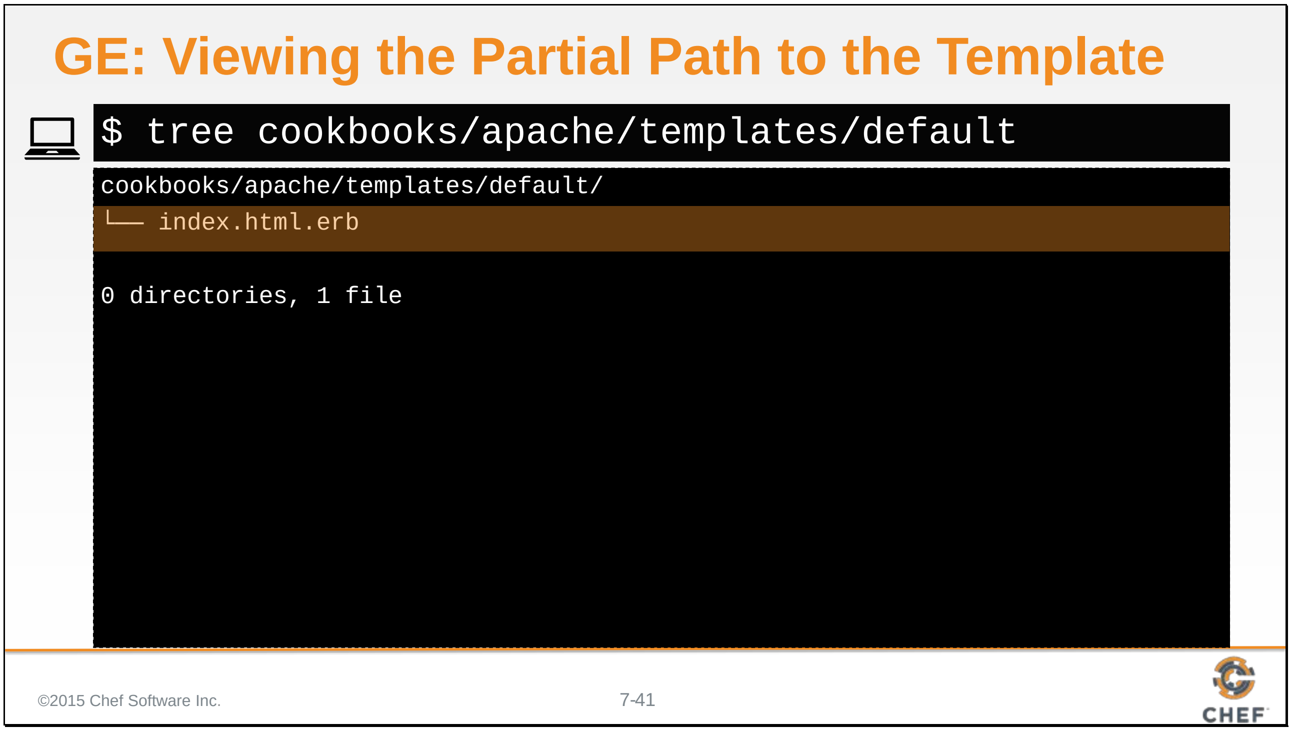
What to Specify as the Source?

~/cookbooks/apache/recipes/server.rb

```
template "/var/www/html/index.html" do
  source "????????????????????"
end
```


Lastly we need to specify a source attribute which contains that path to the template we generated. This path is relative starting from within the cookbook's template directory.

Slide 41



To visualize that with 'tree' we can run it with a path that places us right at the templates directory. So the results will be relative paths from the point specified.

And we see the filepath index.html.erb.

Slide 42

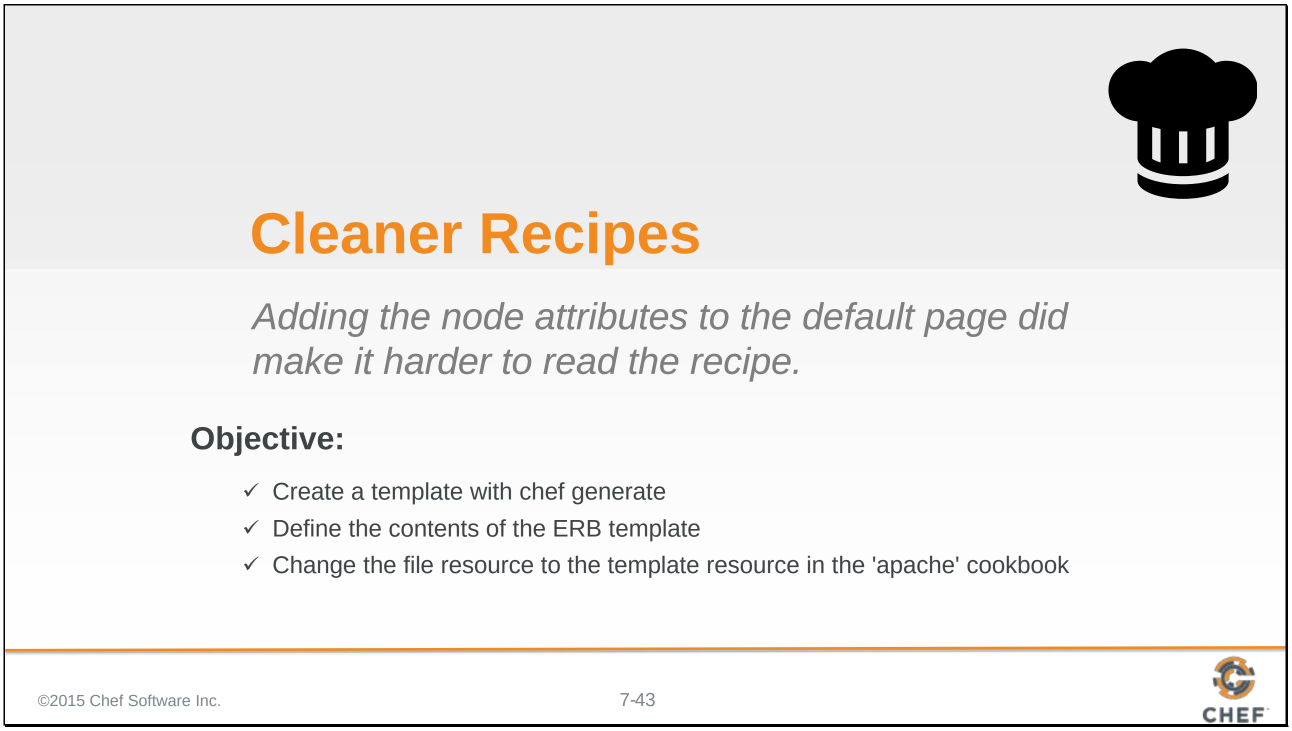
GE: Change the File Resource to a Template

~/cookbooks/apache/recipes/server.rb

```
template "/var/www/html/index.html" do
  source "index.html.erb"
end
```


Now we have the path to our template so we can update the template resource's source attribute value.

Slide 43



We hopefully haven't changed the original goal of our recipe but we have made some changes.

Slide 44

Lab: Update the Version

- ❑ Use kitchen test on the "apache" cookbook
- ❑ Use chef-client to apply the "apache" cookbook's "default" recipe
- ❑ Update the "apache" cookbook's version for this patch
- ❑ Commit the changes

In this lab, you will use 'kitchen' to verify the cookbook and use 'chef-client' to apply the cookbook. If everything is working then update the patch number and commit the changes to version control.

Slide 45

Lab: Test the Cookbook

```
$ cd ~/cookbooks/apache
$ kitchen test
```

```
-----> Starting Kitchen (v1.4.0)
-----> Cleaning up any prior instances of <default-centos-67>
-----> Destroying <default-centos-67>...
       Finished destroying <default-centos-67> (0m0.00s).
-----> Testing <default-centos-67>
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  2.56 kB
       Sending build context to Docker daemon
       Step 0 : FROM centos:centos6
        ---> 72703a0520b7
```


Since kitchen is a cookbook testing tool, you need to move into the cookbook's directory.

Then run the 'kitchen test' command, addressing any issues if they show up.

Slide 46

Lab: Change Directories and Apply the Cookbook

```
$ cd ~
$ sudo chef-client --local-mode -r "recipe[apache]"
```

```
[2015-09-16T14:18:05+00:00] WARN: No config file found or specified on command line, using command line options.
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
[2015-09-16T14:18:09+00:00] WARN: Cloning resource attributes for service[httpd] from prior resource (CHEF-3694)
[2015-09-16T14:18:09+00:00] WARN: Previous service[httpd]: /root/.chef/local-mode-cache/cache/cookbooks/apache/recipes/server.rb:8:in `from_file'
[2015-09-16T14:18:09+00:00] WARN: Current  service[httpd]: /root/.chef/local-mode-cache/ ...
```


When all the tests pass, return to the home directory, so you can execute 'chef-client'.

And then apply the apache cookbook's default recipe to the local system.

Slide 47

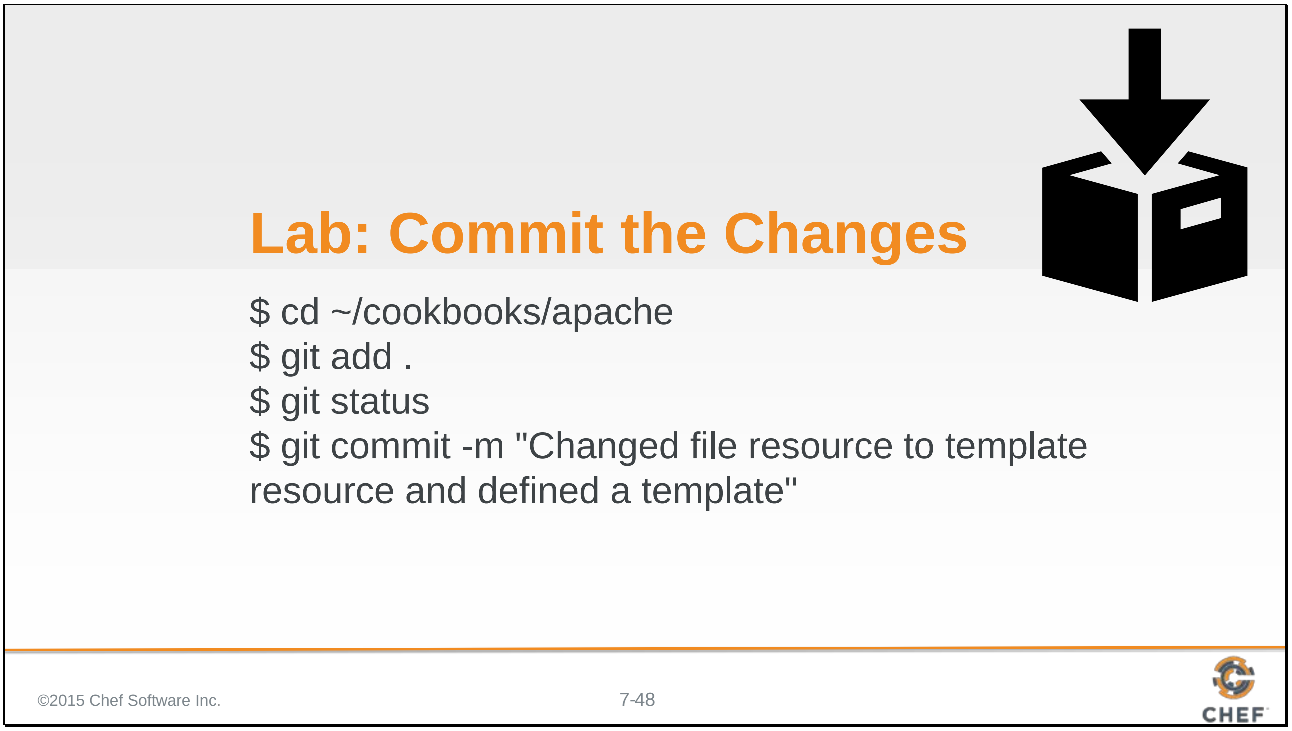
Lab: Update the Cookbook's Patch Number

~/cookbooks/apache/metadata.rb

```
name             'apache'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures apache'
long_description 'Installs/Configures apache'
version          '0.2.1'
```

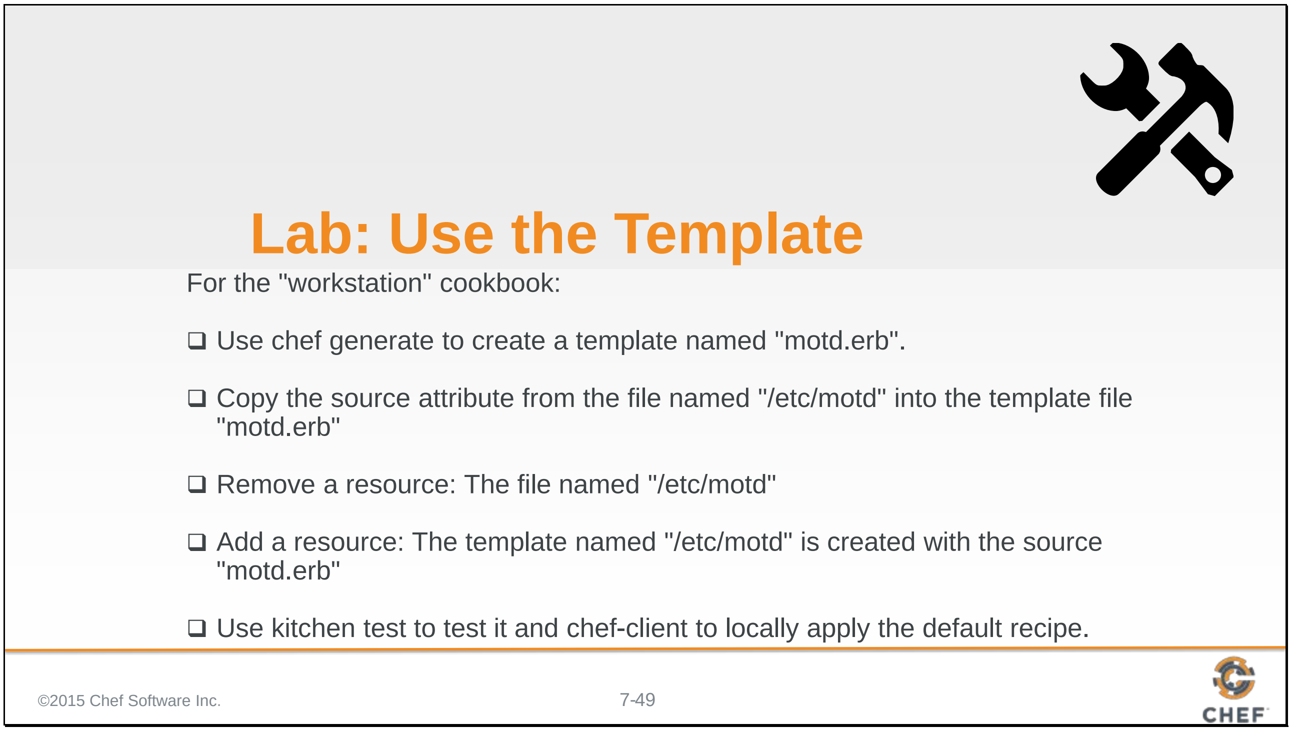

If everything converges correctly, update the version number. As mentioned previously, this is a patch fix.

Slide 48



Return to the cookbook directory and add all the changed files and commit them with a message.

Slide 49



Its time to do that again--this time for the workstation cookbook.

Generate a template named 'motd', copy in the source attribute from the file resource, and then update it to use ERB tags.

Then come back to the recipe. Change it to a template resource and then add a source attribute whose value is that partial path to the new template you created.

Slide 50

Lab: Return Home and Generate the Template

```
$ cd ~
$ chef generate template cookbooks/workstation motd
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/home/chef/template] action create
    - create new directory /home/chef/template
  * template[/home/chef/template/metadata.rb] action create_if_missing
    - create new file /home/chef/template/metadata.rb
    - update content in file /home/chef/template/metadata.rb from none to 000bce
    (diff output suppressed by config)
  * template[/home/chef/template/README.md] action create_if_missing
    - create new file /home/chef/template/README.md
```


Return to the home directory. Run the command to generate the template named 'motd' in the workstation cookbook.

Slide 51

Lab: Copy the Existing Source into the Template

~/cookbooks/workstation/templates/default/motd.erb

```
Property of ...

  IPADDRESS: #{node["ipaddress"]}
  HOSTNAME : #{node["hostname"]}
  MEMORY   : #{node["memory"]["total"]}
  CPU      : #{node["cpu"]["0"]["mhz"]}
```


We can start by copying and pasting the existing content for the Message of the Day file into the template file.

Slide 52

Lab: Update the motd.erb to Use ERB

~/cookbooks/workstation/templates/default/motd.erb

```
Property of ...

  IPADDRESS: <%= node["ipaddress"] %>
  HOSTNAME : <%= node["hostname"] %>
  MEMORY   : <%= node["memory"]["total"] %>
  CPU      : <%= node["cpu"]["0"]["mhz"] %>
```


Replace all the string interpolation with ERB tags.

Slide 53

Lab: Remove the file Resource

~/cookbooks/workstation/recipes/setup.rb

```
file "/etc/motd" do
  content "Property of ...

  IPADDRESS: #{node["ipaddress"]}
  HOSTNAME : #{node["hostname"]}
  MEMORY   : #{node["memory"]["total"]}
  CPU      : #{node["cpu"]["0"]["mhz"]}
"
  mode "0644"
  owner "root"
  group "root"
end
```


Remove the file resource from the setup recipe.

Slide 54

Lab: Replace it with the Template Resource

~/cookbooks/workstation/recipes/setup.rb

```
template "/etc/motd" do
  source "motd.erb"
  mode "0644"
  owner "root"
  group "root"
end
```


...and replace it with the Template resource. The source attribute specifies the file path 'motd.erb' - the new template file that was created.

Slide 55

Lab: Test the Cookbook

```
$ cd ~/cookbooks/workstation
$ kitchen test
```

```
-----> Starting Kitchen (v1.4.0)
-----> Cleaning up any prior instances of <default-centos-67>
-----> Destroying <default-centos-67>...
       Finished destroying <default-centos-67> (0m0.00s).
-----> Testing <default-centos-67>
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  2.56 kB
       Sending build context to Docker daemon
       Step 0 : FROM centos:centos6
        ---> 72703a0520b7
```


Since kitchen is a cookbook testing tool, you need to move into the cookbook's directory.

Then run the 'kitchen test' command, addressing any issues if they show up.

Slide 56

Lab: Change Directories and Apply the Cookbook

```
$ cd ~
$ sudo chef-client --local-mode -r "recipe[workstation]"
```

```
[2015-09-16T14:18:05+00:00] WARN: No config file found or specified on command
line, using command line options.
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
[2015-09-16T14:18:09+00:00] WARN: Cloning resource attributes for
service[httpd] from prior resource (CHEF-3694)
[2015-09-16T14:18:09+00:00] WARN: Previous service[httpd]: /root/.chef/local-
mode-cache/cache/cookbooks/apache/recipes/server.rb:8:in `from_file'
[2015-09-16T14:18:09+00:00] WARN: Current  service[httpd]: /root/.chef/local-
mode-cache/ ...
```


When all the tests pass, return to the home directory, so you can execute 'chef-client'.

And then apply the workstation cookbook's default recipe to the local system.

Slide 57

Lab: Update the Version

- ❑ Update the "workstation" cookbook's version for this patch
- ❑ Commit the changes to the "workstation" cookbook to version control

Slide 58

Lab: Update the Cookbook's Patch Number

~/cookbooks/workstation/metadata.rb

```
name              'workstation'
maintainer        'The Authors'
maintainer_email  'you@example.com'
license           'all_rights'
description       'Installs/Configures workstation'
long_description  'Installs/Configures workstation'
version           '0.2.1'
```


Update the patch version number for the workstation cookbook.

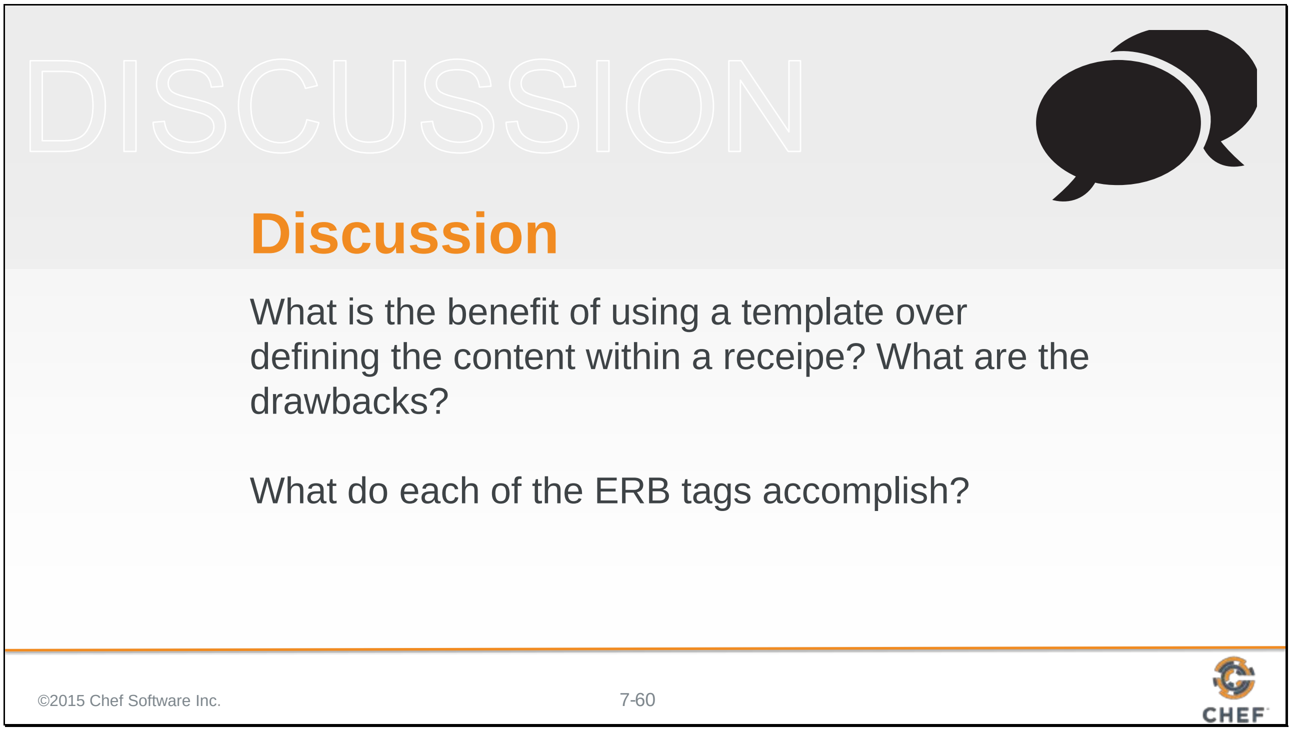
Slide 59

Lab: Commit the Changes

```
$  cd ~/cookbooks/workstation
$ git add .
$ git status
$ git commit -m "Changed file resource to template resource and defined a template"
```


Add and then commit the changes to the workstation cookbook.

Slide 60



Answer these questions.

With your answers, turn to another person and alternate asking each other these questions and sharing your answers.

Slide 61

DISCUSSION

Q&A

What questions can we help you answer?

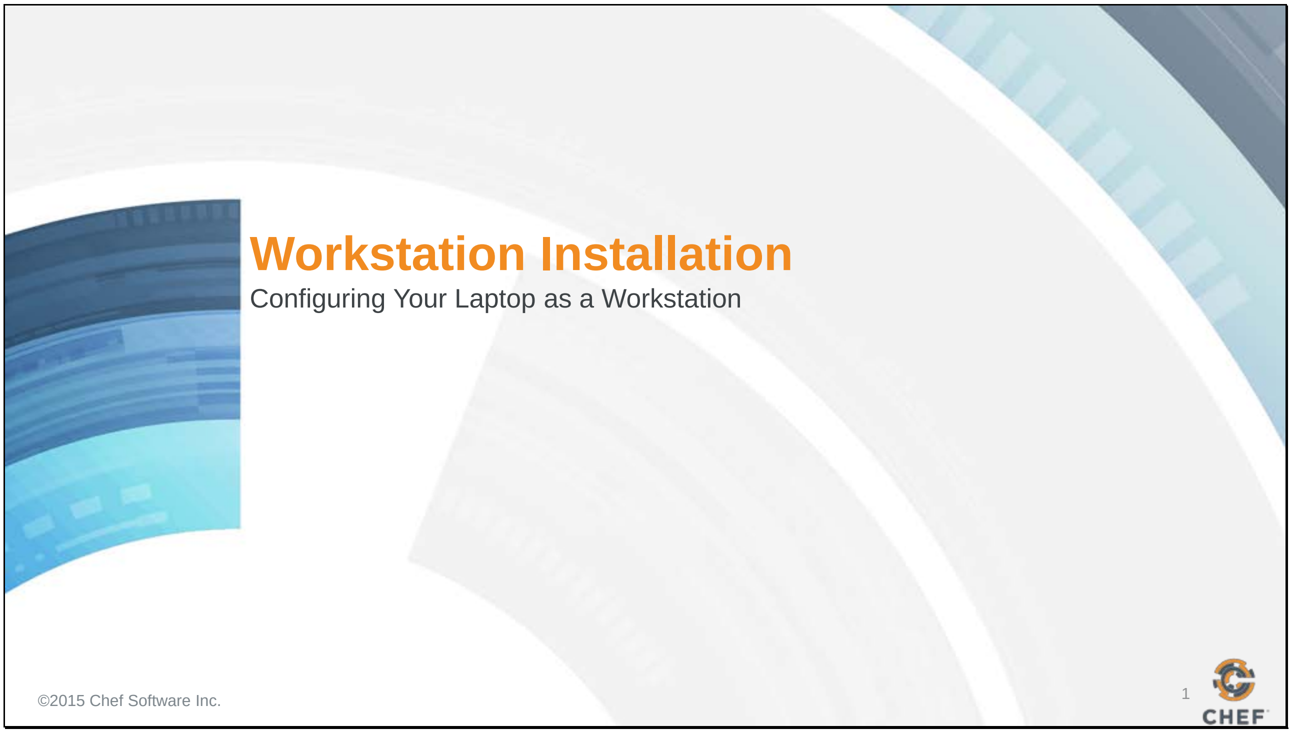
- Resources (file, cookbook_file, template, and remote_file)
- Templates
- ERB

What questions can we help you answer?

Generally or specifically about resources, templates, and ERB.

Slide 62

8: Workstation Installation



Slide 2

Objectives

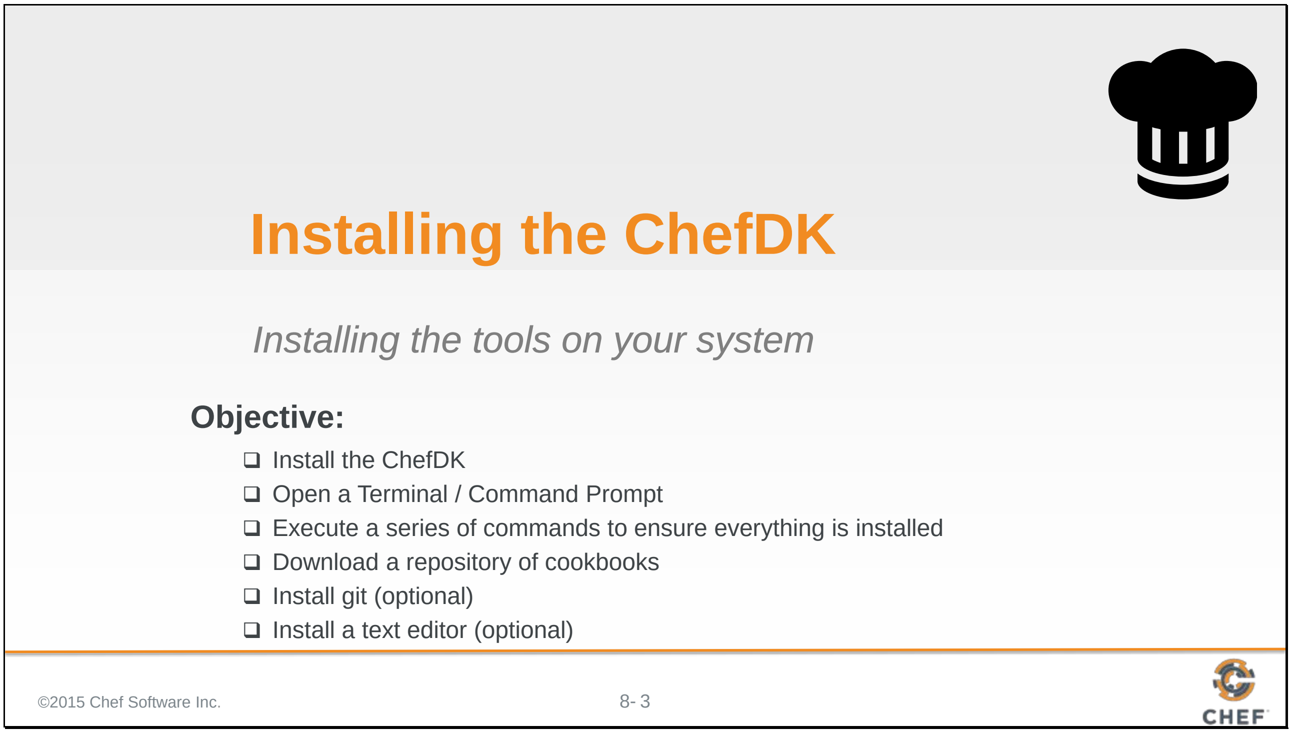
After completing this module, you should be able to

- Ensure that ChefDK is installed on your laptop
- Execute a series of commands to ensure everything is installed
- Download a repository of cookbooks
- Install a local editor like Atom

We have been doing a lot of great work with Chef on this remote workstation that we have provided for you.

In this section we will walk through the installation of the necessary tools and the commands to verify your installation.

Slide 3



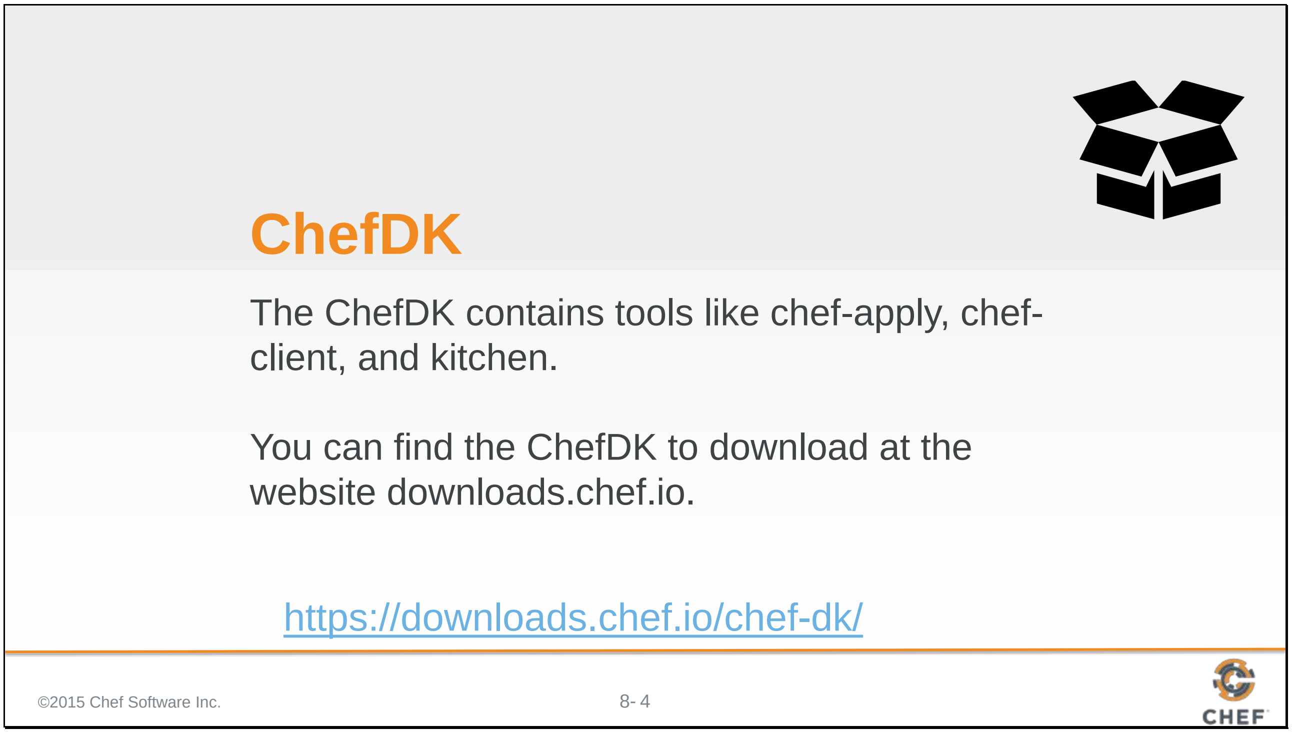
To become a successful Chef developer, you will want to install Chef tools on your local workstation. These tools are available in the Chef Development Kit (ChefDK). After the installation is complete, we will verify that the various tools are working.

Then we will download a copy of the cookbooks that we created together.

After that you can optionally download a number of other tools that will help you in your journey using Chef. The first is git and the second is a text editor.

Let's get started.

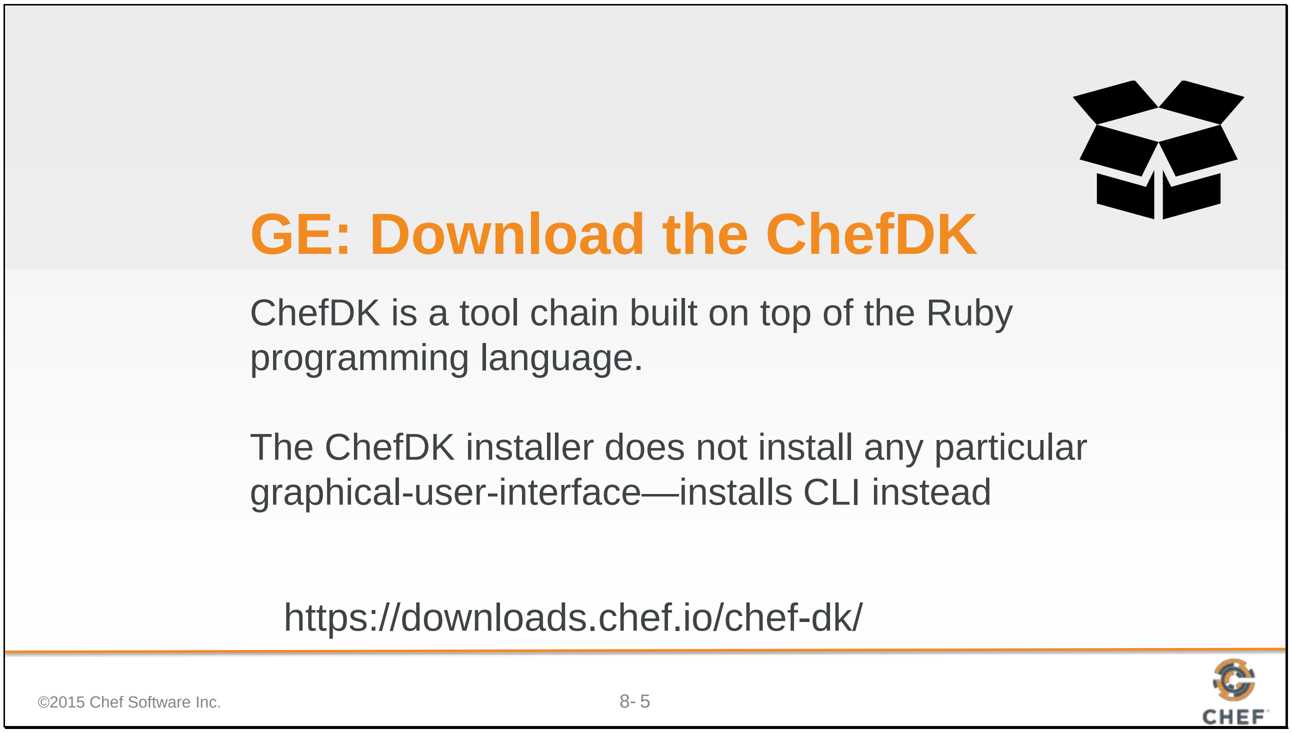
Slide 4



Throughout this course we have been using a number of tools found within the ChefDK. The ChefDK contains tools like 'chef-apply', 'chef-client', and 'kitchen'. It also has a number of other great tools that we will use when connecting to a Chef Server, managing cookbook dependencies, or ensuring the quality of the cookbooks that we write.

You can download the ChefDK at https://downloads.chef.io/chef-dk.

Slide 5



The ChefDK is a tool chain built on top of the Ruby programming language. To assist with making the tools more portable to all platforms we package Ruby and all these tools together in a single platform specific installation package.

The installer does not install any particular graphical user interface, GUI, but instead installs the command-line tools we have been using thus far.

You may have already downloaded the ChefDK previously in this course.

Slide 6

GE: Installing ChefDK

The omnibus installer is used to set up the Chef development kit on a workstation, including the chef-client itself, an embedded version of Ruby, RubyGems, OpenSSL, key-value stores, parsers, libraries, command line utilities, and community tools such as Kitchen, Berkshelf, and ChefSpec.

https://downloads.chef.io/chef-dk/

CHEF

Slide 7

Follow the ChefDK installation wizard's instructions. It could take over 10 minutes to install ChefDK.

ChefDk will be installed into an opscode folder on your laptop.

Slide 8

Lab: Run All These Commands

```
$ chef --version
$ chef-client --version
$ knife --version
$ ohai --version
$ berks --version
$ kitchen --version
$ foodcritic --version
$ rubocop --version
```


Open a local command prompt or something like Windows Power Shell if you prefer and then run these commands.

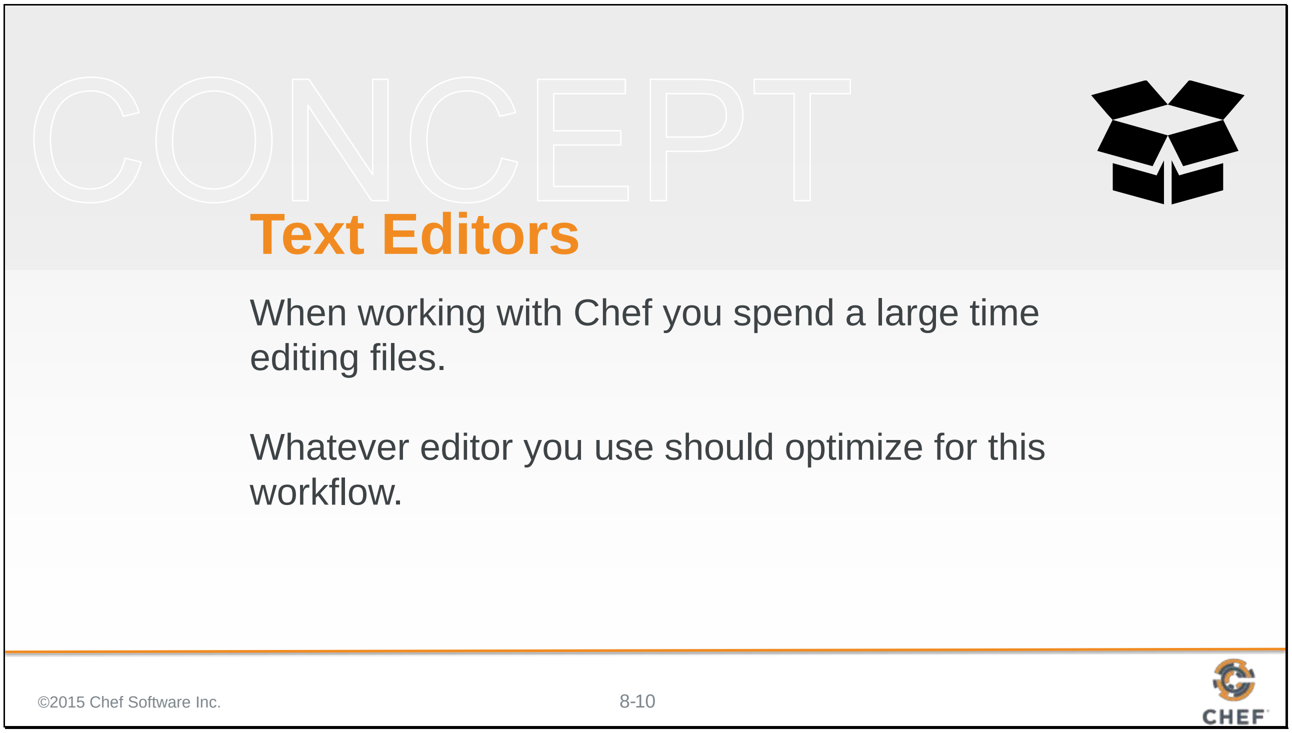
Some of these commands, like 'chef', 'chef-client', 'ohai', and 'kitchen', are the ones that we have used on our remote workstation. Some of these commands you have not seen yet. Later in this course, we'll explore the commands 'knife' and 'berks'. Some of the remaining commands, like 'foodcritic' and 'rubocop', verify the quality of our cookbook code but will not be discussed in the next sections.

All of these commands have the ability to report their versions. This ensures that all the commands are installed properly on your execution path. If any of these commands fail to run this is the time to stop and troubleshoot them.

Slide 9

We used git on the remote workstations. Chef and the Chef community uses git to manage the source code that we write. It is not required that you install git or use git when working with source code. However, we strongly recommend you use a version control tool and if you have not selected one then please install and use git. It's great once you get through the learning curve.

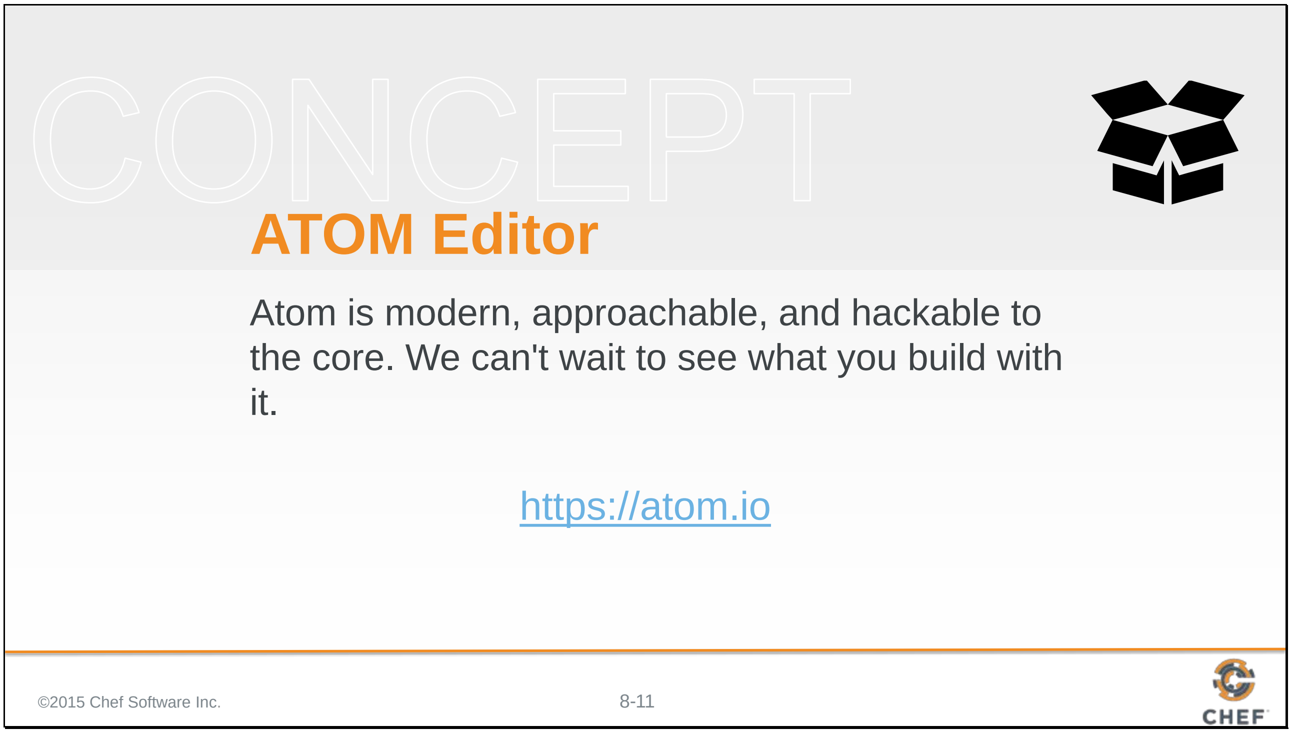
Slide 10



As you have experienced during this introduction to working with Chef, a lot of what you are doing is writing source code in an editor. To work with Chef, you spend a large amount of time editing files, saving your work, and then opening more files. Whatever editor you use should optimize for this workflow.

A large number of basic editors that come standard on your operating system are capable of working with chef: notepad; textedit; kedit; etc. However, they are not always optimized for this workflow.

Slide 11



The Atom editor tool can be customized to do anything, but can also be used productively on the first day without ever touching a config file. Atom is modern, approachable, and hackable to the core. We can't wait to see what you build with it.

You can download Atom at this time, if you don't already have it. You could also use Sublime Text if you already have it.

Slide 12

9: Chef Server

You accomplished a lot so far. You created two cookbooks; one to setup workstations with your tools and a second cookbook that set up a web server that delivered a "Hello, world!" message with some pertinent information about your system.

Slide 2

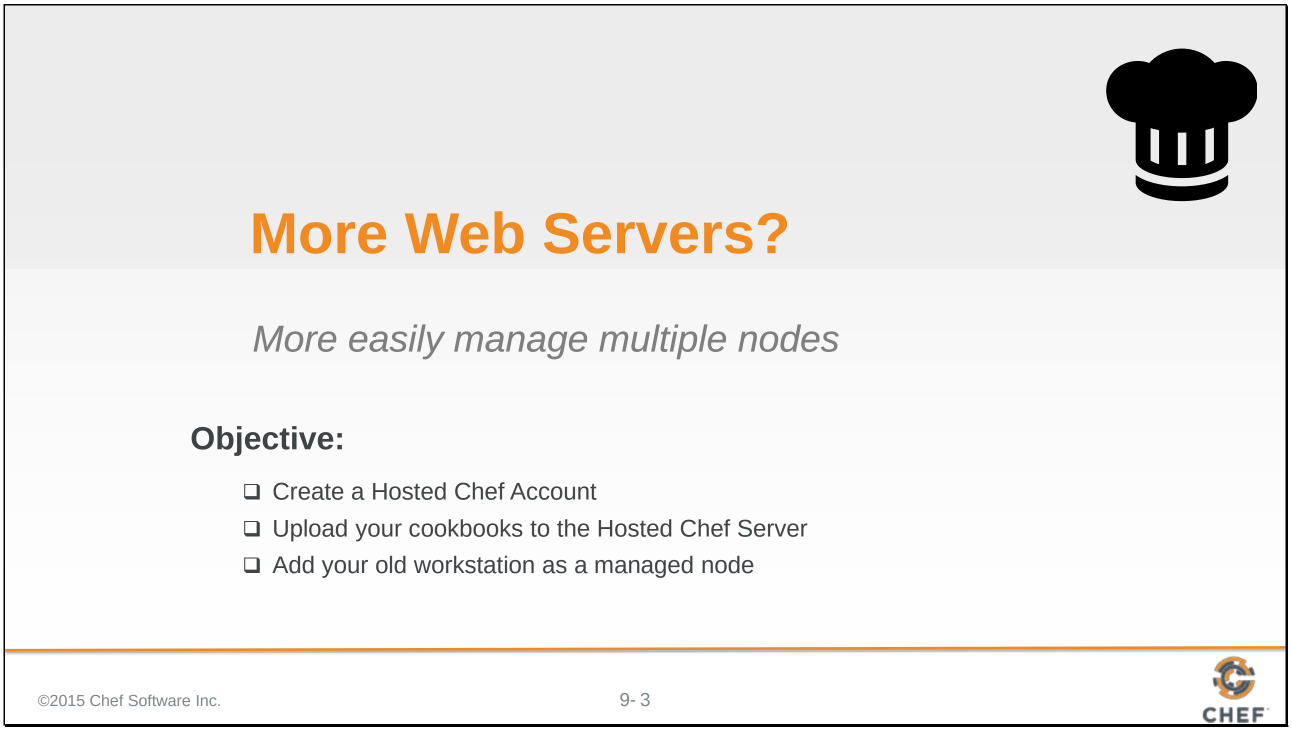
Objectives

After completing this module, you should be able to

- Connect your local workstation (laptop) to a Chef Server
- Upload cookbooks to a Chef Server
- Bootstrap a node
- Manage a node via a Chef Server

In this module you will learn how to connect your local workstation to a Chef Server, upload cookbooks to a Chef Server, bootstrap a node, manage a node via a Chef Server.

Slide 3



Currently, your cookbook exists on one webserver. If you wanted to setup additional web servers to serve additional traffic for your soon-to-be highly successful website, what steps would you need to take to setup an identical system?

Slide 4

Managing an Additional System

To manage another system, you would need to:

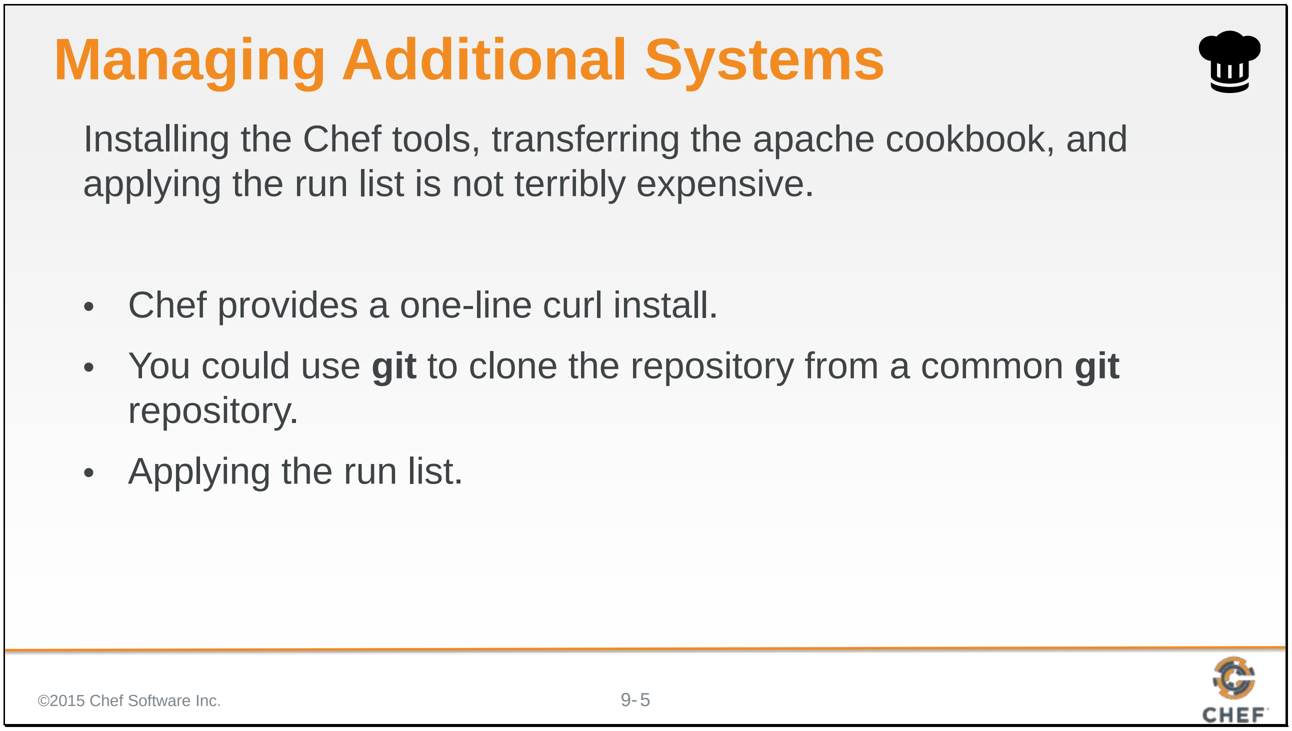
1. Provision a new node within your company or appropriate cloud provider with the appropriate access to login to administrate the system.
2. Install the Chef tools.
3. Transfer the apache cookbook.
4. Run chef-client on the new node to apply the apache cookbook's default recipe.

CHEF

As an exercise, roughly estimate the time it would take to accomplish this series of steps of preparing another node.

- A new system would require us to provision a new node within your company or appropriate cloud provider with the appropriate access to login to administrate the system.
- Install the Chef tools.
- Transfer the apache cookbook.
- Run chef-client locally to apply the apache cookbook's default recipe.

Slide 5



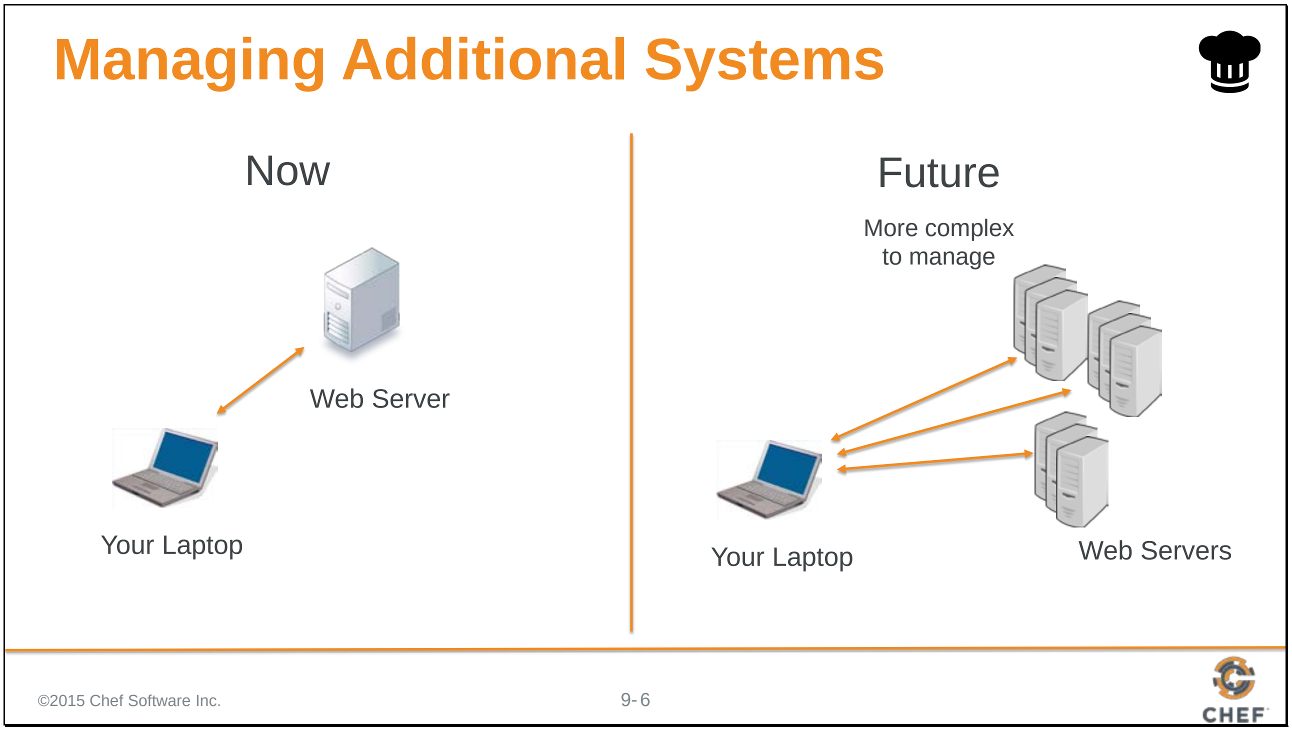
The cost of installing the Chef tools, transferring the apache cookbook, and applying the run list is not terribly expensive.

Chef provides a one-line curl install for the Chef Development Kit (ChefDK).

You could use git to clone the repository from a common git repository. Another option is to archive the cookbook and then using SCP to copy over the contents. A third might be to mount a file share. There are a myriad ways to transfer the cookbooks to the new instance.

Then applying the run list requires the execution of a command on that system.

Slide 6

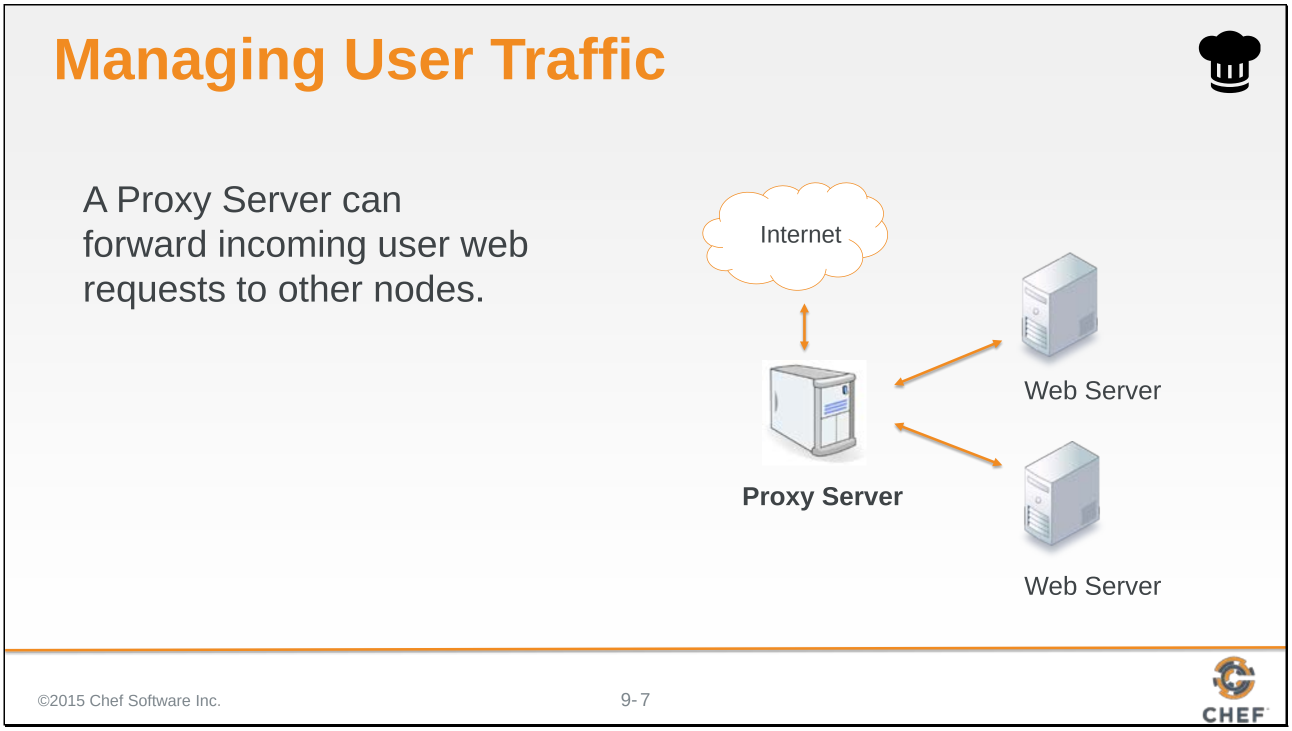


So the overall time required to setup a new instance is not a massive time investment. This manual process will definitely take its toll when requirements demand you manage more than a few additional nodes.

Some may think 10 minutes is not so bad. But what if there were 10 new nodes? 20 new nodes?

As the popularity of your site grows, one server will not be able to keep with all of the web requests. You will need to provision additional machines as demand increases.

Slide 7



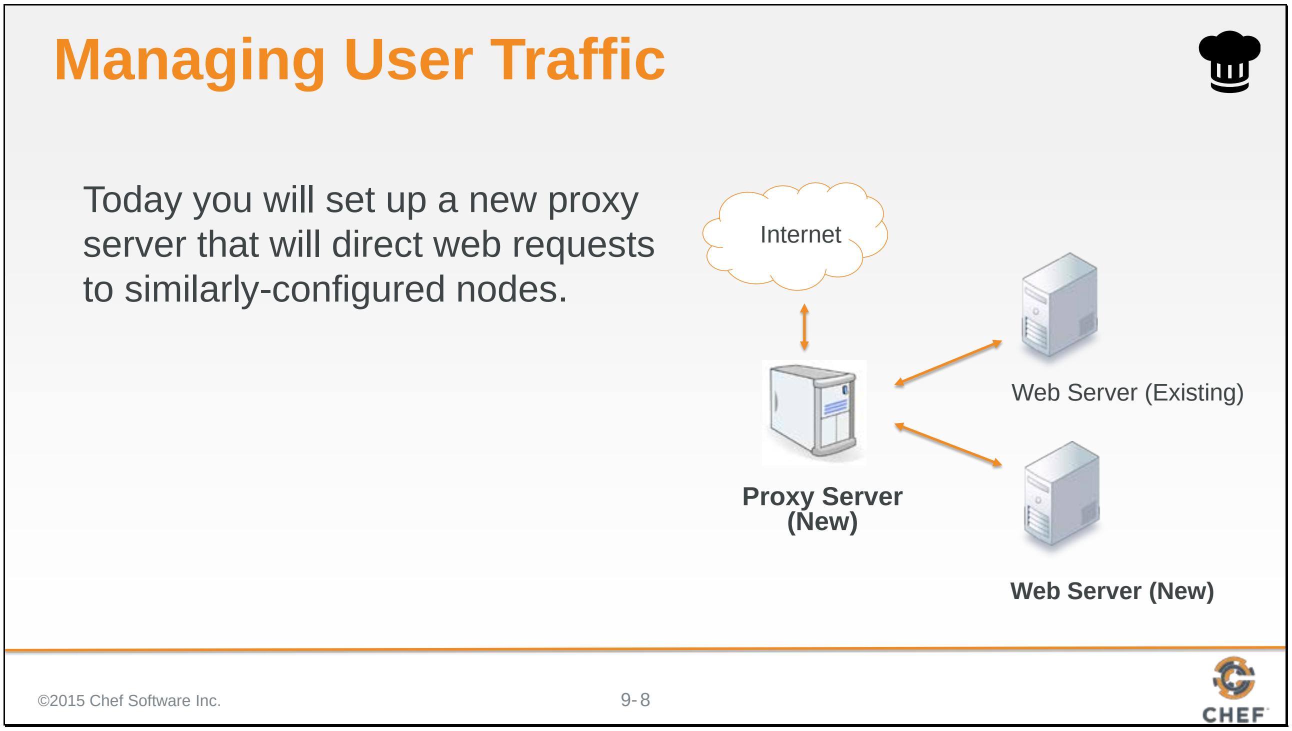
Let's change topics for a moment to managing user web traffic.

In addition to the complexities of configuring and managing multi-server infrastructure, such as web servers, you also need to develop a way to route incoming traffic to each of those web servers and other nodes.

There are many ways that you can route the traffic from one node to a group of similar nodes. This can be done with services by some of the major cloud providers or it can be done with another instance running as a proxy server.

A proxy server allows us to receive incoming requests and forward those requests to other nodes. A proxy server allows us to receive incoming requests and forward those requests to other nodes.

Slide 8



Today you are going to set up a proxy server that will direct web requests to similar configured nodes. Those nodes will be running your default web page that you deploy with the apache cookbook's default recipe.

You have one system already configured as a web server. You will need to set up another web server.

You will also need to set up a node to act as the proxy to both of these web servers.

Slide 9

Steps to Set up Proxy and Web Servers

Web Server

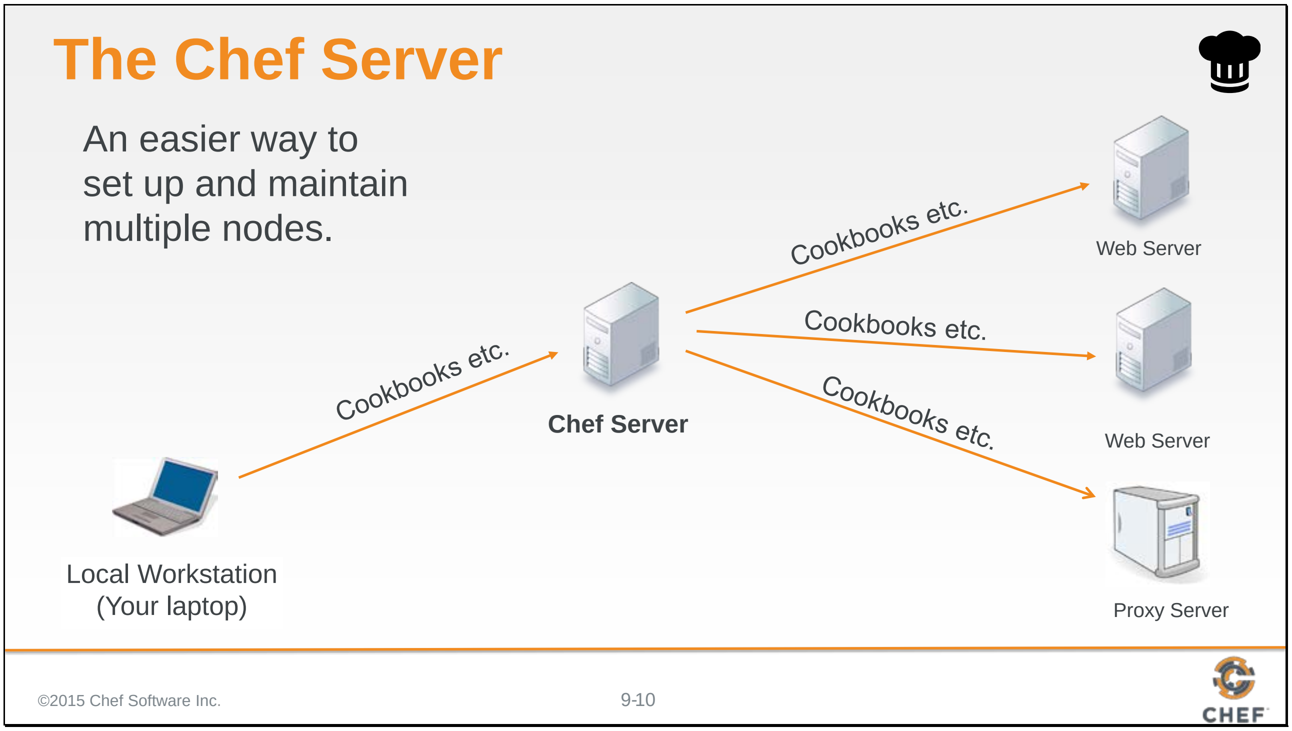
1. Provision the instance
2. Install Chef
3. Copy the Web Server cookbook
4. Apply the cookbook

Proxy Server

1. Create the Proxy cookbook
2. Provision the instance
3. Install Chef
4. Copy the Proxy cookbook
5. Apply the cookbook

Whether you tackle installing, configuring, or running a proxy server or recreate a second instance running the apache cookbook's default recipe, you will need to solve the problem of how you can manage multiple systems. Each system would need to have Chef installed, the cookbooks copied onto each system, and a run list of the recipes to apply to each system.

Slide 10



One way to solve that problem is with a Chef Server.

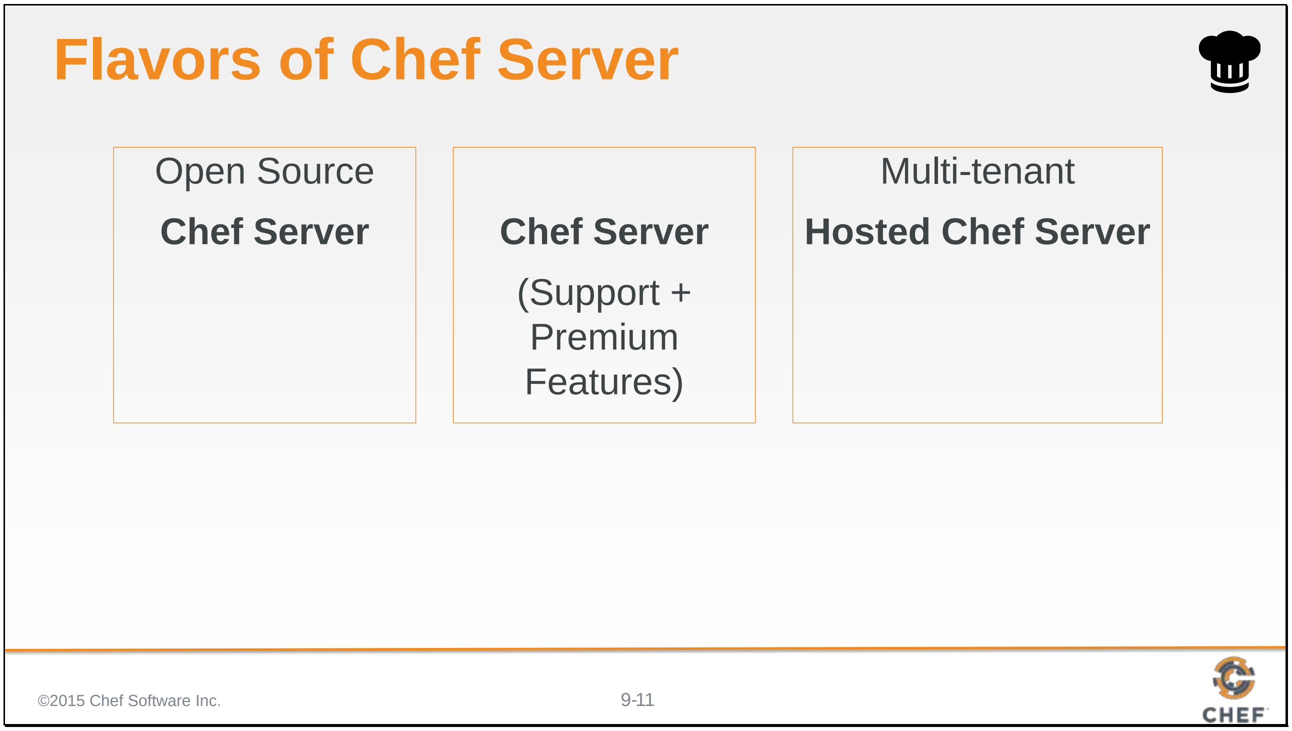
The Chef Server is designed to help us manage multiple nodes in this situation.

The Chef Server acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by 'chef-client'.

Nodes, such as web servers, proxy servers, load balancers, etc., use 'chef-client' to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server). In a production environment, the 'chef-client' runs in an automated mode—it polls the Chef Server for updates at set intervals and the applies any configuration changes.

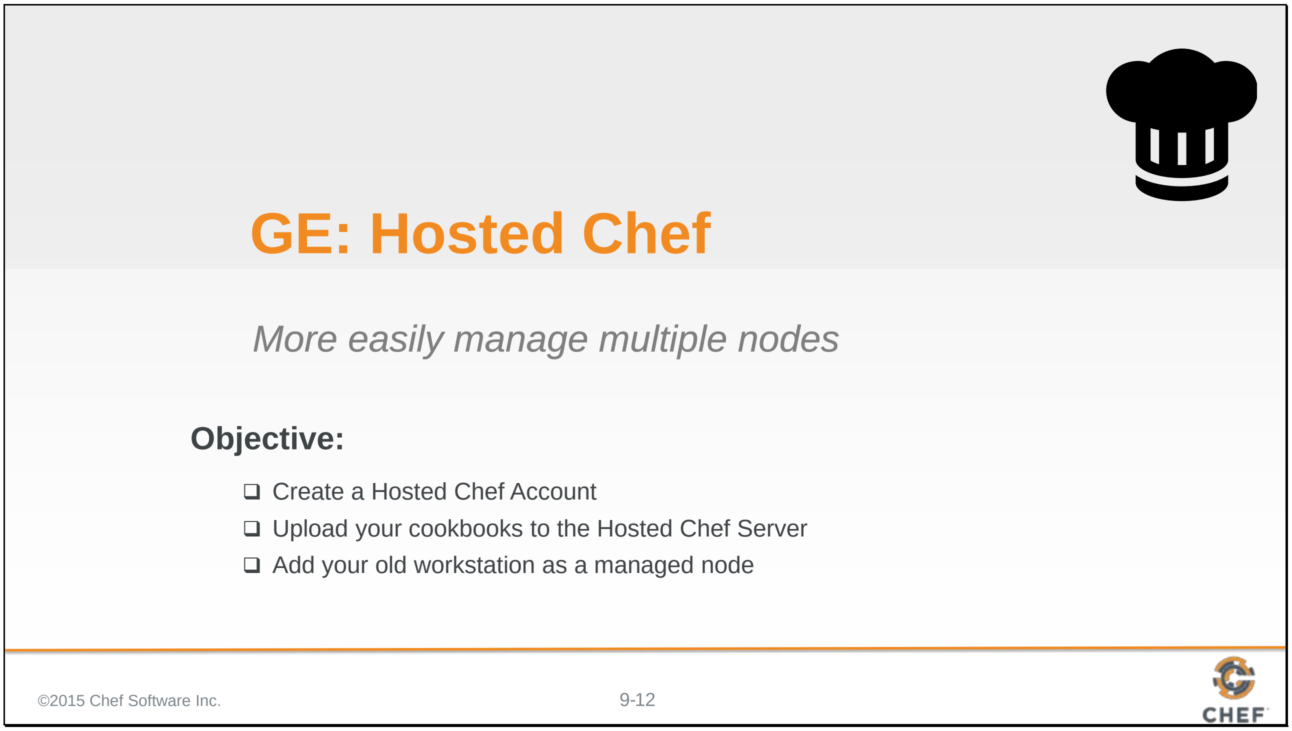
This scalable approach distributes the configuration effort throughout the organization.

Slide 11



At the core we offer Chef Server as an open source project freely available for anyone to deploy. We offer support and additional premium features. Lastly, we have Hosted Chef Server, which is a multi-tenant Chef Server that you host as a service. This by far is the quickest way to get started with and is free as long as you remain under the reasonable node amount.

Slide 12



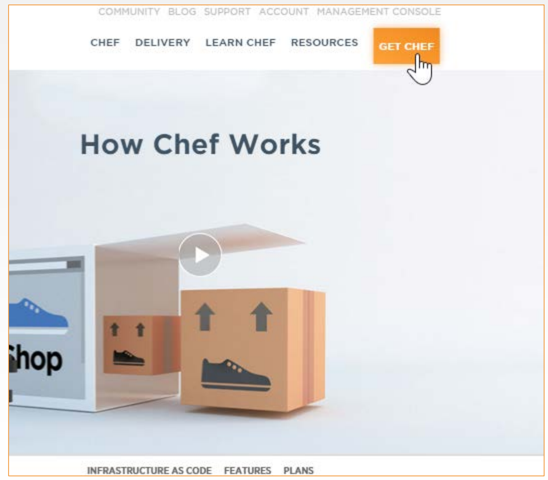
In the interest of getting things done with a relatively small node count, it seems like the Hosted Chef Server option is best.

Slide 13

GE: Signing Up for a Hosted Chef Account

Steps

1. Navigate to https://www.chef.io
2. From the resulting window, click **Get Chef**.



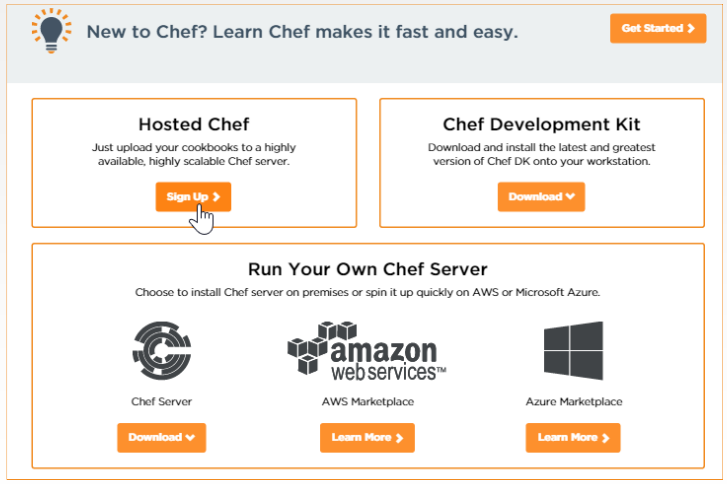
To get started with Hosted Chef Server, visit the Chef website and sign up for a Hosted Chef Account.

Slide 14

GE: Signing Up for a Hosted Chef Account

Steps

3. From the resulting window, click the Hosted Chef **Sign Up** button.



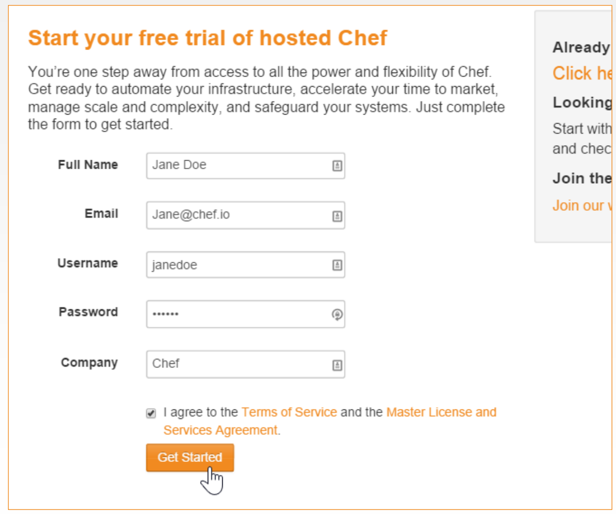
Slide 15

GE: Signing Up for a Hosted Chef Account

Steps

4. Fill out the form as indicated in this image using your name and a valid email address and then click **Get Started**.

Note: You should write down your new user name and remember your password.

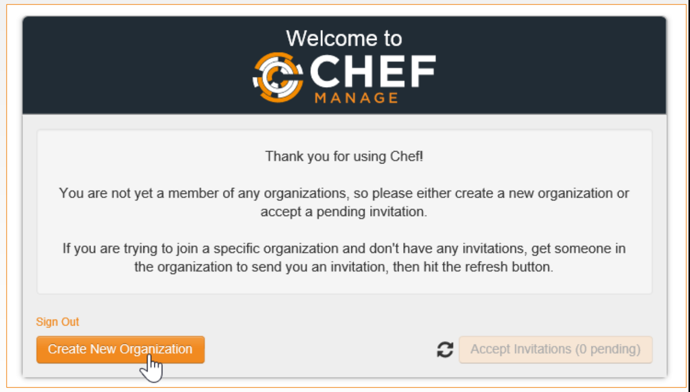


Slide 16

GE: Signing Up for a Hosted Chef Account

Steps

5. From the resulting page, click the **Create New Organization** button.

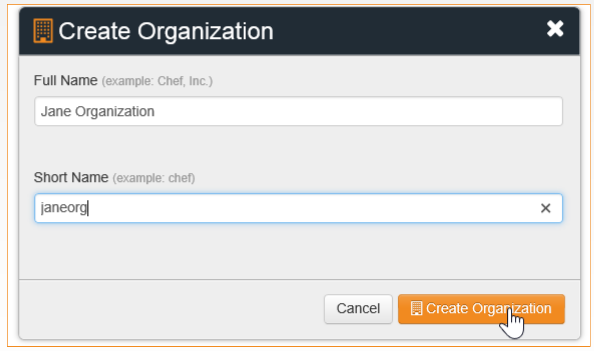


Slide 17

GE: Signing Up for a Hosted Chef Account

Steps

6. Fill out the resulting Create Organization form and then click **Create Organization**.



An organization is a structure within managed Chef that allows multiple companies or entities to exist on the same Chef Server without your paths ever crossing. You might think of it as like setting up a unique username for your organization.

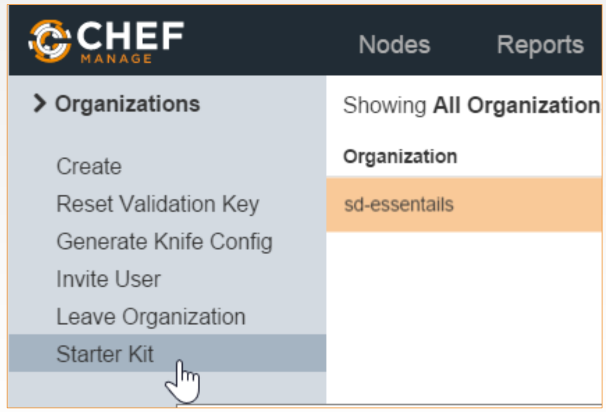
All of the cookbooks, instances and other configuration details that you manage with Chef will be stored on the Chef Server for this particular organization. No other organization will have access to it.

Slide 18

GE: Signing Up for a Hosted Chef Account

Steps

7. From the resulting page, click your new organization to highlight it and then click **Starter Kit**.

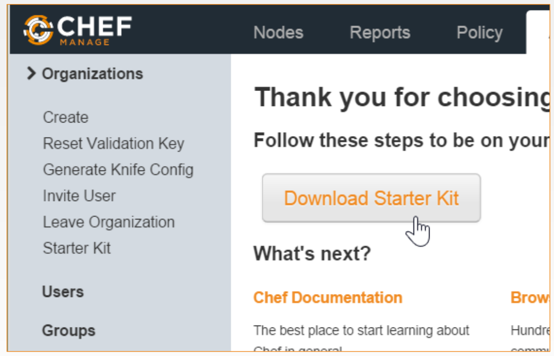


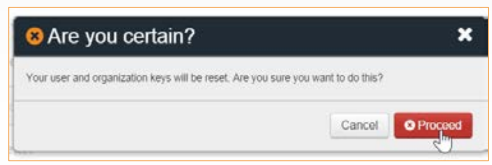
Slide 19

GE: Signing Up for a Hosted Chef Account

Steps

8. From the resulting window, click the **Download Starter Kit** button.
9. Click the **Proceed** button when prompted.





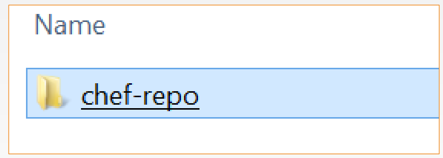
The starter kit will warn that it will reset your organization key and personal key. If this is a new account and new organization this reset is totally fine. If you already have an account or this is an existing organization please understand that you are destroying the existing keys that already exist on a workstation.

Slide 20

GE: Signing Up for a Hosted Chef Account

Steps

10. Open the downloaded zip file and copy chef-repo folder that's contained in the zip file.
11. Paste the chef-repo folder to a location on your laptop, such as your home directory.

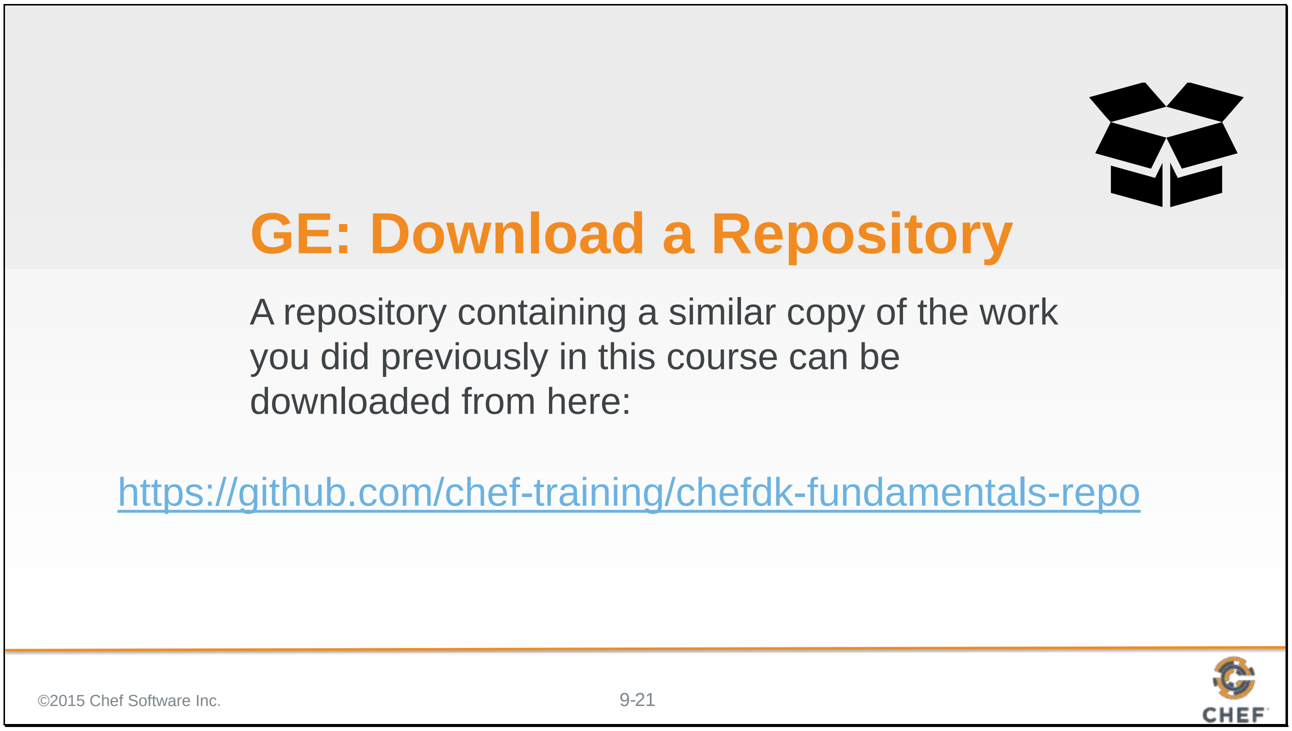


Note: Ensure that the path to the chef-repo does not have a space in it. Examples:

Mac: /home/*username*/chef-repo

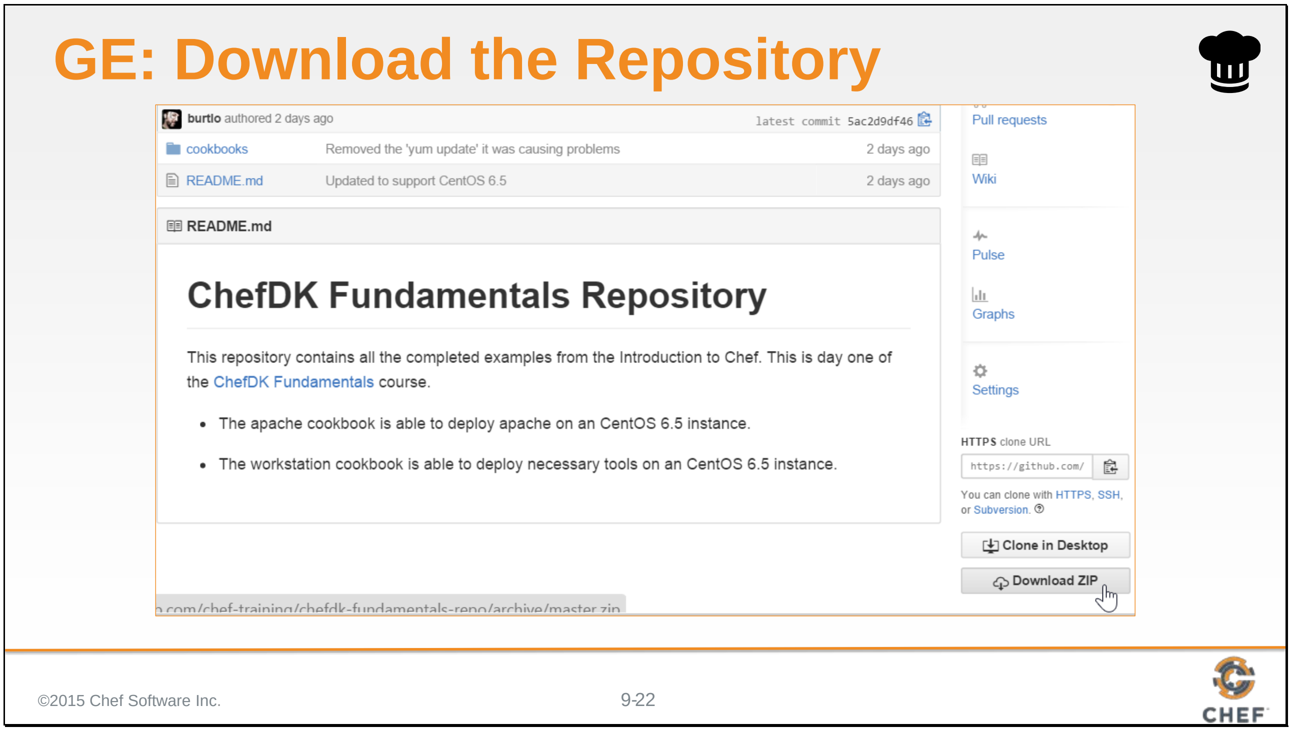
Windows: C:\Users*username*\chef-repo

Slide 21



The cookbooks that were created during the first modules can be found here. These are not the exact cookbooks that you created but ones that have been completed with additional comments and details added.

Slide 22



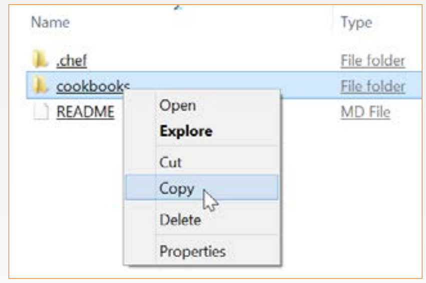
You may clone the repository or download the zip file. Both of those links can be found in the bottom right.

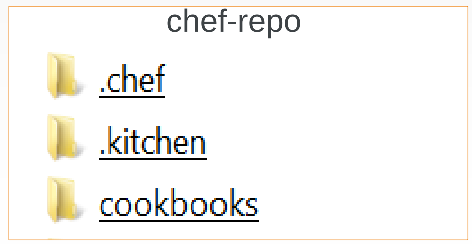
Slide 23

GE: Paste the cookbooks Folder

Steps

- Open the downloaded chefdk-fundamentals-repo-master zip file and then copy **only** the **cookbooks** folder that's contained in the zip file.
- Replace the **cookbooks** folder that's in your chef-repo folder with the copied cookbooks folder.





After you download and open the chefdk-fundamentals-repo archive, copy the included cookbooks folder and paste it into your chef-repo that you unzipped from the Start Kit. Let the new cookbooks folder (that you got from the chefdk-fundamentals-repo) overwrite the existing cookbooks folder that was in your chef-repo folder.

Important: If you had an existing chef-repo prior to class that you want to preserve, save a copy of your old cookbooks folder before pasting the new one into your chef-repo.

Slide 24

GE: Navigate to the chef-repo

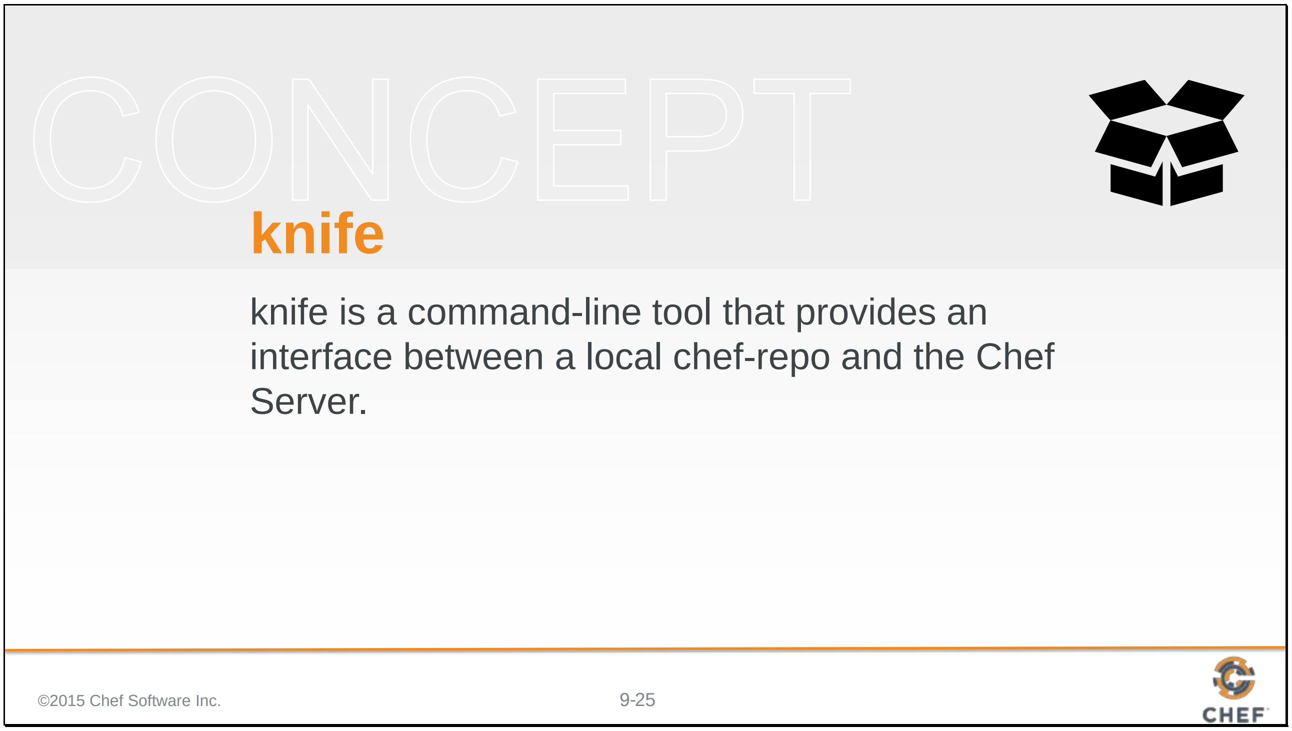
```
$ cd ~/chef-repo
```


The starter kit contains the configuration to reach the Chef Server and your credentials to validate the communicate between your workstation and the Chef Server.

To verify the connection with the Chef Server you will need to run commands within the repository you downloaded.

Open a terminal or command prompt and navigate to the chef-repo directory.

Slide 25



knife is a command-line tool that allows us to request and send information to the Chef Server.

knife helps users manage:

- Nodes
- Cookbooks and recipes
- Roles
- and more

knife provides a number of sub-commands.

Slide 26

GE: knife --help

```
$ knife --help
```

```
Available subcommands: (for details, knife SUB-COMMAND --help)

** BOOTSTRAP COMMANDS **
knife bootstrap FQDN (options)
knife bootstrap windows ssh FQDN (options)
knife bootstrap windows winrm FQDN (options)

** CLIENT COMMANDS **
knife client bulk delete REGEX (options)
knife client create CLIENT (options)
knife client delete CLIENT (options)
knife client edit CLIENT (options)
```


You can look at all the commands with 'knife –help'.

This will display all the sub-commands available. In your case you want to verify that the client list contains a single entry so you need to look for help for the specific command 'knife client --help'.

Slide 27

GE: knife client --help

```
$ knife client --help
```

```
Available client subcommands: (for details, knife SUB-COMMAND --help)

** CLIENT COMMANDS **
knife client bulk delete REGEX (options)
knife client create CLIENT (options)
knife client delete CLIENT (options)
knife client edit CLIENT (options)
knife client list (options)
knife client reregister CLIENT (options)
knife client show CLIENT (options)
```


This will give us an even smaller subset of the commands related specifically to asking the Chef Server about client information. A general command is the list command which will output all the clients that the Chef Server currently maintains.

Slide 28

GE: knife client list

```
$ knife client list
```

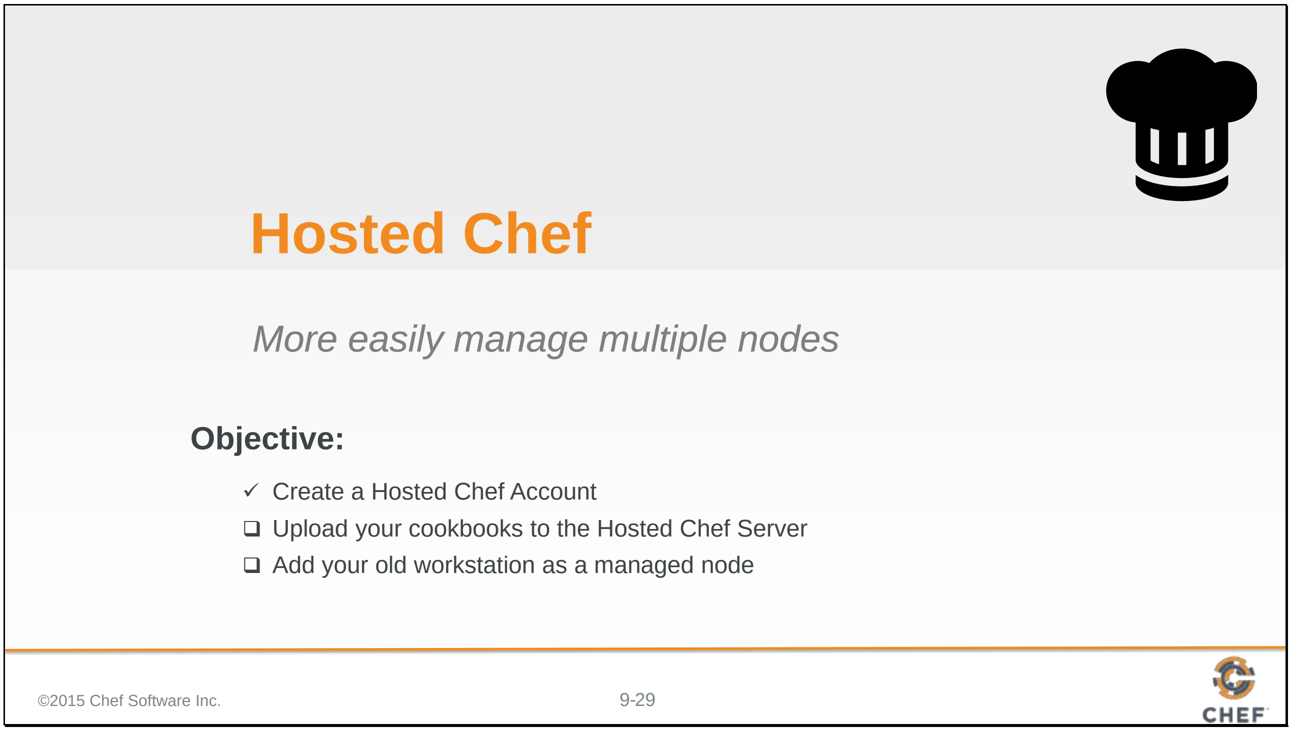
```
ORGNAME-validator
```


For your Chef Server account there should be a single client that is the organization name: validator. This is a special key that has access to the Chef Server. The important thing is that the result does not contain an error with the configuration or authenticating with the Chef Server.

If you receive an error ensure that you:

- typed the command correctly
- executed the command within the chef repository
- are connected to the internet and not blocking ssl connections from your own system's proxy servers or virtual private networks
- have a .chef directory, within the chef repository, which contains the knife configuration file (knife.rb), personal key, and organizational key

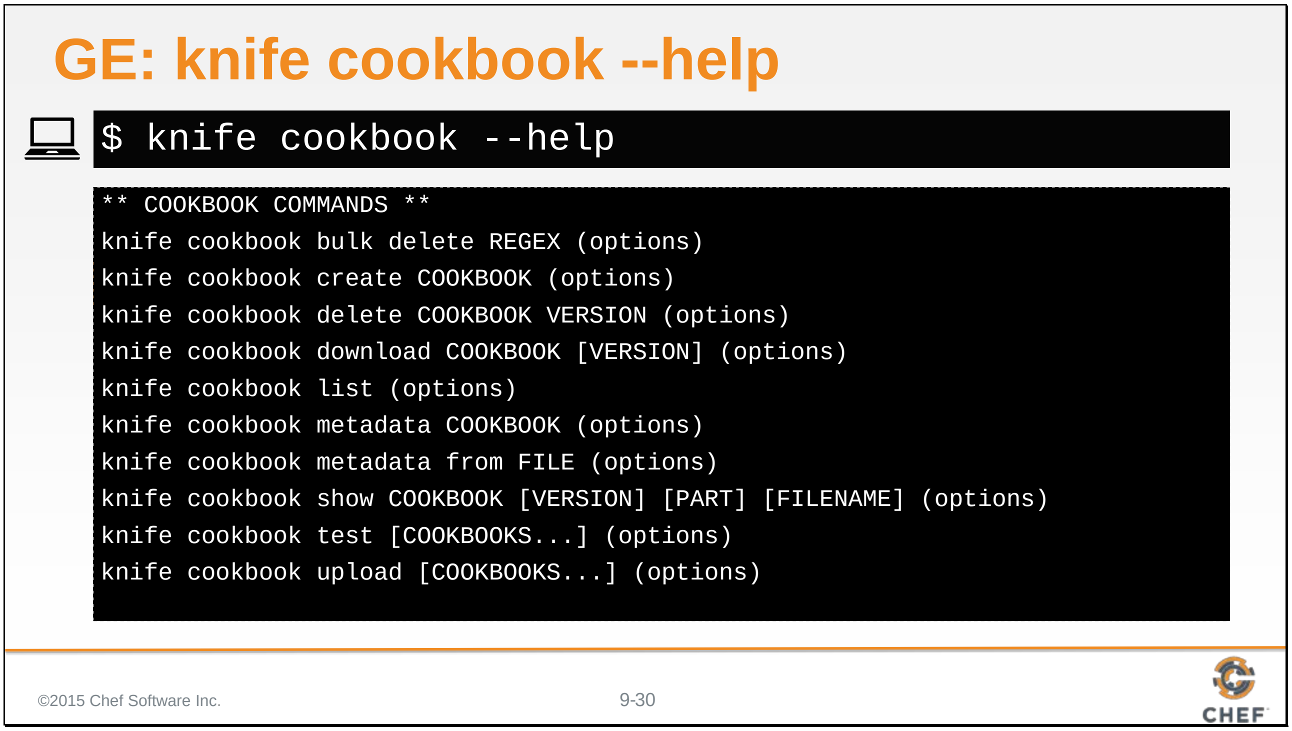
Slide 29



With all that complete, you are now able to communicate with the Chef Server. At this point we will refer to the system in front of you, with the chef repository, the configuration, and the keys installed as your workstation.

When working with Chef with a Chef Server, the workstation is the location where you will compose your cookbook code. When that code is complete, you will then upload it to the Chef Server.

Slide 30



Similar to asking the Chef Server about the list of available clients, you can also ask for information about cookbooks. You can find all the commands related to the cookbooks subcommand by running `knife cookbook --help`.

Similar to the list of clients, you can examine a list of cookbooks.

Slide 31

GE: knife cookbook list

```
$ knife cookbook list
```


Running this command will return the cookbooks currently uploaded to the Chef Server. The empty response should come as no surprise.

You want to change that. So you are going to upload each of your cookbooks to the Chef Server.

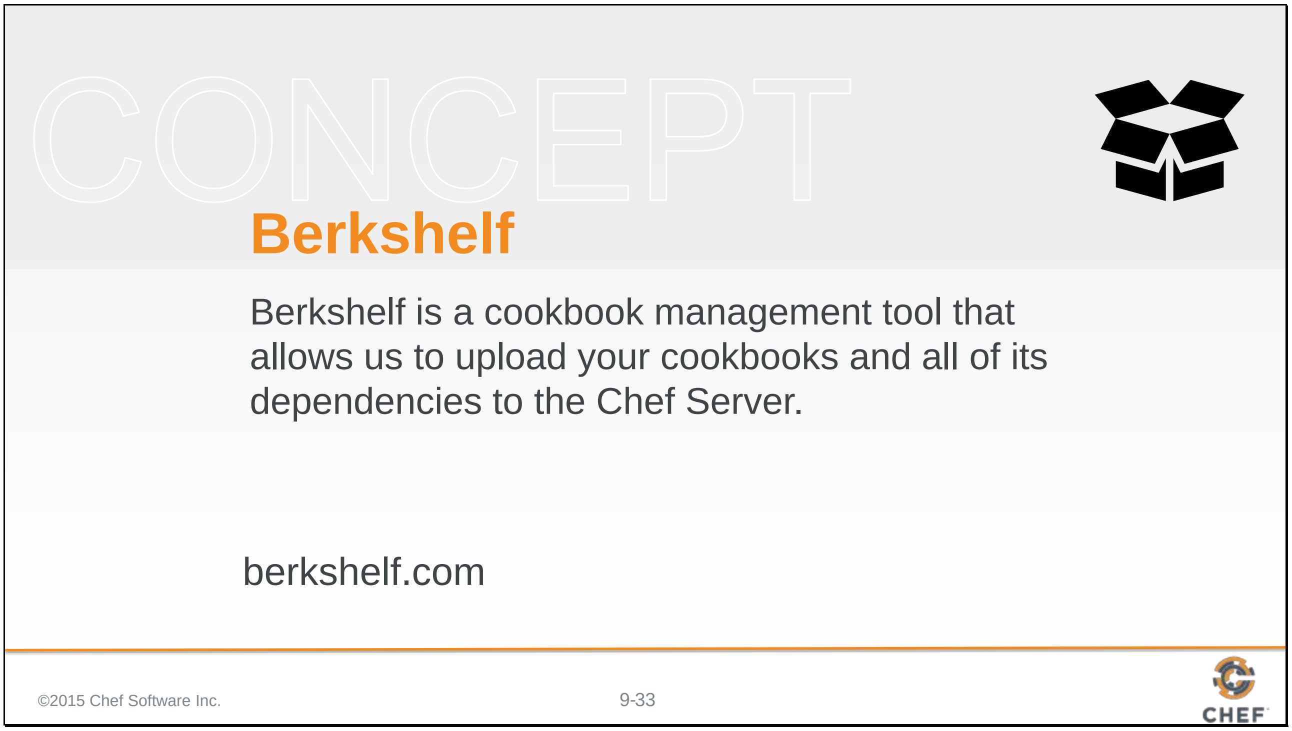
Slide 32

GE: Change to the cookbooks/apache Directory

```
$ cd cookbooks/apache
```


To upload a cookbook to the Chef Server you need to be within the directory of the cookbook. Let us start with the apache cookbook. Change directory into the apache cookbook directory which is within the cookbooks directory.

Slide 33



To upload the cookbook you will need to use another tool called Berkshelf.

Berkshelf is a cookbook management tool that allows us to upload your cookbooks and all of its dependencies to the Chef Server. In this instance, your current cookbooks have no dependencies, but in the future when they do, Berkshelf will assist you in ensuring those are all uploaded.

Slide 34

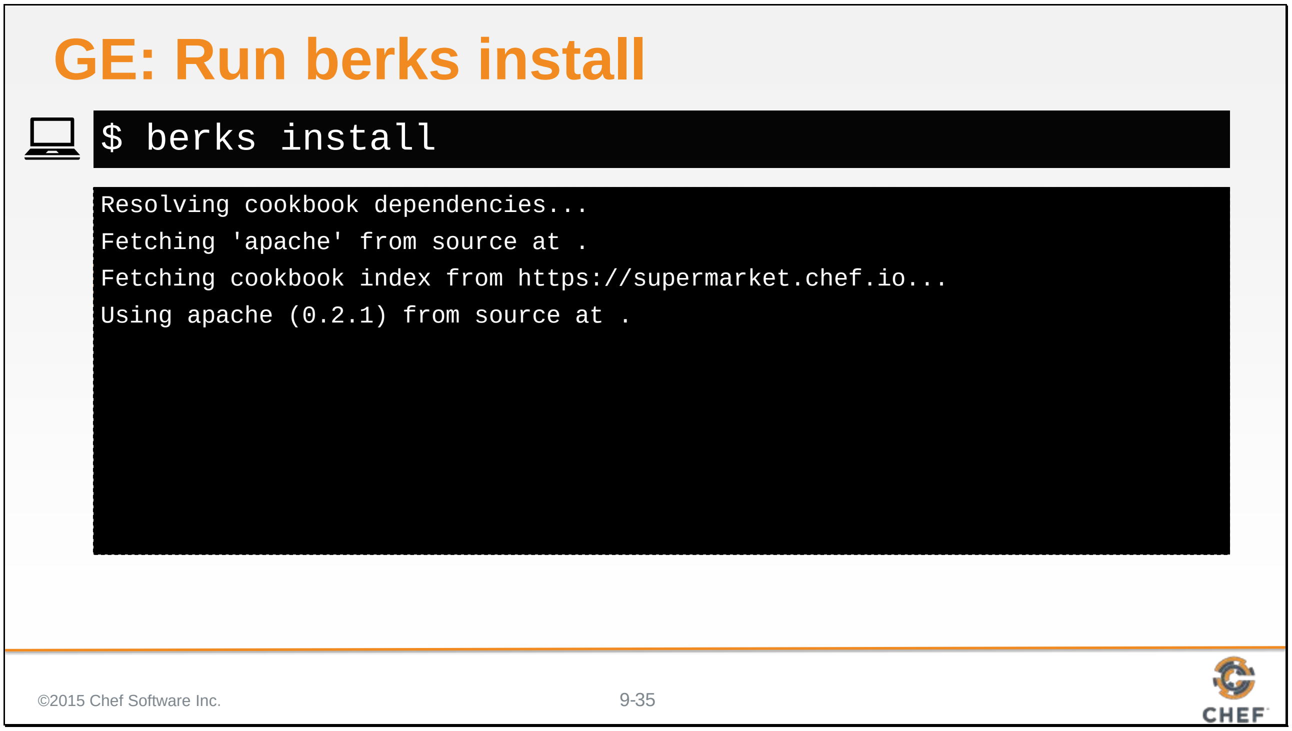
GE: Run berks --help

```
$ berks --help
```

```
Commands:
  berks apply ENVIRONMENT     # Apply version locks from Berksfile.lock to a Chef
environment
  berks contingent COOKBOOK   # List all cookbooks that depend on the given cookbook in
your
  berks cookbook NAME [PATH]  # Create a skeleton for a new cookbook
  berks help [COMMAND]        # Describe available commands or one specific command
  berks info [COOKBOOK]       # Display name, author, copyright, and dependency information
  berks init [PATH]           # Initialize Berkshelf in the given directory
  berks install               # Install the cookbooks specified in the Berksfile
  berks list                  # List cookbooks and their dependencies specified by your
  berks outdated [COOKBOOKS]  # List dependencies that have new versions available that
  berks package [PATH]        # Vendor and archive the dependencies of a Berksfile
  berks search NAME           # Search the remote source for cookbooks matching the partial
```


Berkshelf is a command-line tool that you can ask to see available the commands.

Slide 35



Berkshelf is used on a per-cookbook basis. As dependencies are often per cookbook you'll need to change into the directory of the cookbook.

You should install any dependencies that your cookbook might have. Again, in this instance there are no dependencies external to this cookbook but Berkshelf ensures that this is the case when it runs the 'berks install' command.

You'll see that it finds the current cookbook within your current directory, it contacts the Supermarket for any external dependencies, and then ...

Slide 36

GE: See the Berksfile.lock

```
$ ls -al (or ls -Force if using Powershell)
```

```
drwxr-xr-x 7 chef chef 4096 Aug 27 18:44 .
drwxr-xr-x 4 chef chef 4096 Aug 27 16:17 ..
drwxr-xr-x 8 chef chef 4096 Aug 27 16:07 .git
-rw-r--r-- 1 chef chef  126 Aug 27 15:46 .gitignore
drwxr-xr-x 3 chef chef 4096 Aug 27 18:45 .kitchen
-rw-r--r-- 1 chef chef  183 Aug 27 18:44 .kitchen.yml
-rw-r--r-- 1 chef chef   47 Aug 27 15:46 Berksfile
-rw------- 1 chef chef   77 Aug 27 18:45 Berksfile.lock
-rw-r--r-- 1 chef chef   54 Aug 27 15:46 README.md
-rw-r--r-- 1 chef chef  974 Aug 27 15:46 chefignore
-rw-r--r-- 1 chef chef  198 Aug 27 15:46 metadata.rb
drwxr-xr-x 2 chef chef 4096 Aug 27 16:34 recipes
```


...it completes by writing a Berksfile.lock to the file system.

The Berksfile.lock is a receipt of all the cookbooks and dependencies found at the exact moment that you ran 'berks install'.

Slide 37

GE: See the Contents of the Berksfile.lock

```
$ cat Berksfile.lock
```

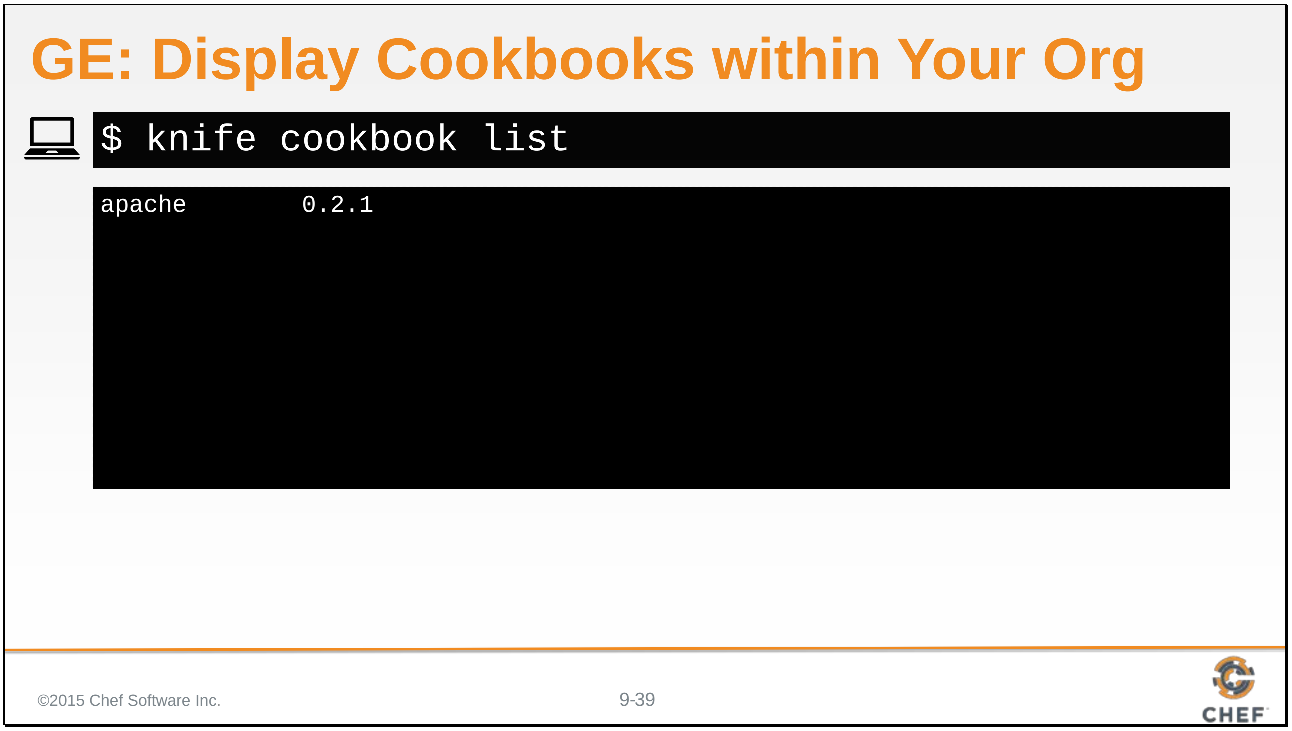
```
DEPENDENCIES
  apache
    path: .
    metadata: true

GRAPH
  apache (0.2.1)
```


This lock file is useful to ensure that in the future you use the same dependencies when working with the cookbook.

Slide 38

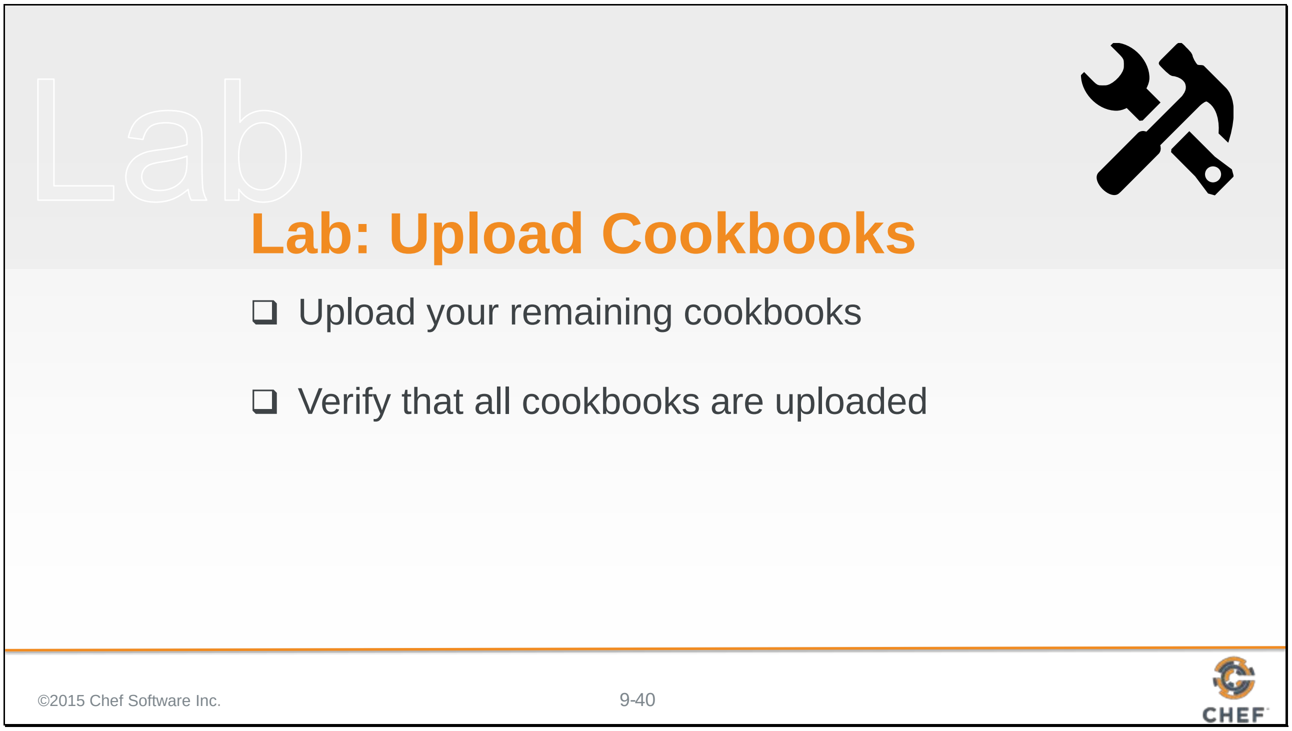
With the dependencies accounted for, it is time to upload the to the Chef Server. This is another sub-command that Berkshelf provides called 'upload'. Run the command to upload the apache cookbook to the Chef Server.

Slide 39



When that is complete you can return to the cookbook command that allows you to display the cookbooks within your organization by running this command. This will show you that the Chef Server has the apache cookbook that you have uploaded.

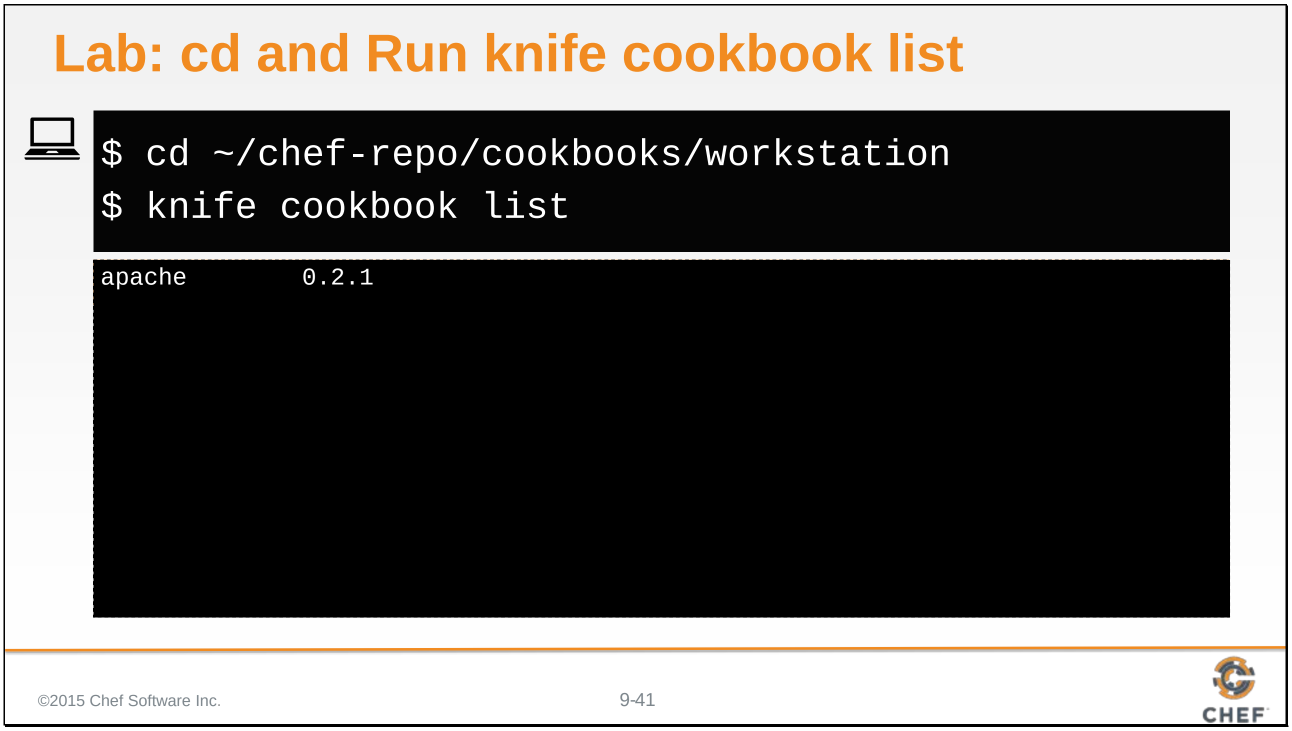
Slide 40



As an lab upload the remaining cookbooks within the cookbooks directory. After you have done that verify that the cookbooks have been uploaded.

.

Slide 41



The one remaining cookbook is the workstation cookbook. Berkshelf is a cookbook management tool that examines the contents and dependencies of a single cookbook.

- Change into the cookbooks directory.
- Verify that the cookbook is not currently uploaded.

Slide 42

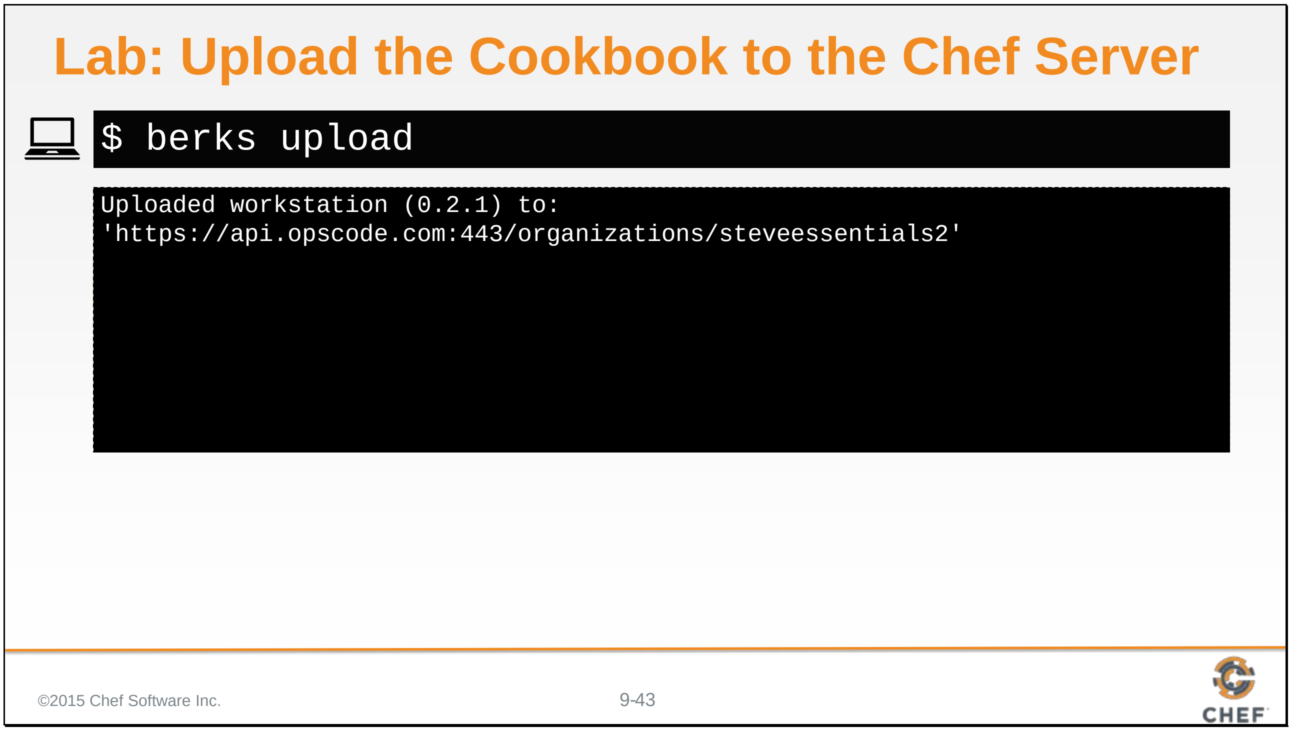
Lab: Install the Cookbook Dependencies

```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'workstation' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using workstation (0.2.1) from source at .
```

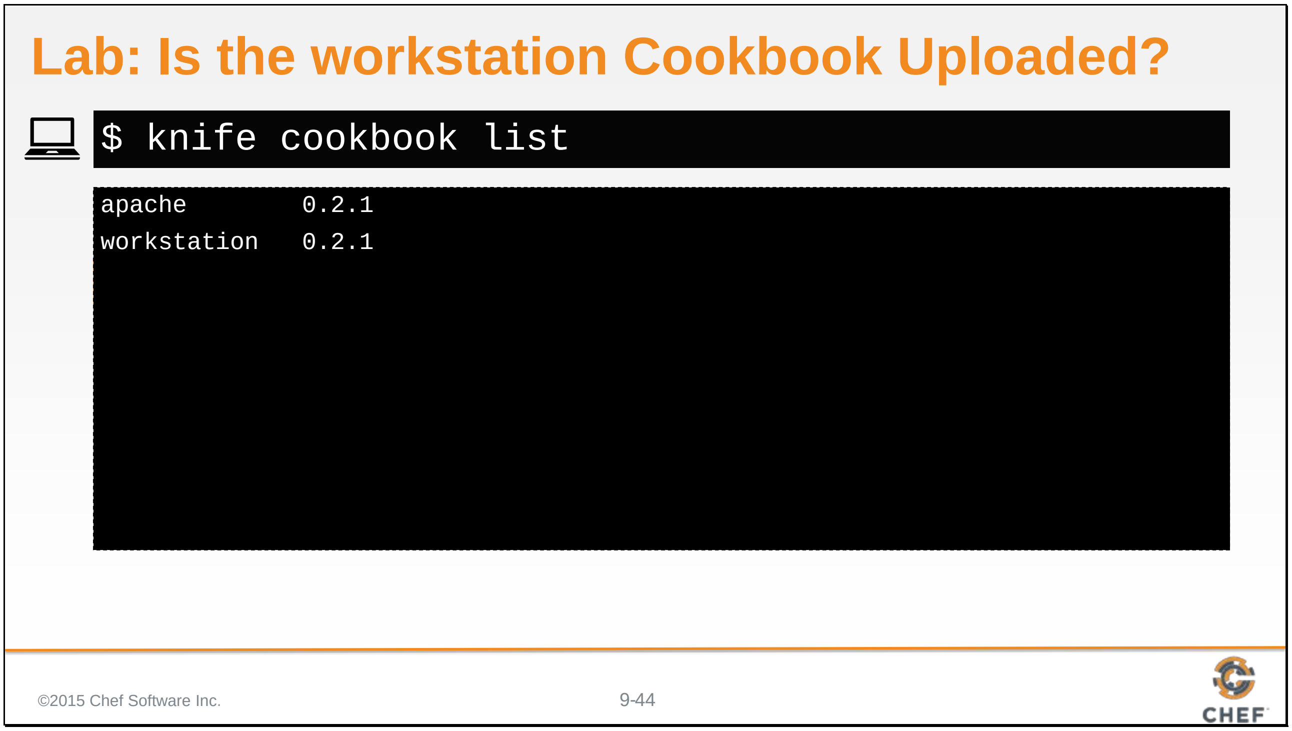

Run "berks install" to install all the cookbook dependencies.

Slide 43



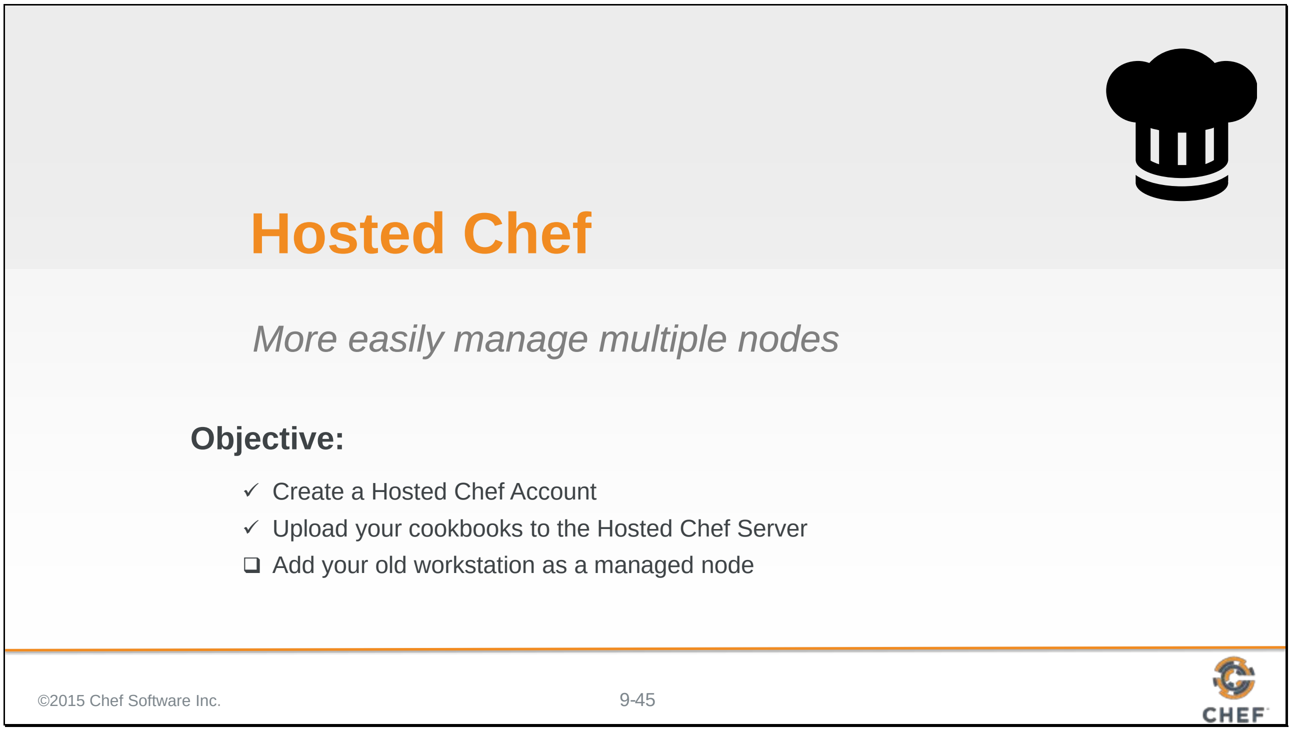
Run "berks upload" to upload the cookbook and all its dependencies to the Chef Server.

Slide 44



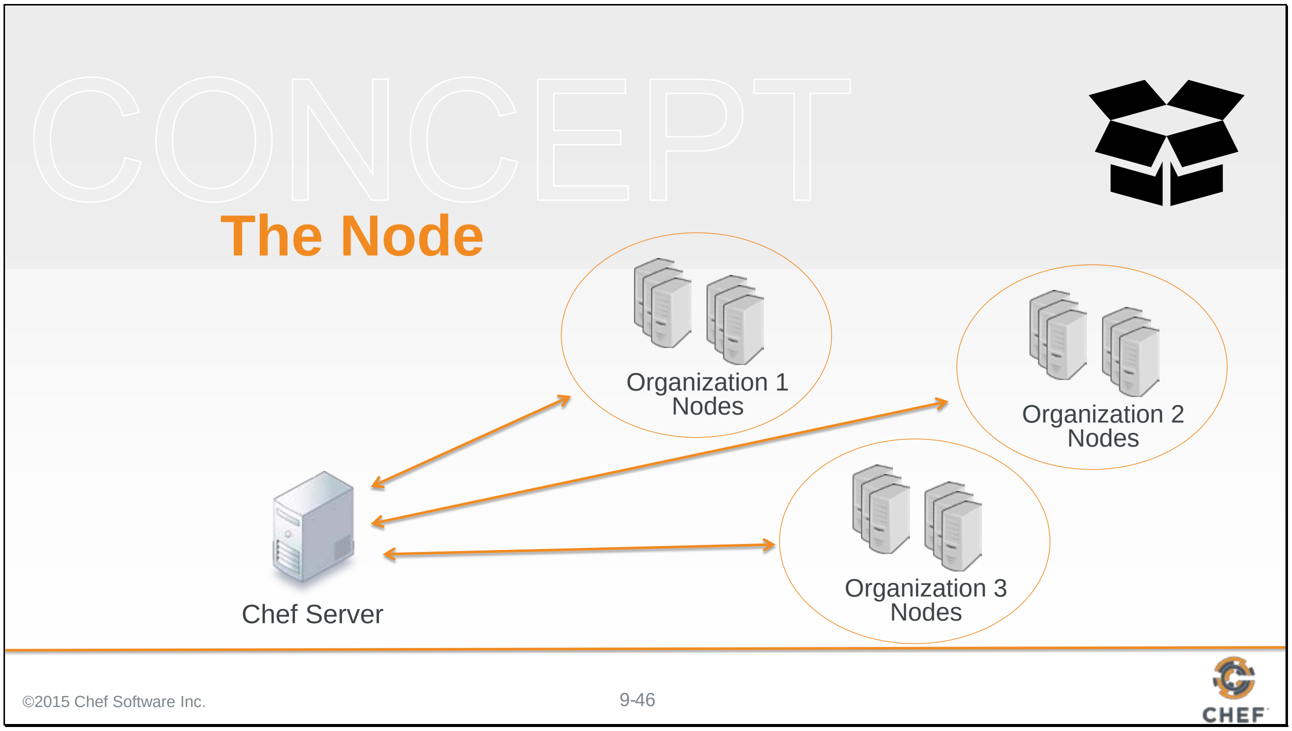
Lastly, run "knife cookbook list" to validate that the workstation cookbook is now uploaded to the Chef Server.

Slide 45



You have one remaining objective and that is to add an instance as a node within your organization.

Slide 46



As you know by now, a node is a server that Chef is managing. A node could be a web server, an application server, a database server, a proxy server, and so on.

A node can only join one organization. To be a node means that it has Chef installed, has configuration files in place, and when you run the chef-client application with no parameters it will successfully contact the Chef Server and ask it for the run list that it should apply and the cookbooks required to execute that run list.

When a node is part of the organization you manage that information on the Chef Server as well. A Chef Server can manage multiple organizations. Managing that information in a Chef Server allows us to use for inventory, querying and searching.

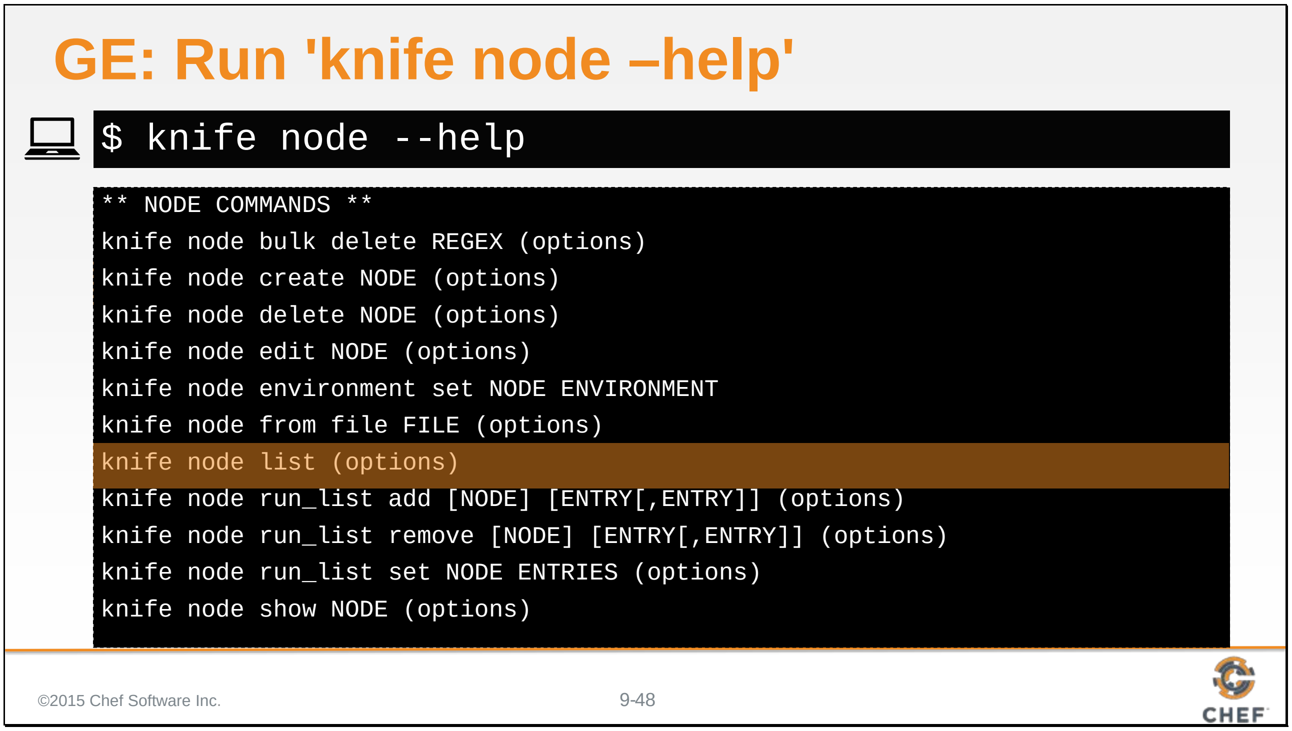
Slide 47

GE: Change to the chef-repo

```
$ cd ~/chef-repo
```

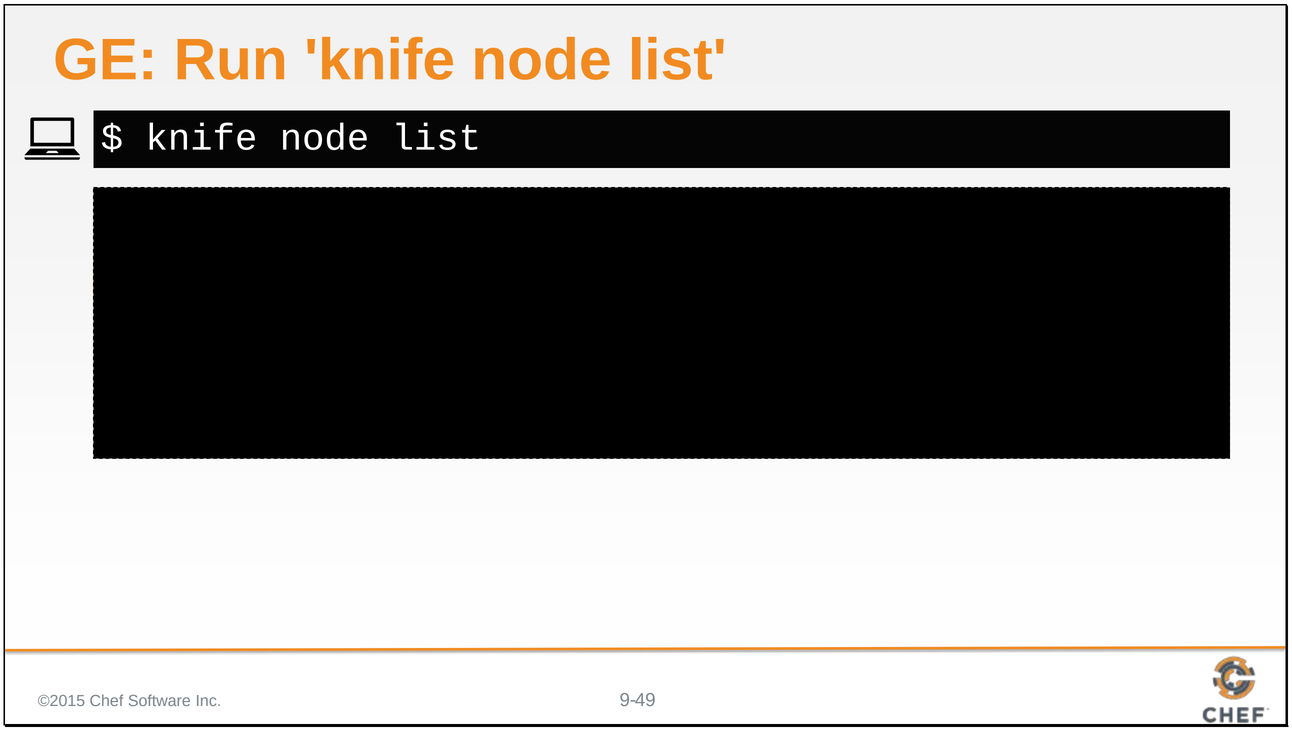

Let's add the instance we used previously as a workstation now as a managed node. Return to the root of the chef repository.

Slide 48



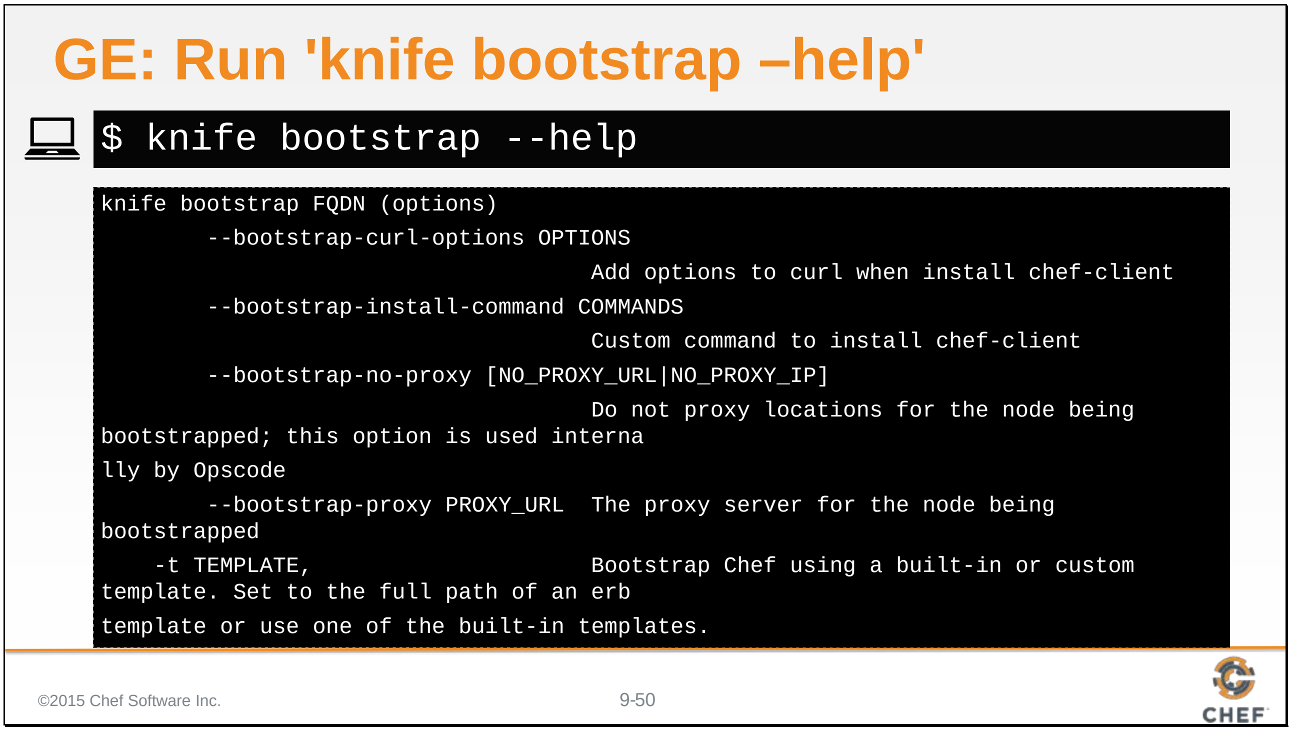
Verify that you have no existing nodes within your organization. You can use the 'knife node –help' command to see that you can ask for the list of all nodes within your organization with the list command.

Slide 49



Run "knife node list" to see that you have no nodes currently registered with your Chef Server. At this point the results should be blank.

Slide 50

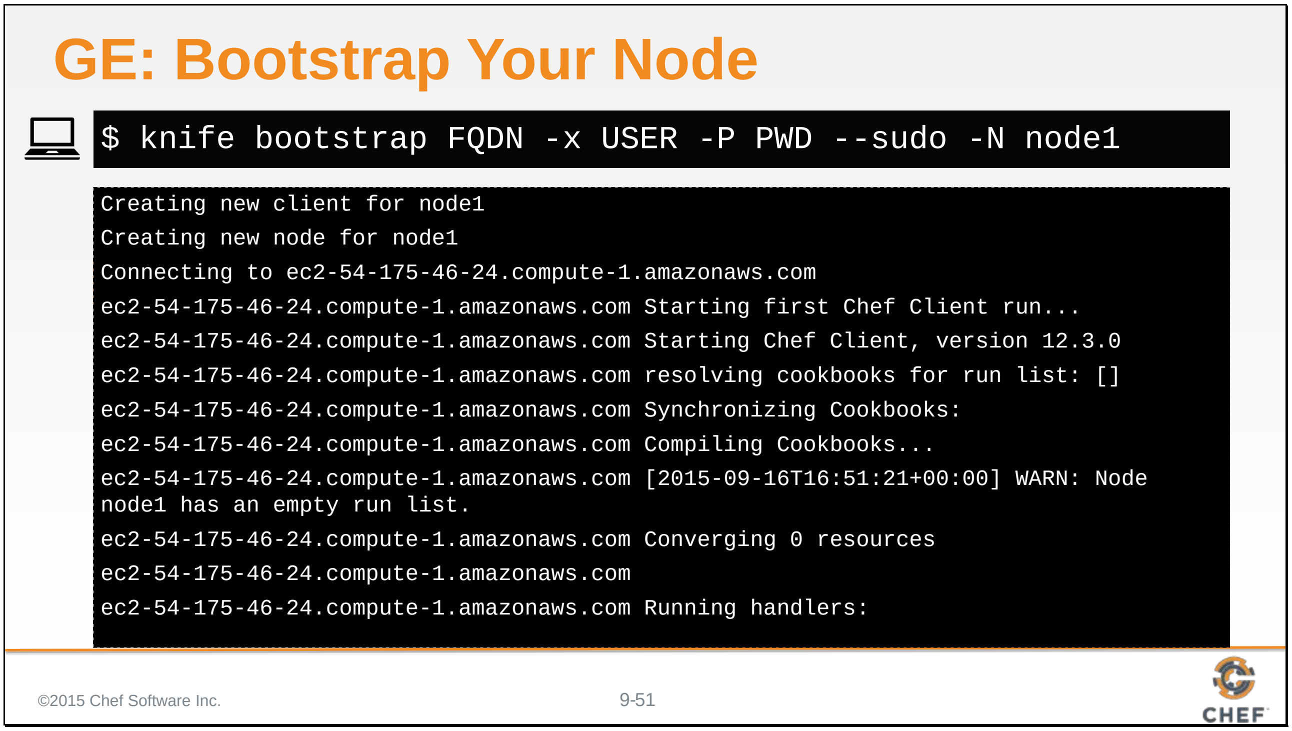


Knife provides a bootstrap subcommand that takes a number of options.

When you bootstrap an instance it is performing the following:

* Installing chef tools if they are not already installed
* Configuring Chef to communicate with the Chef Server
* Running chef-client to apply a default run list

Slide 51



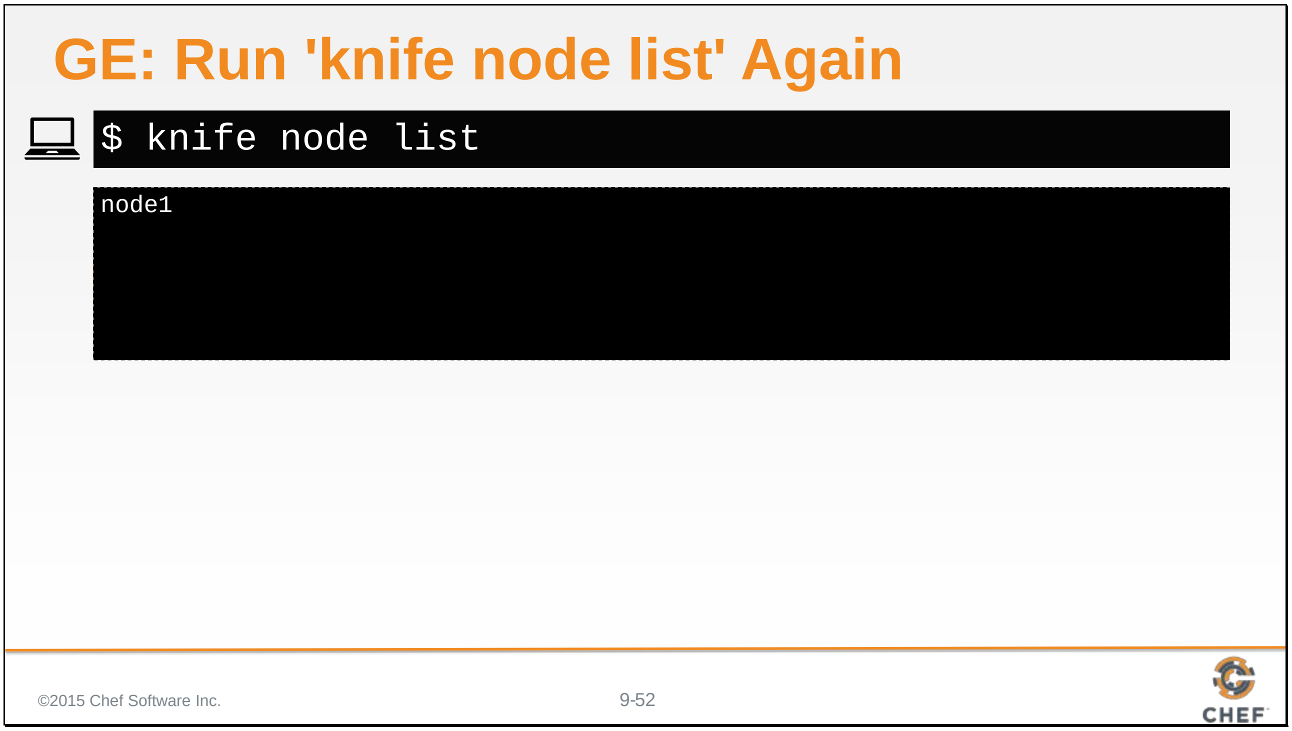
To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-x' flag and the password '-P' flag.

Include the '--sudo' flag because you are installing software and writing configuration to directories traditionally owned by the root user.

Name the node with the '-N' flag. This is optional but makes it easier for us to communicate. When we ask you to look at the details of node 1 or login to node 1, it will be easier to remember than the fully-qualified domain name.

When executing the command, the output will tell us what it installed and ran.

Slide 52



When bootstrapping is done, you can see that your organization knows about the new node by again running the command "knife node list". You now see that you have a new node, node1, uploaded to the Chef Server.

Slide 53

GE: View More Information About Your Node

```
$ knife node show node1
```

```
Node Name:   node1
Environment: _default
FQDN:        ip-172-31-8-68.ec2.internal
IP:          54.175.46.24
Run List:
Roles:
Recipes:
Platform:    centos 6.7
Tags:
```


CHEF

You can see more information about a particular node with the command 'knife node show node1'. This will display a summary of the node information that the Chef Server stores.

Slide 54

GE: Add a Recipe to a Run List

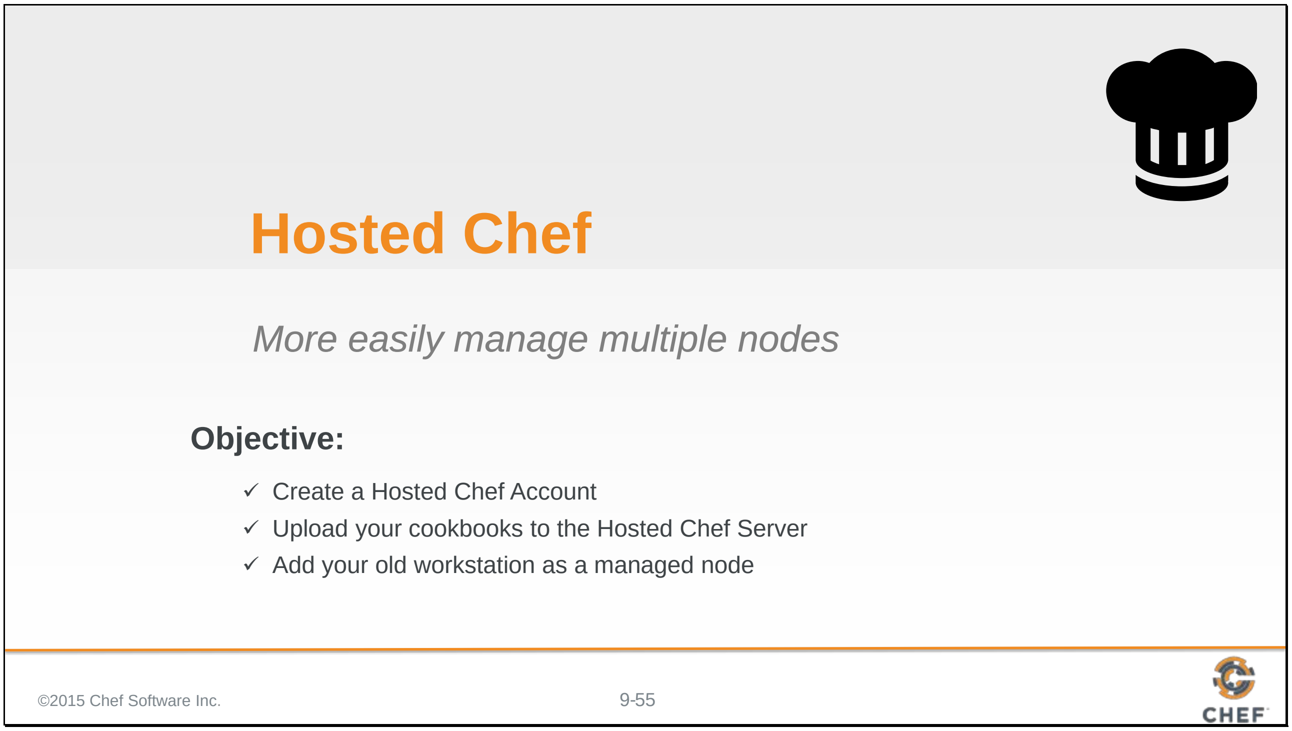
```
$ knife node run_list add node1 "recipe[apache]"
```

```
node1:
  run_list: recipe[apache]
```

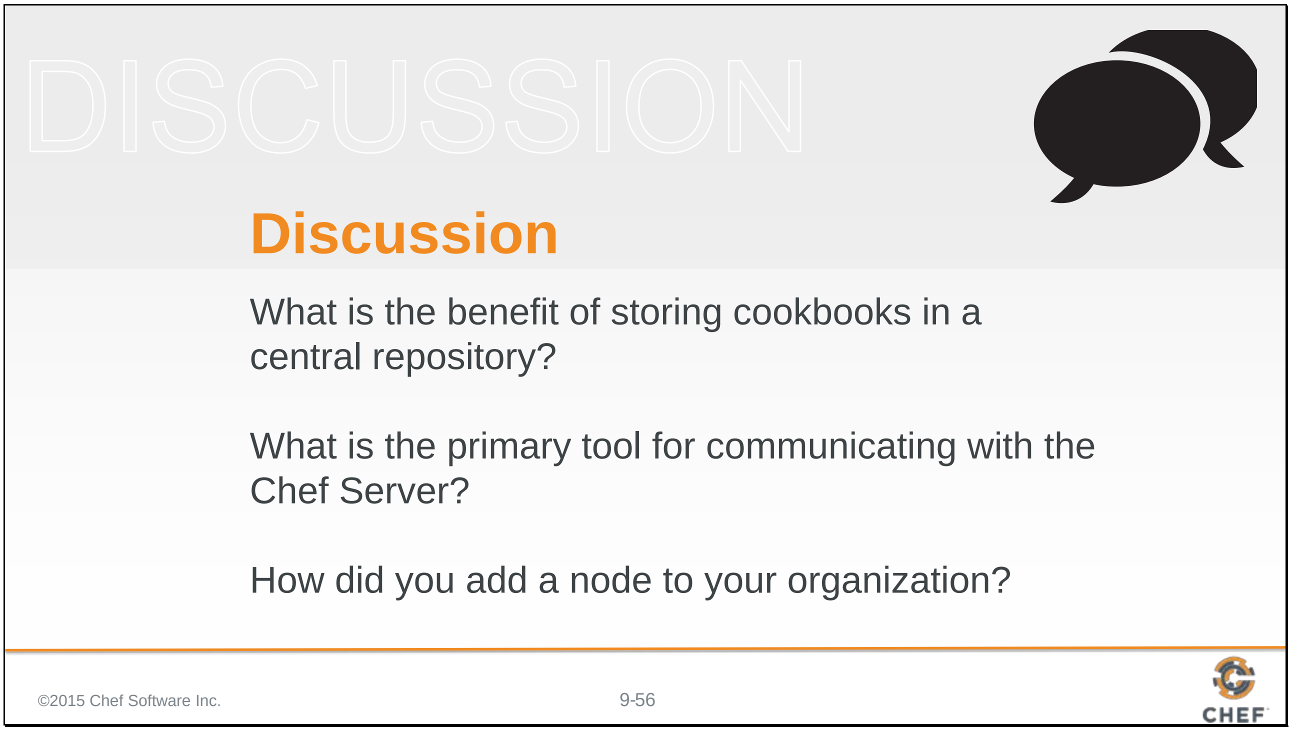

node1 does not have a list of recipes that it applies to the system by default. You can make Chef Server tell node1 to apply a specific run-list the next time node 1 runs 'chef-client'.

You can do that through the 'knife node run_list add' command. In this example, you are adding to node1's run-list the apache cookbook's default recipe.

Slide 55


Hosted Chef
More easily manage multiple nodes
Objective:
✓ Create a Hosted Chef Account
✓ Upload your cookbooks to the Hosted Chef Server
✓ Add your old workstation as a managed node
©2015 Chef Software Inc.
9-55
CHEF

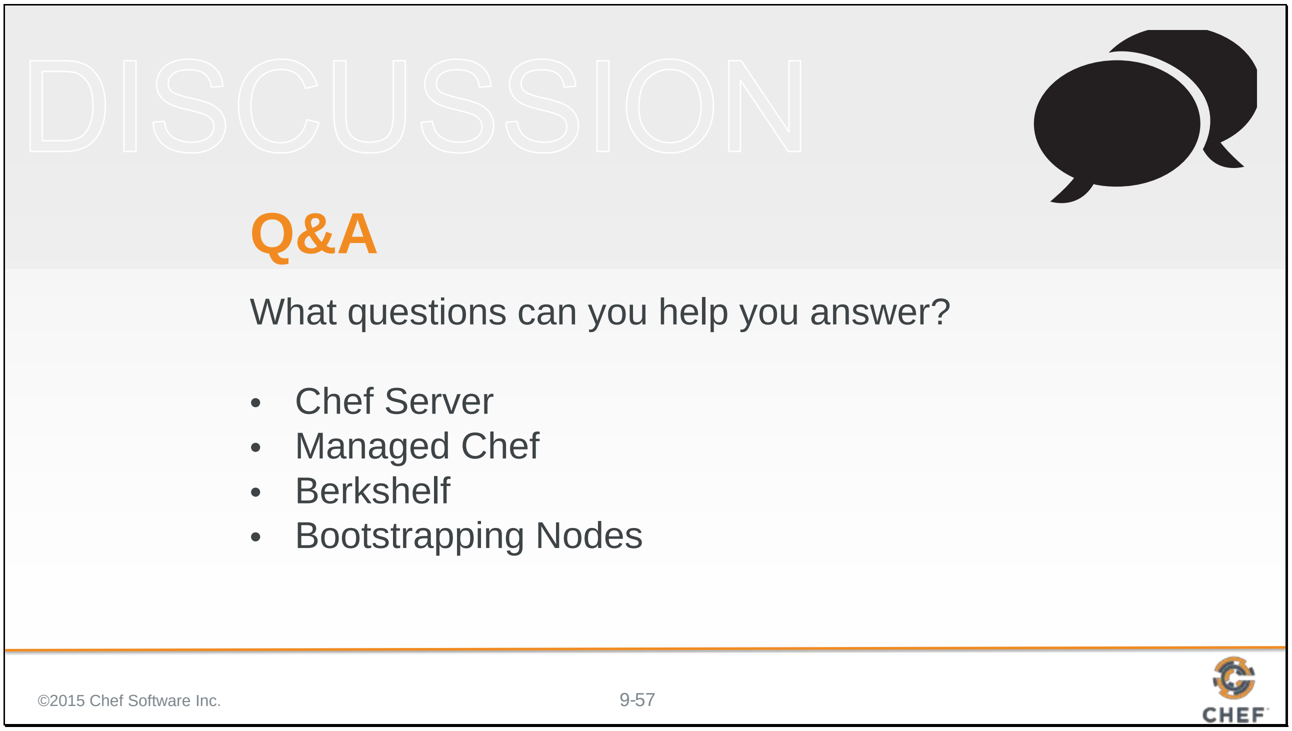
Slide 56



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 57



With all of the objectives complete you are finished with this section. What question can you answer for you?

Slide 58

10: Community Cookbooks

Slide 2

Objectives

After completing this module, you should be able to

- Find cookbooks on the Chef Super Market
- Create a wrapper cookbook
- Replace the existing default values
- Upload a cookbook to Chef Server
- Bootstrap a new node that runs the cookbook

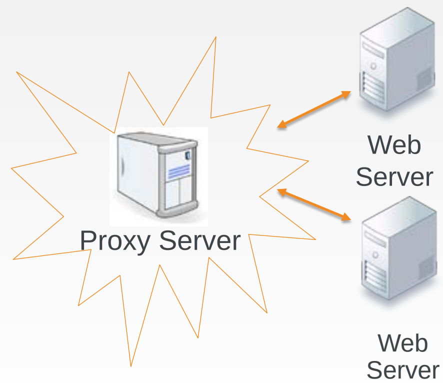
In this module you will learn how to find cookbooks on the Chef Super Market, create a wrapper cookbook, replace the existing default values, upload a cookbook to Chef Server, and bootstrap a new node that runs the cookbook.

Slide 3

Proxy Server

Adding a proxy will allow us to better grow our infrastructure.

Receives requests and relays them to other systems.

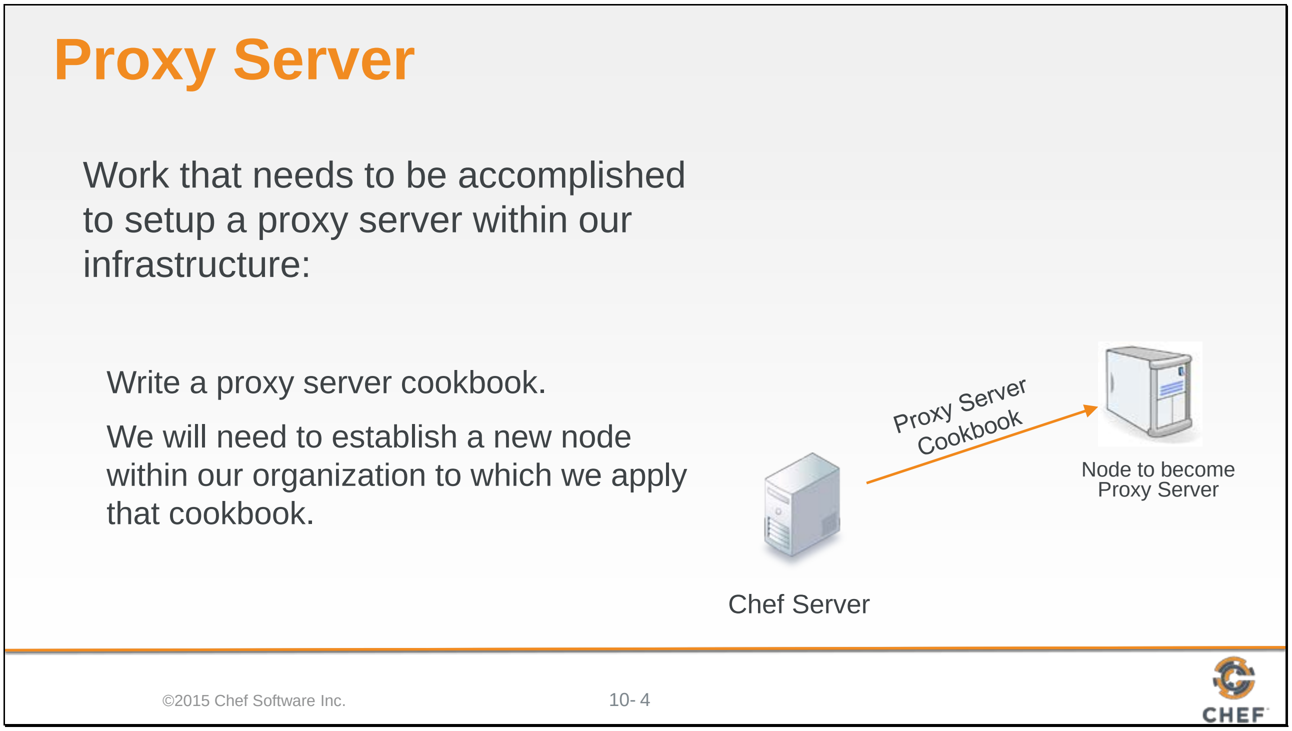


With a single web server running with our organization, it's now time to talk about the next goal to tackle. We need to setup a proxy server.

A proxy server is able to receive requests and relay them to other systems. In our case, we specifically want to use the proxy server to balance the entire traffic load between one or more systems.

This means we will need to establish a new node within our organization, install the necessary software to make the node a proxy server, and configure it so that it will relay requests to our existing node running apache and to future nodes.

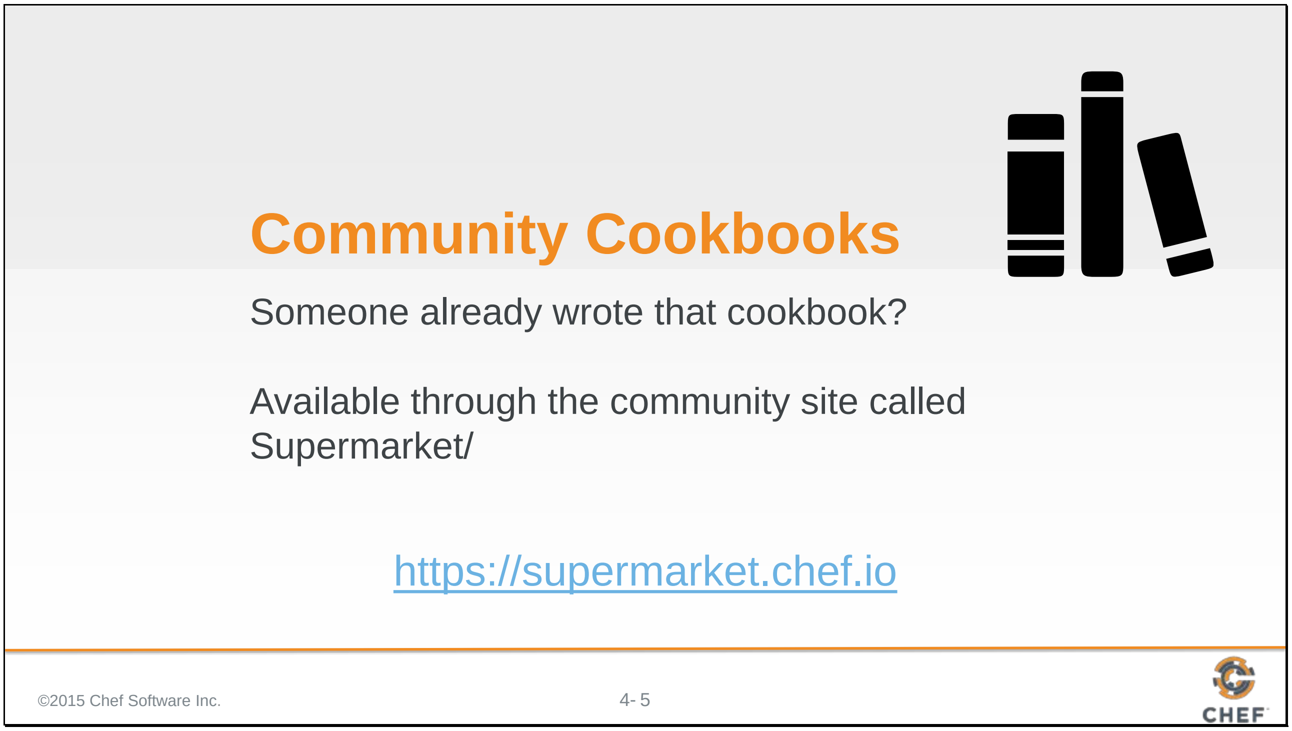
Slide 4



Similar to how we installed and configured apache on our first node, we could do the same thing here with a proxy server. We could learn the package name for the application 'haproxy', learn which file manages the configuration, learn how to compose the configuration with custom values, and then manage the service.

Package, Template and Service are the core of configuration management. Nearly all the recipes you write for an application will center on using these three resources. We could spend some time focused on composing the cookbook recipe and testing it on our platform with our custom configuration.

Slide 5



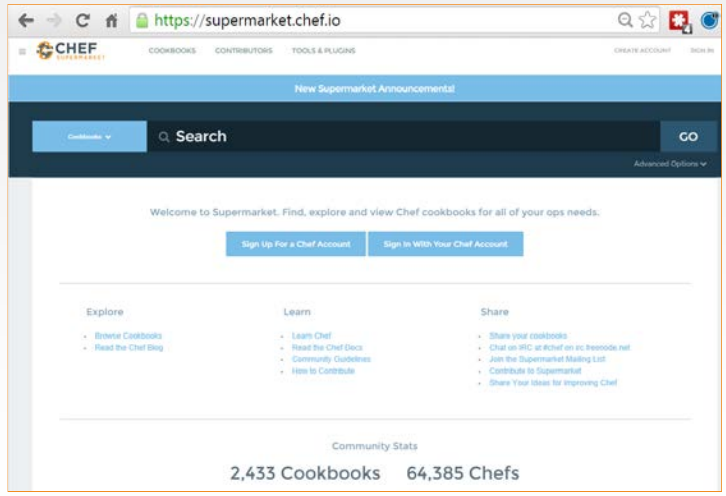
But what if we told you someone already wrote that cookbook?

Someone already has and that cookbook is available through the community site called Supermarket. Supermarket is a public repository of all the cookbooks shared by other developers, teams, and companies who want to share their knowledge and hard work with you to save you time.

Slide 6

Community Cookbooks

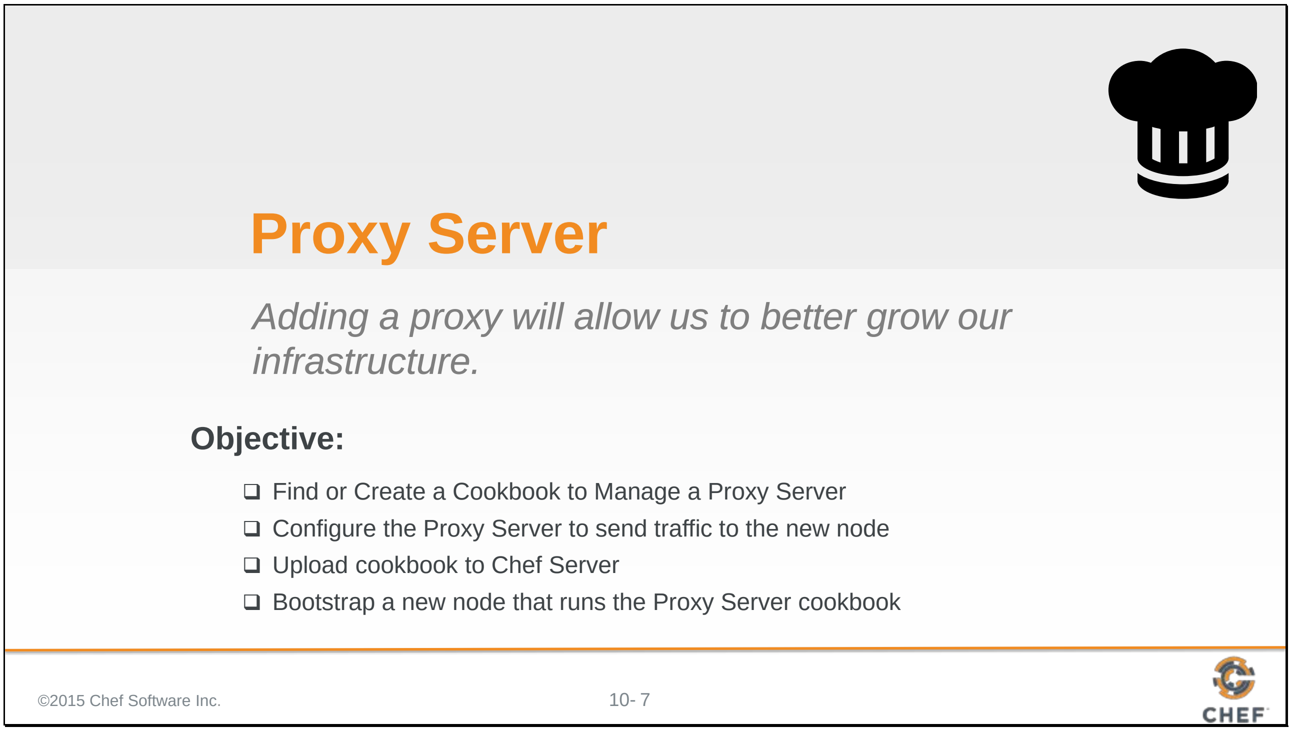
- Community cookbooks are managed by individuals.
- Chef does not verify or approve cookbooks in the Supermarket.
- Cookbooks may not work for various reasons.
- Still, there are real benefits to community cookbooks.



An important thing to remember is that on the community site are cookbooks managed by individuals. Chef does not verify or approve the cookbooks found in the Supermarket. These cookbooks solved problems for the original authors and then they decided to share them. This means that the cookbooks you find in the Supermarket may not be built or designed for your platform. It may not take into special consideration your needs and requirements. It may no longer be actively maintained.

Even if the cookbook does not work as a whole, there is still value in reading and understand the source code and extracting the pieces you need when creating your own. With all that said, there is a real benefit to the community site. When you find a cookbook that helps you deliver value quickly, it can be a tremendous boon to your productivity. This is what we are going to take advantage of with the haproxy cookbook.

Slide 7

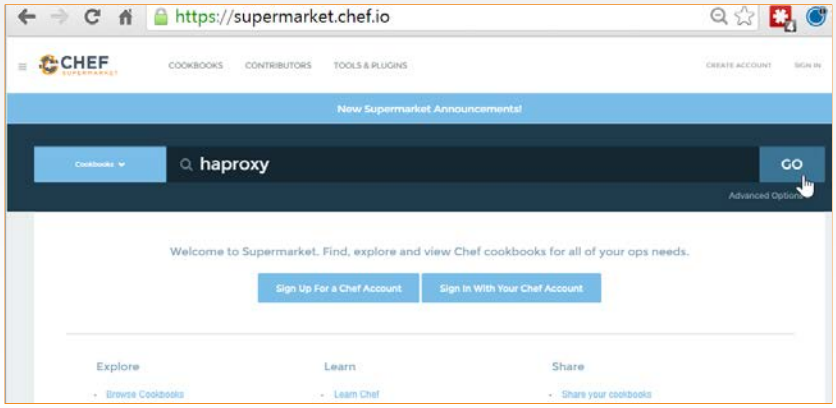


Lets find the haproxy cookbook within the community site to learn more about it.

Slide 8

GE: Searching in the Supermarket

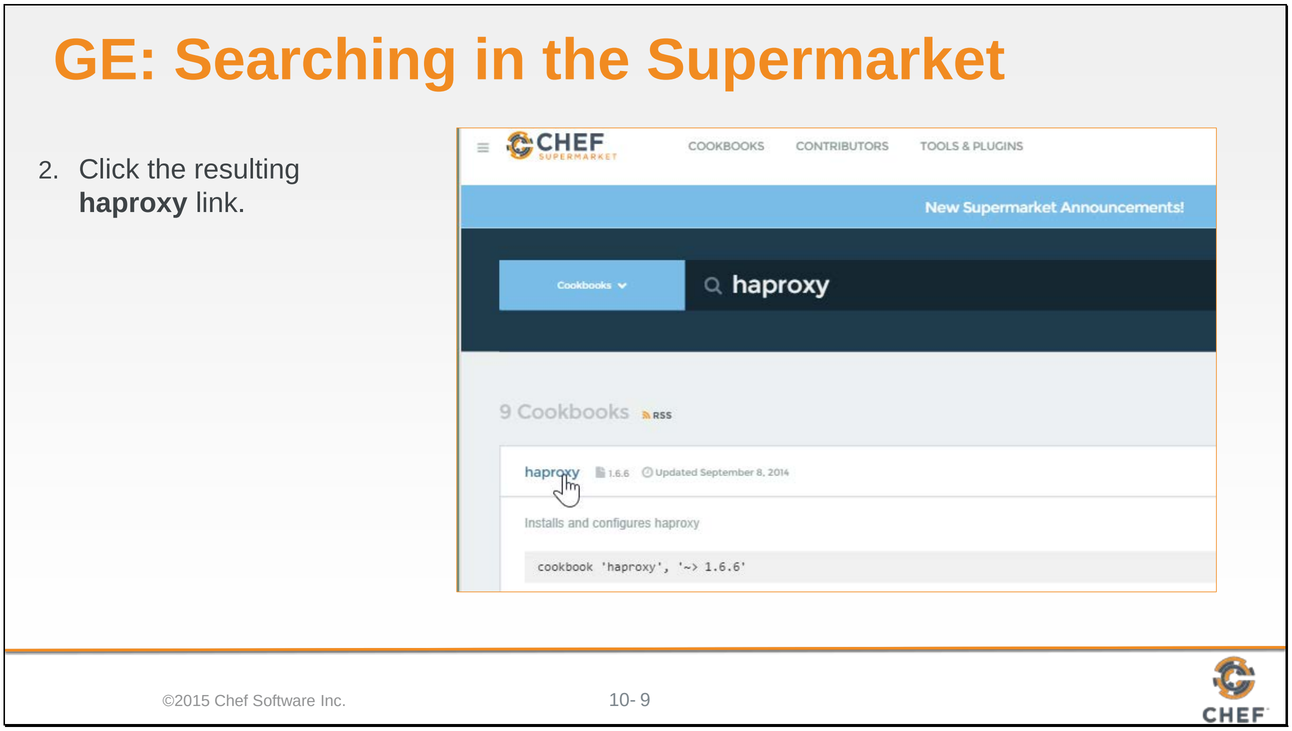
1. From the https://supermarket.chef.io page, type **haproxy** in the search field and then click the **GO** button.



From the Supermarket main page type the search term "haproxy" and the click the **GO** button.

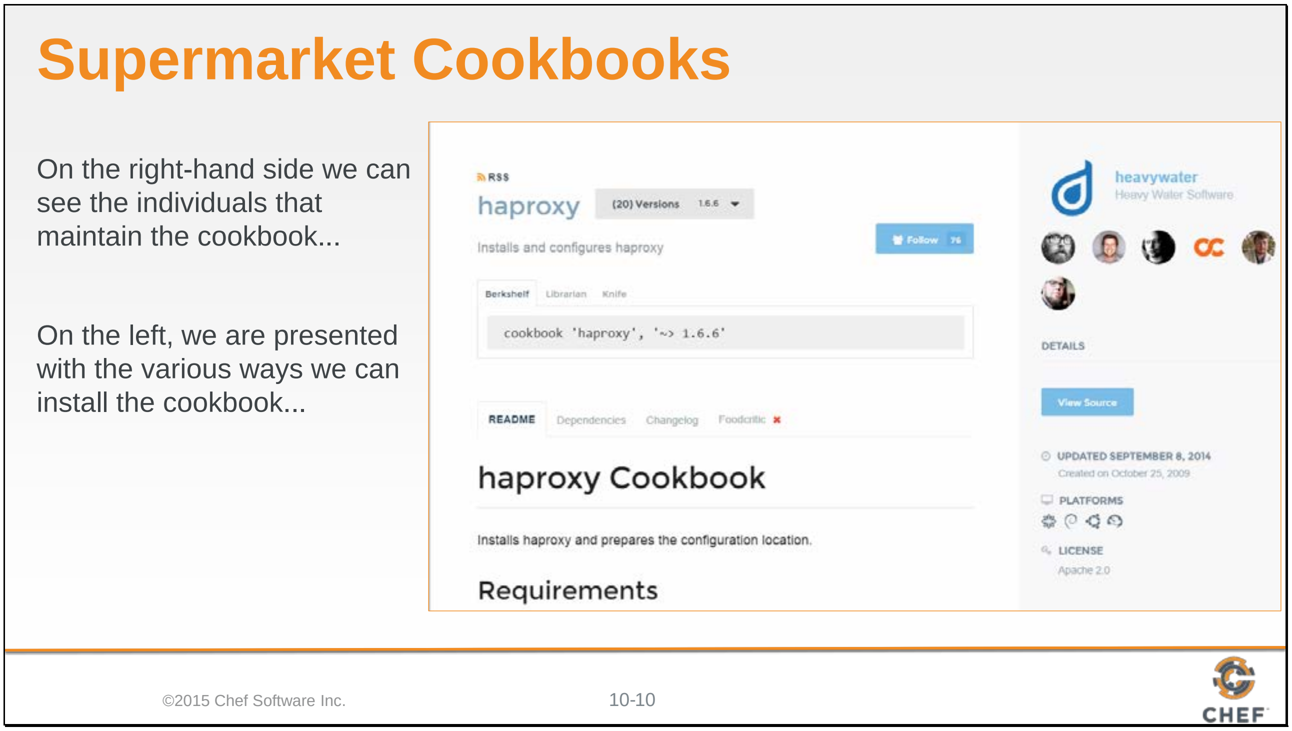
Below the search term will show us all the matching cookbooks. The haproxy cookbook is in that result set.

Slide 9



Cookbooks usually map one-to-one to a piece of software and usually are named after the piece of software that they manage. Select the cookbook named haproxy from the search results.

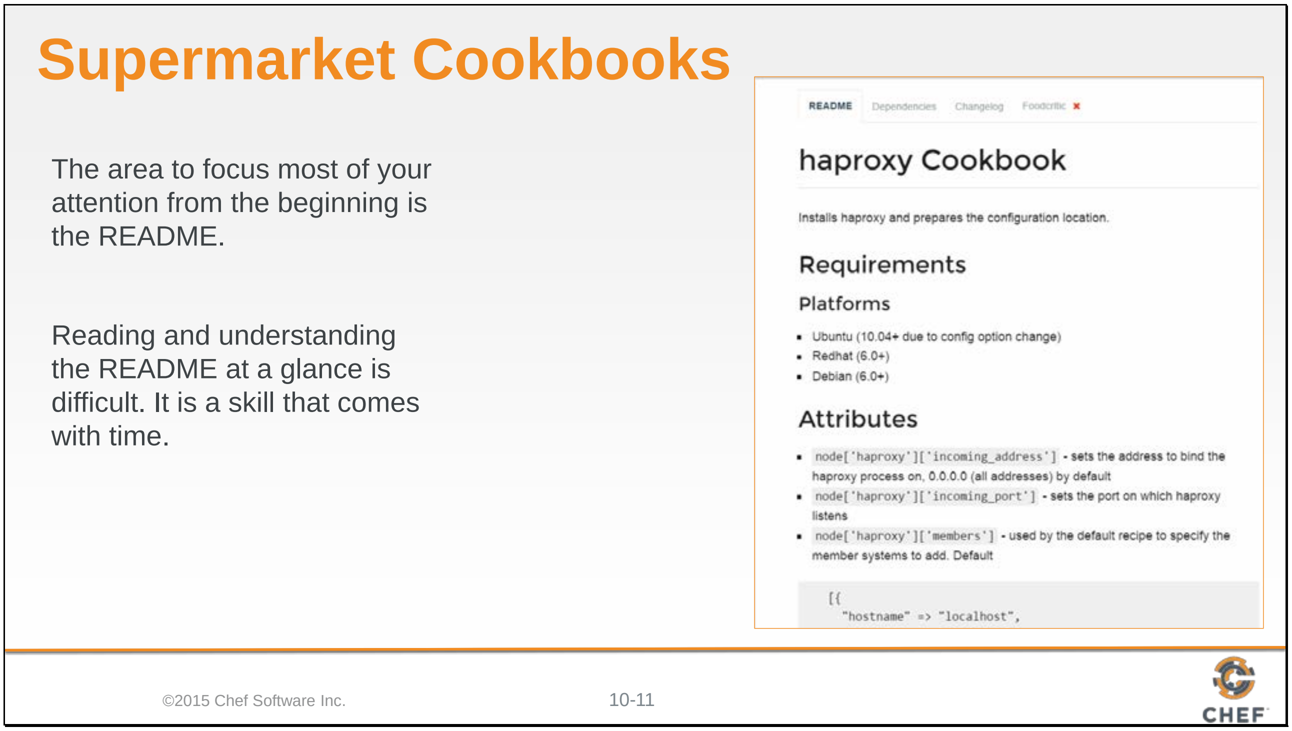
Slide 10



At this point you are presented with information that describes the cookbook. Starting on the right-hand side we see the individuals that maintain the cookbook, a link to view the source details, last updated date, supported platforms, licensing, and a link to download the cookbook.

On the left, we are presented with the various ways we can install the cookbook, the README that describes information about the cookbook, any cookbooks that this cookbook may depend on, a history of the changes, and its food critic rating--which is a code evaluator for best practices.

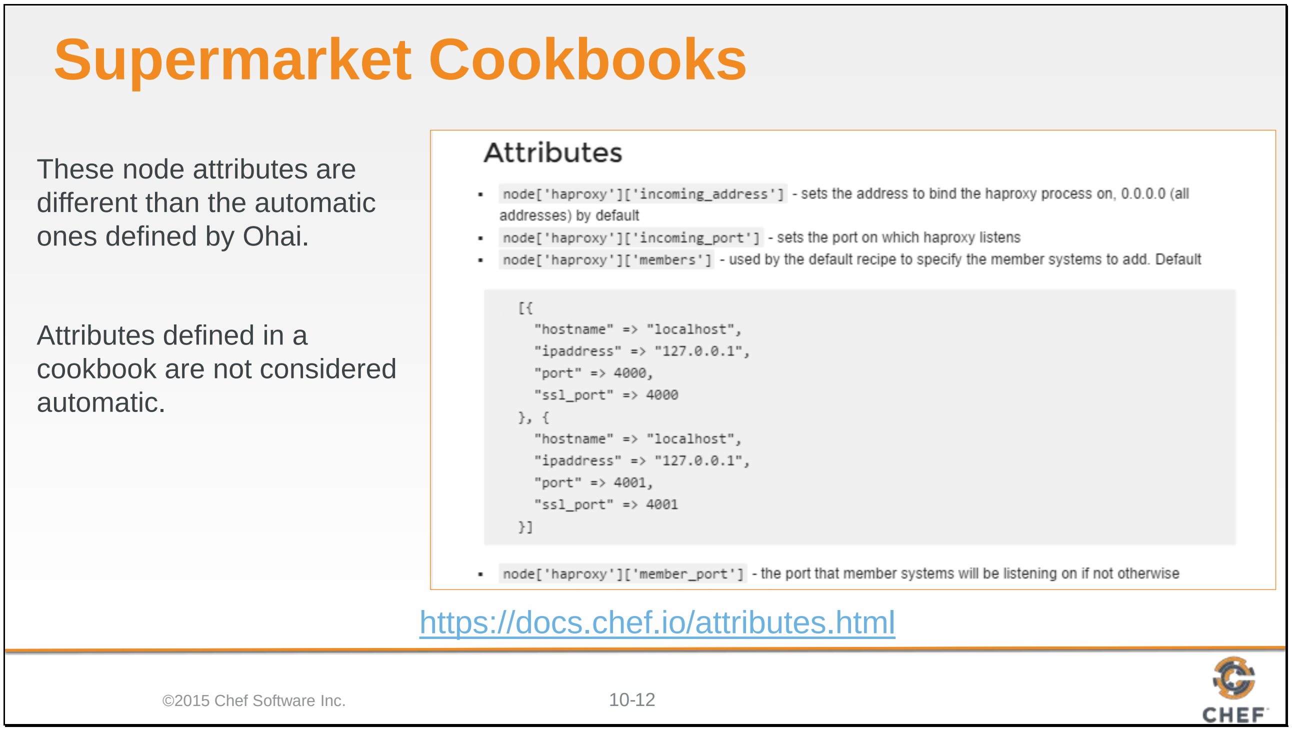
Slide 11



The area to focus most of your attention from the beginning is the README. The README describes the various attributes that are defined within the cookbook and the purpose of the recipe. This is the same README file found in the cookbooks we currently have within our organization. This one, however, has had far more details added to give new users like us the ability to understand more quickly what the cookbook does and how it does it.

Reading and understanding the README at a glance is difficult. It is a skill that comes with time. For the haproxy cookbook there is an defined attribute that establishes the members that receive the proxy requests from the proxy server. This is available in a node attribute available through `node['haproxy']['members']`.

Slide 12



Prior to this point we have seen how node attributes are defined by Ohai but cookbooks also have this ability to define node attributes. These node attributes are different than the ones defined by Ohai as well. Ohai attributes are considered automatic attributes and generally inalienable characteristics about the node.

Attributes defined in a cookbook are not considered automatic. They are simply default values that we may change. There are many ways that we provide new default values for these. One way that we will learn is defining a wrapper cookbook.

Slide 13

A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook but allows us to define new default values for the recipes.

This is a common method for overriding cookbooks because it allows us to leave the original cookbook untouched. We simply provide new default values that we want and then include the recipes that we want to run.

Lets generate our wrapper cookbook named myhaproxy. Traditionally we would name the cookbook with a prefix of the name of our company and then follow it by the cookbook name 'company-cookbook'.

Slide 14

GE: CD and Generate the Cookbook

```
$ cd ~/chef-repo
$ chef generate cookbook cookbooks/myhaproxy
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[C:/Users/sdelfante/chef-repo/cookbooks/myhaproxy] action create
    - create new directory C:/Users/sdelfante/chef-repo/cookbooks/myhaproxy
  * template[C:/Users/sdelfante/chef-repo/cookbooks/myhaproxy/metadata.rb] action
create_if_missing
    - create new file C:/Users/sdelfante/chef-repo/cookbooks/myhaproxy/metadata.rb
    - update content in file C:/Users/sdelfante/chef-
repo/cookbooks/myhaproxy/metadata.rb from none to 899276
    (diff output suppressed by config)
  * template[C:/Users/sdelfante/chef-repo/cookbooks/myhaproxy/README.md] action
create_if_missing
```


Change to your chef-repo directory and then generate your new cookbook.

Slide 15

GE: Create a Dependency in the Cookbook

~/chef-repo/cookbooks/myhaproxy/metadata.rb

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

depends 'haproxy', '~> 1.6.6'                +
```

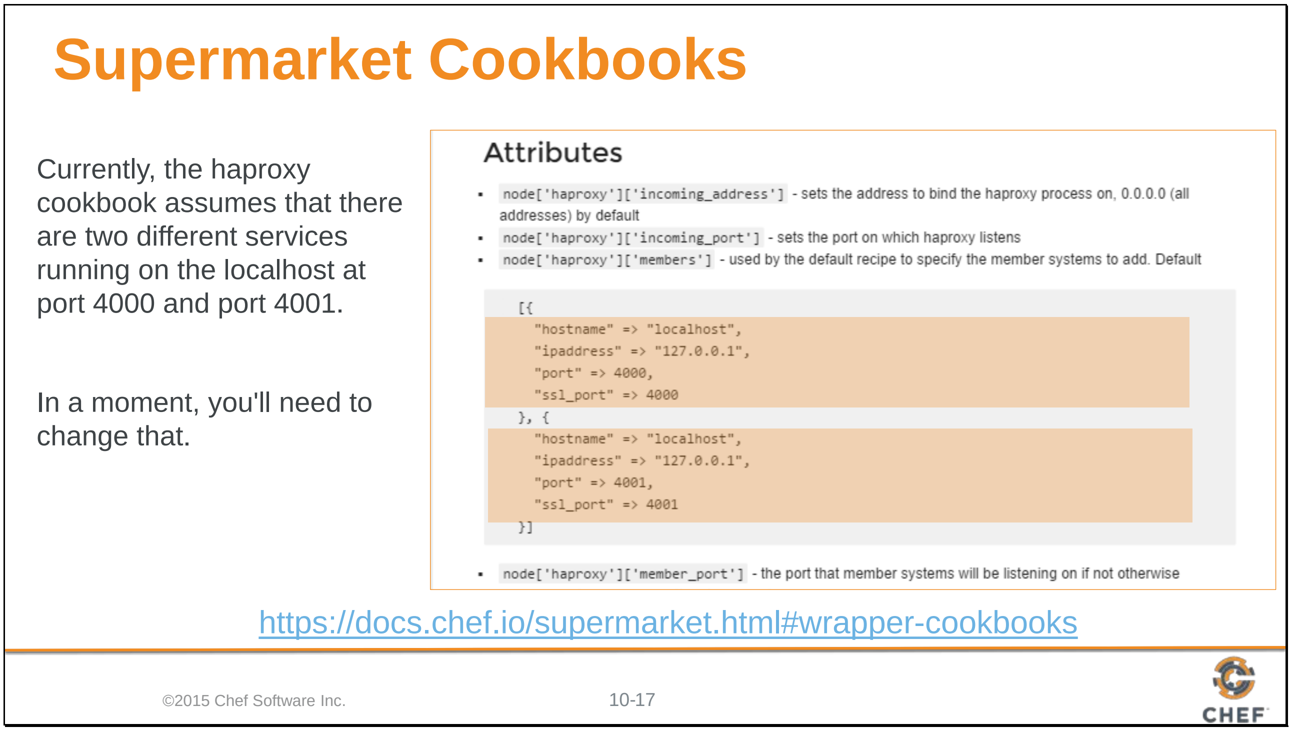

Set up a dependency within your haproxy cookbook. Establishing this dependency informs the Chef Server that whenever you deliver this cookbook to a node, you should also deliver with it the mentioned dependent cookbooks.

This is important because your cookbook is simply going to set up new default values and then execute the recipes defined in the original cookbook.

Slide 16

Now that you have the dependency on the haproxy cookbook in your wrapper cookbook, you need to learn what new default values you need to add to the recipe.

Slide 17



Currently the haproxy cookbook assumes that there are two different services running on the localhost at port 4000 and port 4001. The haproxy process will relay messages to itself to those two ports.

That is not our configuration. First, we currently only have one system that we want to route traffic. Second, we want to have the traffic routed not to localhost but instead to our webserver, node1, which will have a completely different hostname and IP address.

Slide 18

GE: Capture Node's Public Host Name and IP

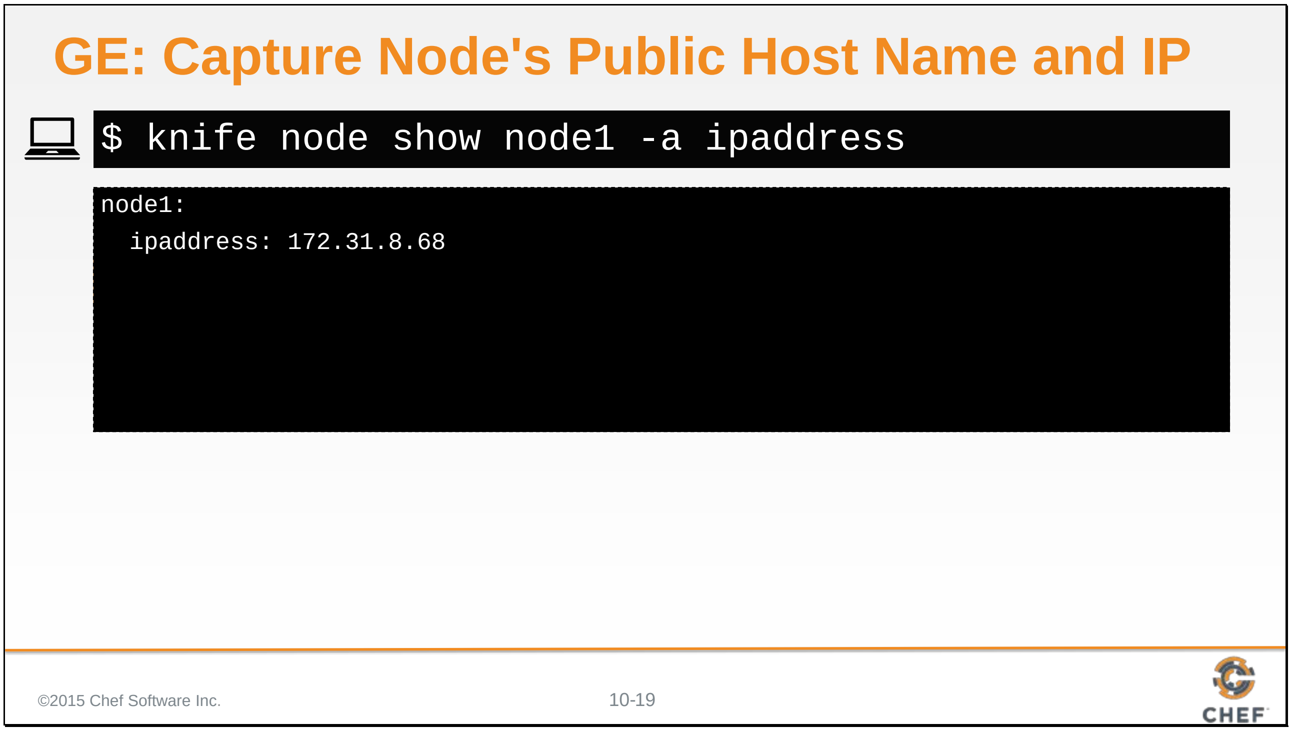
```
$ knife node show --help
```

```
knife node show NODE (options)
    -a ATTR1 [--attribute ATTR2] ,   Show one or more attributes
        --attribute
    -s, --server-url URL             Chef Server URL
        --chef-zero-host HOST        Host to start chef-zero on
        --chef-zero-port PORT        Port (or port range) to start chef-zero on.
Port ranges
    -k, --key KEY                    API Client Key
        --[no-]color                 Use colored output, defaults to false on
Windows, true
    -c, --config CONFIG              The configuration file to use
        --defaults                   Accept default values for all questions
    -d, --disable-editing            Do not open EDITOR, just accept the data as is
    -e, --editor EDITOR              Set the editor to use for interactive commands
```


This new default value for the haproxy members needs to define the information about the webserver node, node1. So you need to capture the node's public host name and public IP address.

The 'knife node show' command will display information about the node. You can ask to see a specific attribute on a node with the –a flag or the --attribute flag.

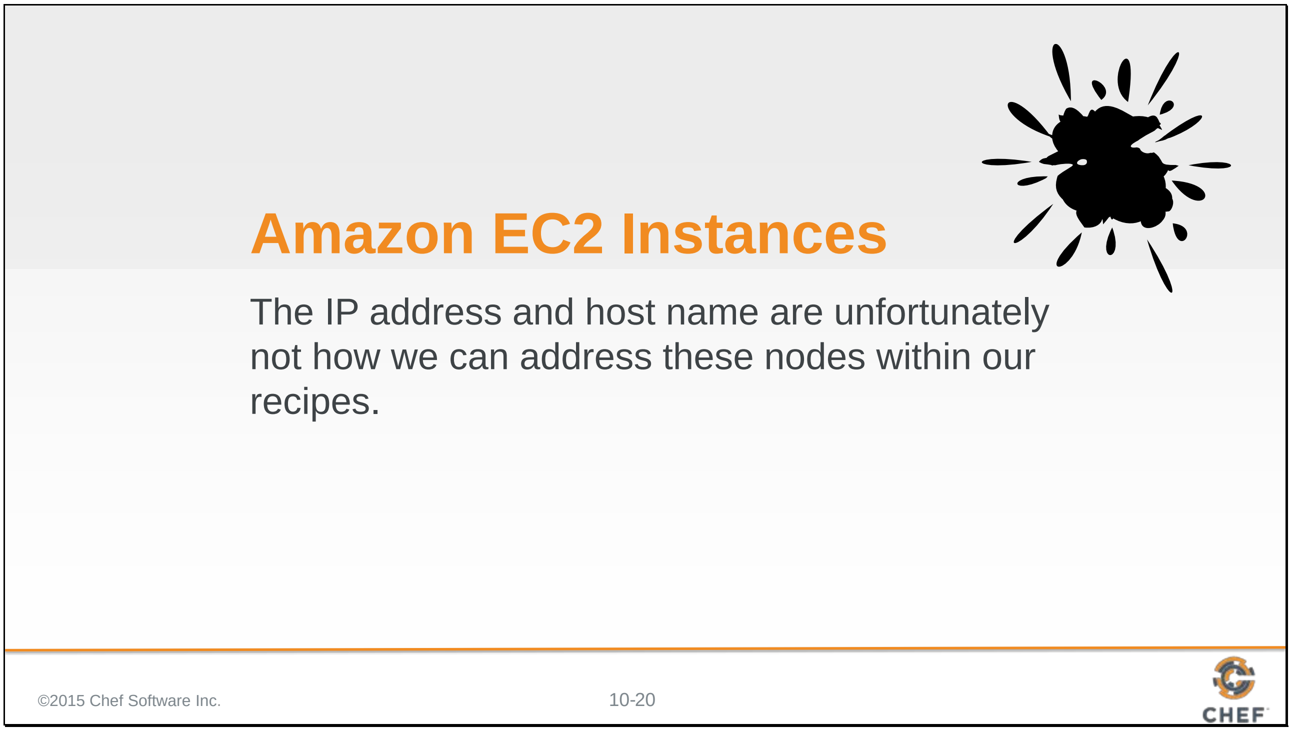
Slide 19



You can display the IP address of node1 with the '-a' flag and specifying the attribute 'ipaddress'.

With cloud providers that generate machines for you often assign internal IP addresses, those values may not work properly.

Slide 20



The reason you may need to ask the node for a different set of attributes is that we are using Amazon as a cloud provider for our instances. These instances are displaying the internal IP address when we ask for the ipaddress attribute.

Ohai collects attributes from the current cloud provider and makes them available in an attribute named 'cloud'. We can look at the cloud attribute on our first node and see that it returns for us information about the node.

Slide 21

GE: Capture Node's Public Host Name and IP

```
$ knife node show node1 -a cloud
```

```
node1:
  cloud:
    local_hostname:  ip-172-31-8-68.ec2.internal
    local_ipv4:      172.31.8.68
    private_ips:     172.31.8.68
    provider:        ec2
    public_hostname: ec2-54-175-46-24.compute-1.amazonaws.com
    public_ips:      54.175.46.24
    public_ipv4:     54.175.46.24
```


CHEF

If you use 'knife node show' to display the 'cloud' attribute for node1, you will see the local, private, and public connection information.

Capture and write down the public hostname and the public ipv4 address of node1. You will need this in the recipe you are going to write.

Slide 22

GE: Edit the myhaproxy/recipes/default.rb

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
#
# Cookbook Name:: myhaproxy
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights
Reserved.


include_recipe "haproxy::default"
```


First, within the myhaproxy cookbook you will use the include_recipe method to specify the fully-qualified name of the cookbook and recipe that you want to execute. In this case, when you run your wrapped cookbooks recipe, you'll want it to run the original cookbook's default recipe.

Slide 23

GE: Edit the myhaproxy/recipes/default.rb

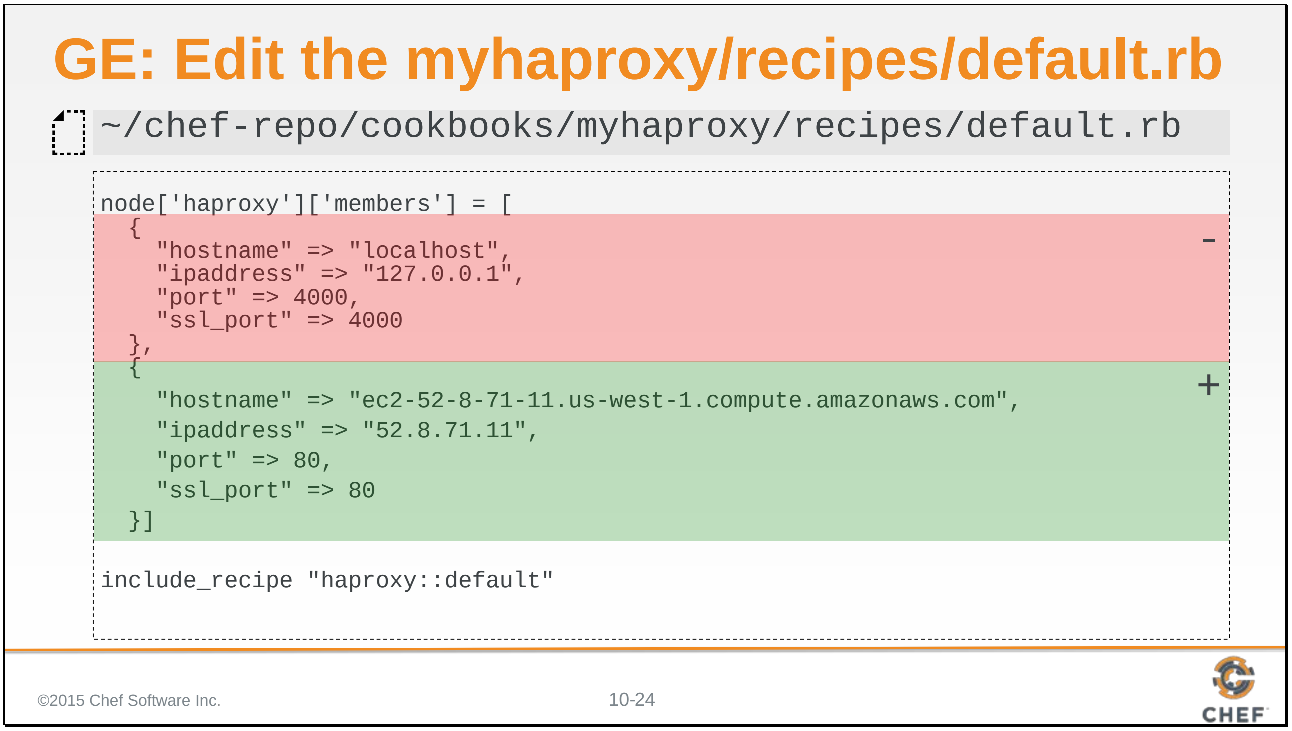
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
#

node['haproxy']['members'] = [
{
    "hostname" => "localhost",
    "ipaddress" => "127.0.0.1",
    "port" => 4000,
    "ssl_port" => 4000
  }, {
    "hostname" => "localhost",
    "ipaddress" => "127.0.0.1",
    "port" => 4001,
    "ssl_port" => 4001
}]

inlude_recipe "haproxy::default"
```


Without changing anything any further, using this cookbook will simply execute the original cookbooks' recipe with all the same default values. Before you execute that recipe, you'll need to override the default values with your own.

Copy and paste the original default values into your recipe, as shown here.

Slide 24



Remove one of the entries within the members array (shown in red).

Then update the information for the remaining member to include the public ipaddress and hostname for node1 (shown in green).

Slide 25

GE: Edit the myhaproxy/recipes/default.rb

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node.default['haproxy']['members'] = [{
    "hostname" => "ec2-52-8-71-11.us-west-1.compute.amazonaws.com",
    "ipaddress" => "52.8.71.11",
    "port" => 80,
    "ssl_port" => 80
  }]

include_recipe "haproxy::default"
```


To replace a default attribute in a recipe you have to use 'node.**default**['haproxy']['members']...'

So you need to change:

'node['haproxy']['members']'
to
'node.**default**['haproxy']['members']'

Slide 26

GE: Edit the myhaproxy/recipes/default.rb

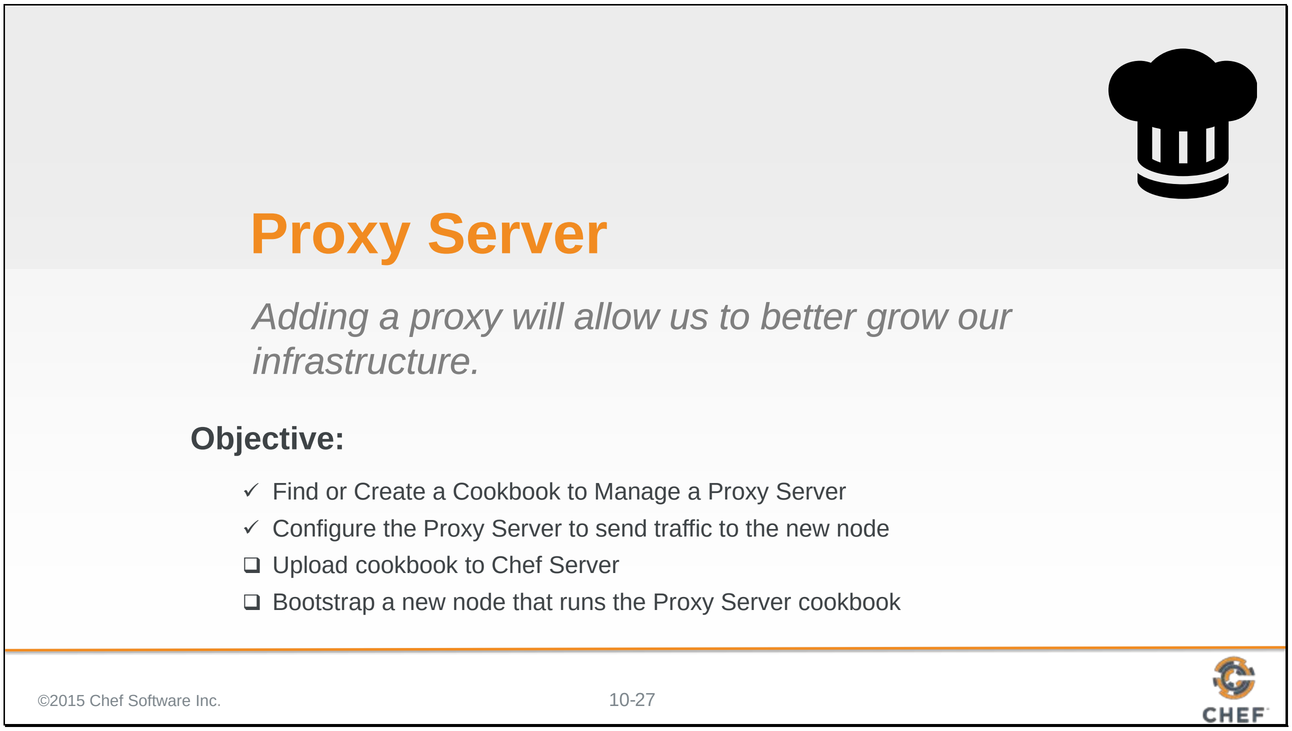
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node.default['haproxy']['members'] = [{
    "hostname" => "ec2-52-8-71-11.us-west-1.compute.amazonaws.com",
    "ipaddress" => "52.8.71.11",
    "port" => 80,
    "ssl_port" => 80
  }]

include_recipe "haproxy::default"
```


The final default recipe for the wrapper cookbook 'myhaproxy' looks like the above.

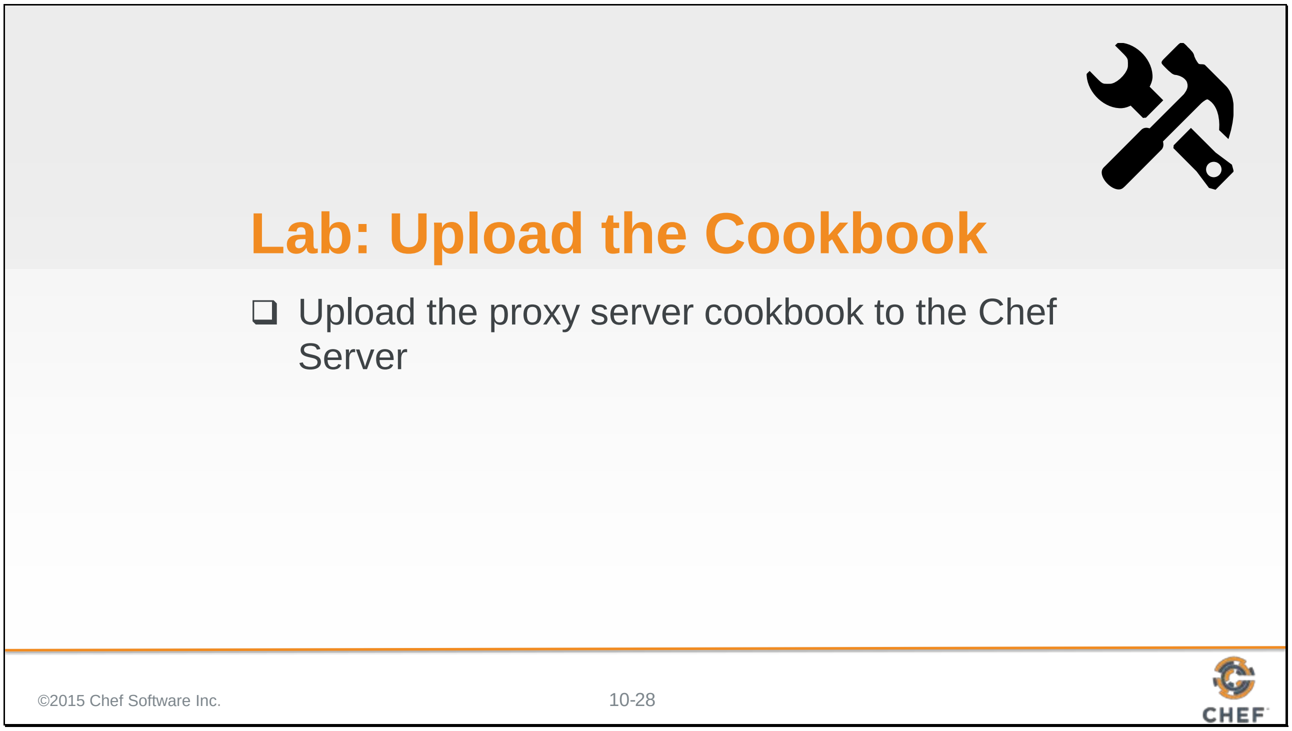
Save your recipe file.

Slide 27



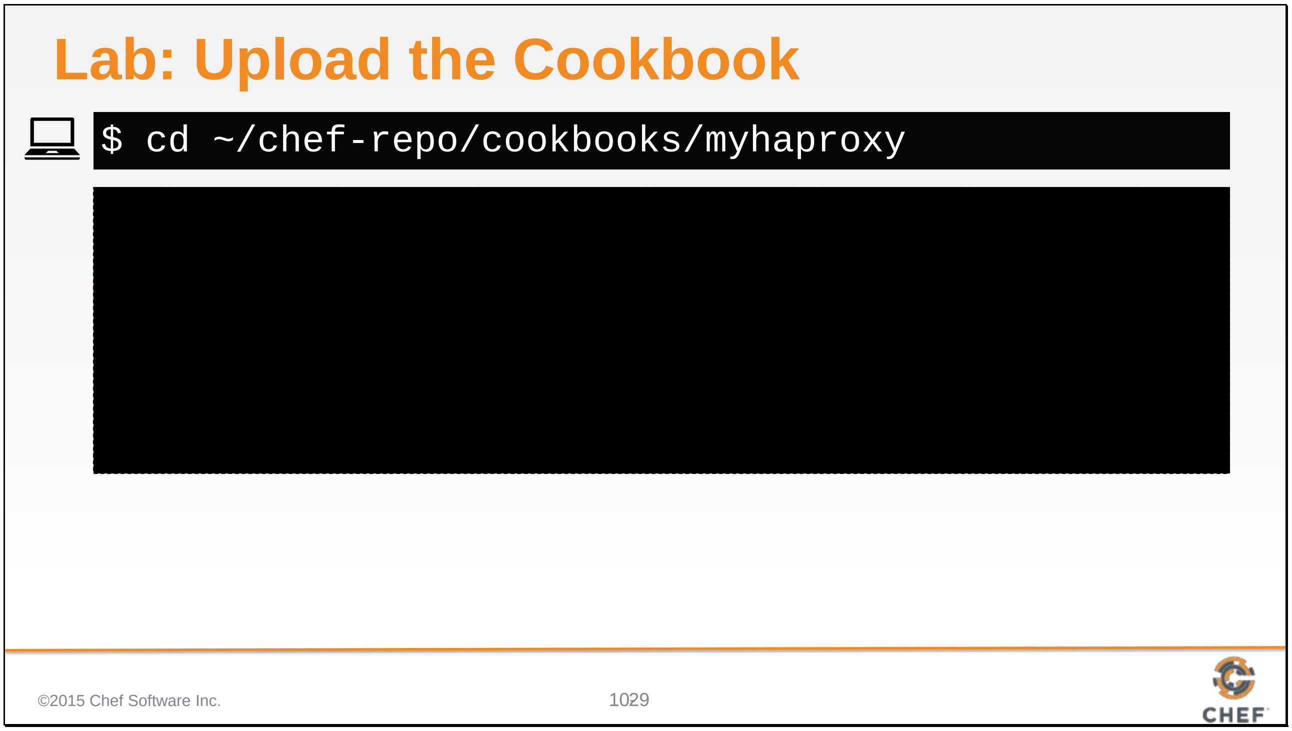
You have completed creating the wrapper cookbook. It is time to upload to the Chef Server.

Slide 28



As a lab exercise, upload the cookbook to the Chef Server

Slide 29



Lets review that lab.

You change into the directory for the 'myhaproxy' cookbook.

Slide 30

Lab: Upload the Cookbook

```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'myhaproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using build-essential (2.2.3)
Using cpu (0.2.0)
Using haproxy (1.6.6)
Using myhaproxy (0.1.0) from source at .
```


We use the Berkshelf to upload our cookbooks. This is where Berkshelf really shines as a tool.

Run the command "berks install".

When you run this command for a cookbook that has a dependency, you'll see that Berkshelf will download the haproxy cookbook and its dependencies as well. The haproxy cookbook is dependent on the build-essential cookbook and the cpu cookbook. If any of those cookbooks had dependencies, berkshelf would find those and download them as well.

Slide 31

Lab: Upload the Cookbook

```
$ berks upload
```

```
Uploaded build-essential (2.2.3) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded cpu (0.2.0) to: 'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded haproxy (1.6.6) to: 'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded myhaproxy (0.1.0) to: 'https://api.opscode.com:443/organizations/steveessentials2'
```


After installing all the necessary dependent cookbooks, we used 'berks upload' to send the cookbook and all its dependencies to the Chef Server. This is again an easier method to manage dependencies instead of manually identifying the dependencies and then uploading each single cookbook at a time.

Slide 32

Lab: Verify the Cookbook Upload

```
$ knife cookbook list
```

```
apache              0.2.1
build-essential     2.2.3
cpu                 0.2.0
haproxy             1.6.6
myhaproxy           0.1.0
workstation         0.2.1
```


When that is complete you can verify that you've uploaded your cookbook and all of its dependencies.

Slide 33

The myhaproxy cookbook's default recipe is ready to be assigned to a run list of a node. So we'll need another node. The new proxy node.

Slide 34

Lab: Bootstrap a Proxy Server

- ❑ Bootstrap a new node
- ❑ Update the run list of the new node to include the wrapper proxy server cookbook
- ❑ SSH to that system and run chef-client
- ❑ Verify that traffic to the proxy server is relayed to the web server.

Bootstrap this node the same as you did before but this time define the run list to converge the myhaproxy's default recipe.

After setting that value, SSH into that node with the provided user name and password.

Then run 'sudo chef-client' to apply the recipes defined in this node's run list.

Then verify that your new node's default website is properly redirecting traffic to the original web node you previously set up.

Slide 35

Lab: Bootstrap a New Node

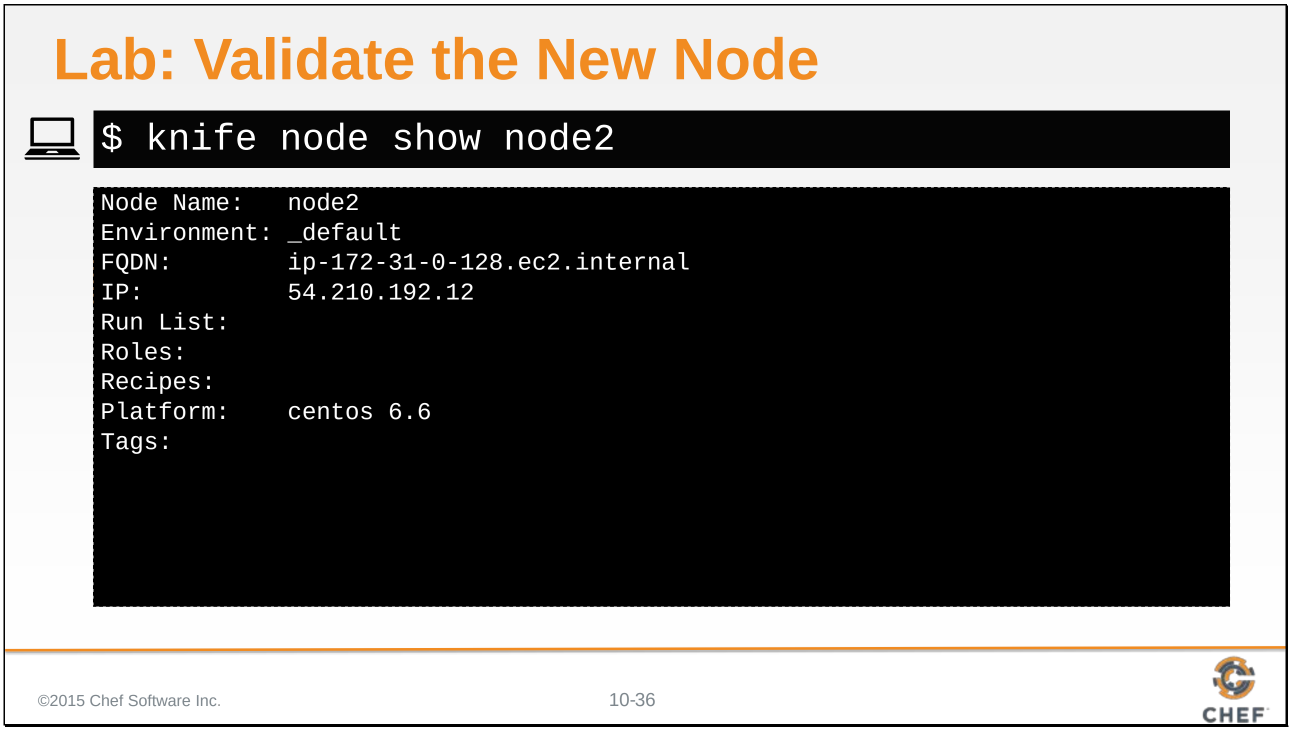
```
$ knife bootstrap FQDN2 -x USER -P PWD --sudo -N node2
```

```
Creating new client for node2
Creating new node for node2
Connecting to ec2-54-210-192-12.compute-1.amazonaws.com
ec2-54-210-192-12.compute-1.amazonaws.com Starting first Chef Client run...
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list: []
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-192-12.compute-1.amazonaws.com [2015-09-16T17:13:10+00:00] WARN:
Node node2 has an empty run list.
ec2-54-210-192-12.compute-1.amazonaws.com Converging 0 resources
ec2-54-210-192-12.compute-1.amazonaws.com
ec2-54-210-192-12.compute-1.amazonaws.com Running handlers:
```


CHEF

First you bootstrap a new node named node2.

Slide 36



After the node is bootstrapped, validate that it was added correctly to the organization.

Slide 37

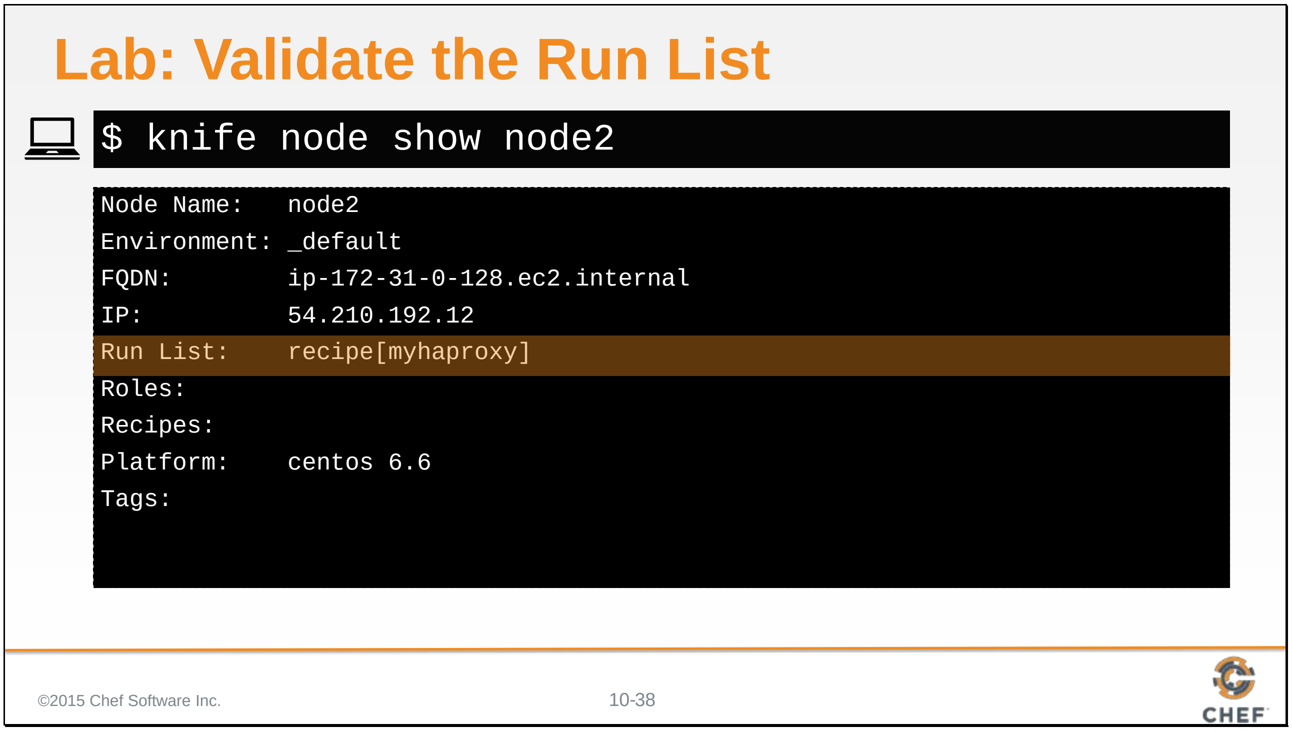
Lab: Define the Run List

```
$ knife node run_list add node2 "recipe[myhaproxy]"
```

```
node2:
  run_list: recipe[myhaproxy]
```

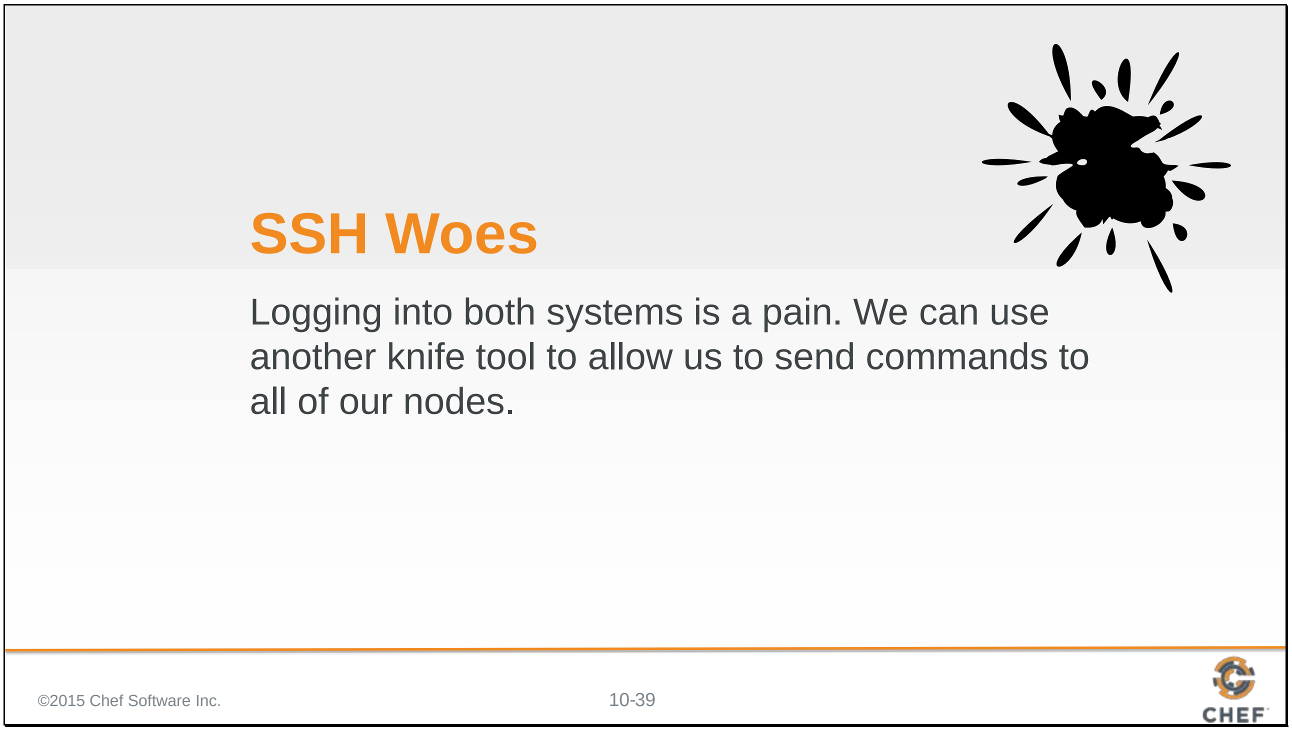

Define an initial run list for that node to converge the default recipe of the myhaproxy cookbook.

Slide 38



Ensure the run list has been set correctly for node2.

Slide 39



We asked you to login to that remote node and run 'sudo chef-client' to apply the new run list defined for that node. This does in fact work but considering that we may need to execute this command for this node and many future nodes, it seems like a lot of windows and commands that we would need to execute.

Slide 40

GE: Using knife ssh

```
$ knife ssh --help
```

```
knife ssh QUERY COMMAND (options)
    -a, --attribute ATTR             The attribute to use for opening the connection
- default depends on the context
    -s, --server-url URL             Chef Server URL
        --chef-zero-host HOST        Host to start chef-zero on
        --chef-zero-port PORT        Port (or port range) to start chef-zero on.
Port ranges like 1000,1010 or 8889-9999 will try all given ports until one works.
    -k, --key KEY                    API Client Key
        --[no-]color                 Use colored output, defaults to false on
Windows, true otherwise
    -C, --concurrency NUM            The number of concurrent connections
    -c, --config CONFIG              The configuration file to use
        --defaults                   Accept default values for all questions
```


CHEF

To make our lives easier, the 'knife' command provides a subcommand named 'ssh' that allows us to execute a command across multiple nodes that match a specified search query.

Slide 41

GE: Define the Run List

```
$ knife ssh "*:*" -x USERNAME -P PASSWORD "sudo chef-client"
```

```
ec2-54-175-46-24.compute-1.amazonaws.com  Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-175-46-24.compute-1.amazonaws.com  resolving cookbooks for run list:
["apache"]
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-54-175-46-24.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com    - apache
ec2-54-175-46-24.compute-1.amazonaws.com  Compiling Cookbooks...
ec2-54-175-46-24.compute-1.amazonaws.com  Converging 3 resources
ec2-54-175-46-24.compute-1.amazonaws.com  Recipe: apache::server
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com   - build-essential
ec2-54-210-192-12.compute-1.amazonaws.com   - cpu
```

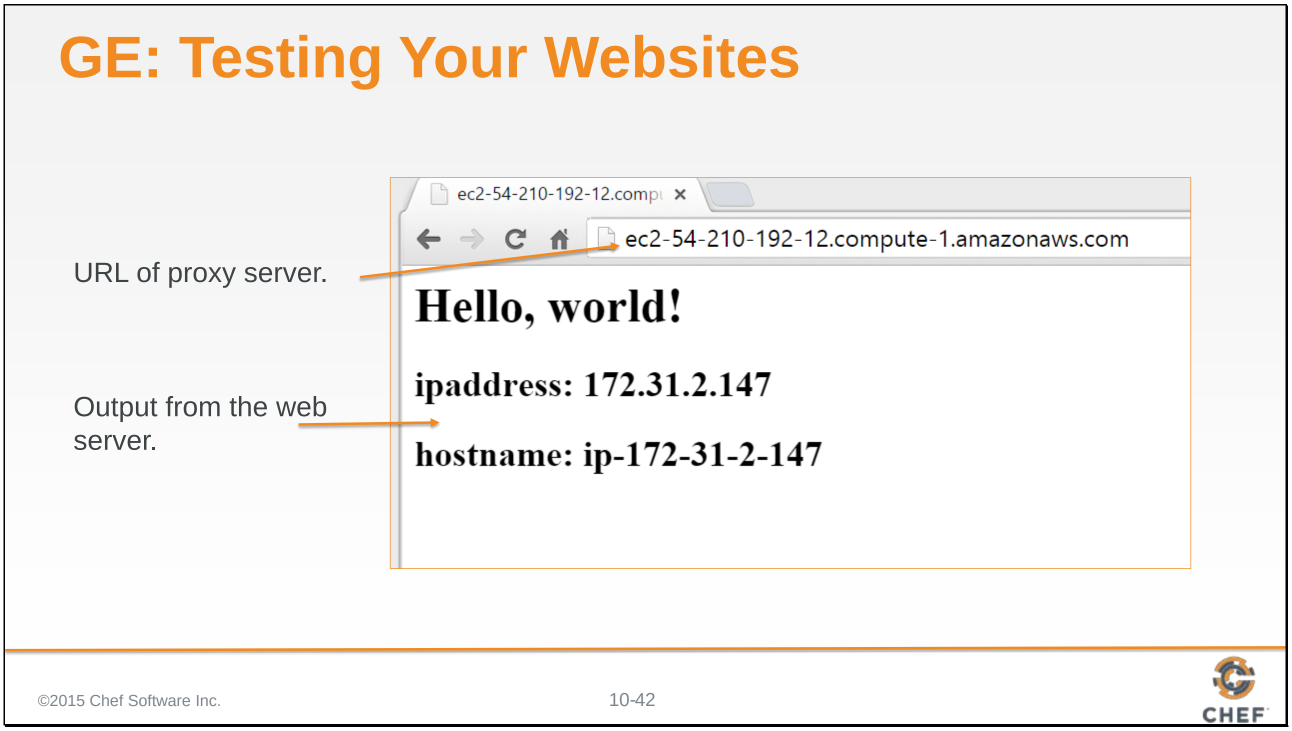

There are a lot of options for defining the search criteria that we will continue to explore. The most important criteria in this instance is star-colon-star. This means that we want to issue a command to all nodes.

So if you want to execute a "sudo chef-client" run for all of your nodes, you should write out this command.

You would need to provide the user name to log into the system, the password for that system, and then finally the command to execute.

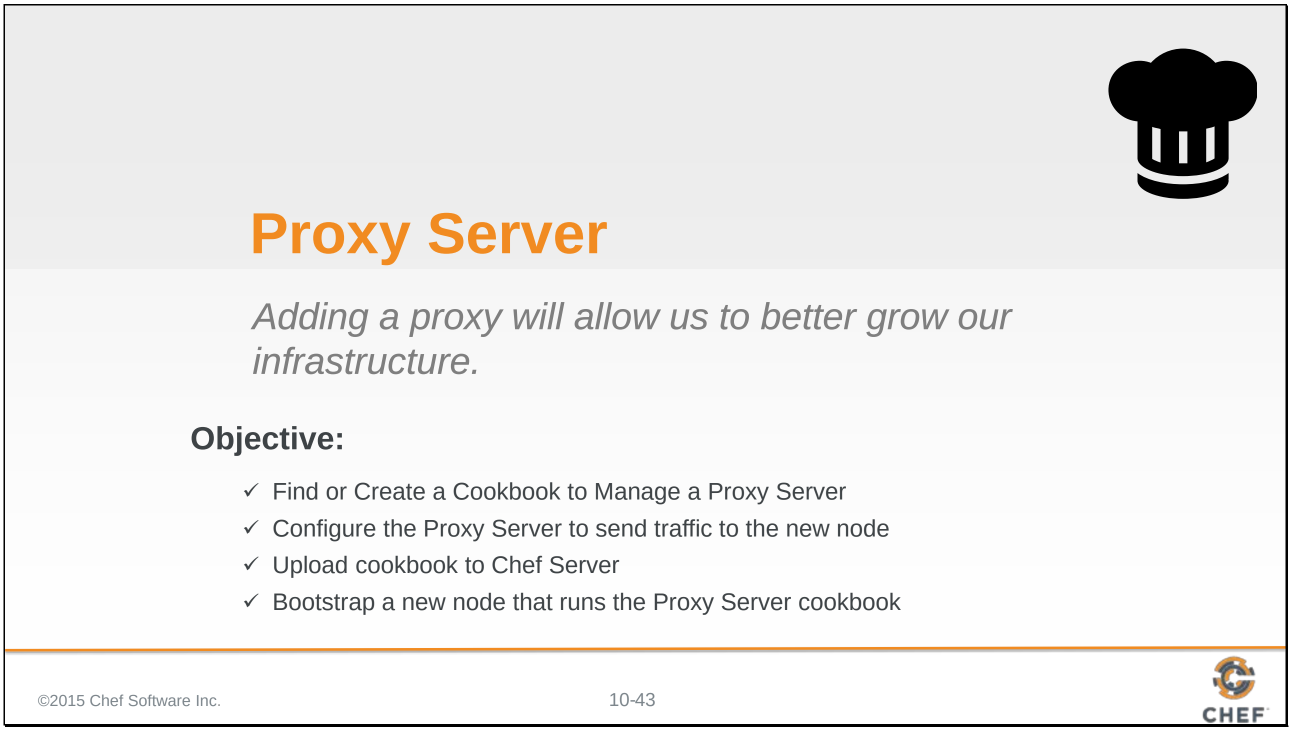
In this way, you could easily ask your nodes to update from your current workstation as long as they all have the same login credentials. For more security, you should likely use SSH keys and forego specifying a username and password.

Slide 42



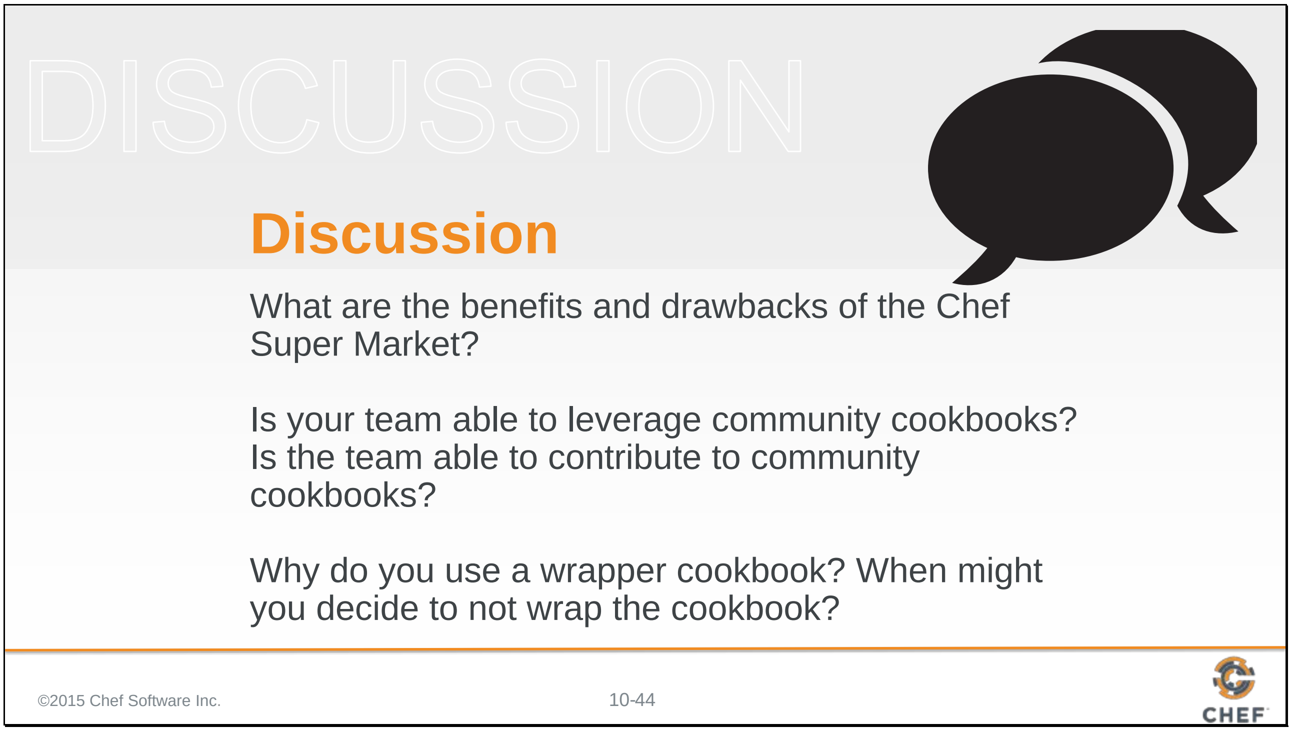
Point a web browser to the URL or public IP address of your proxy server. It should display the web page of the web server node that the proxy server is configured to serve.

Slide 43



With your node running the myhaproxy's cookbook's default recipe--relaying traffic to your first node running the apache cookbook's default recipe--you have moved closer to creating the original topology we set out to define today.

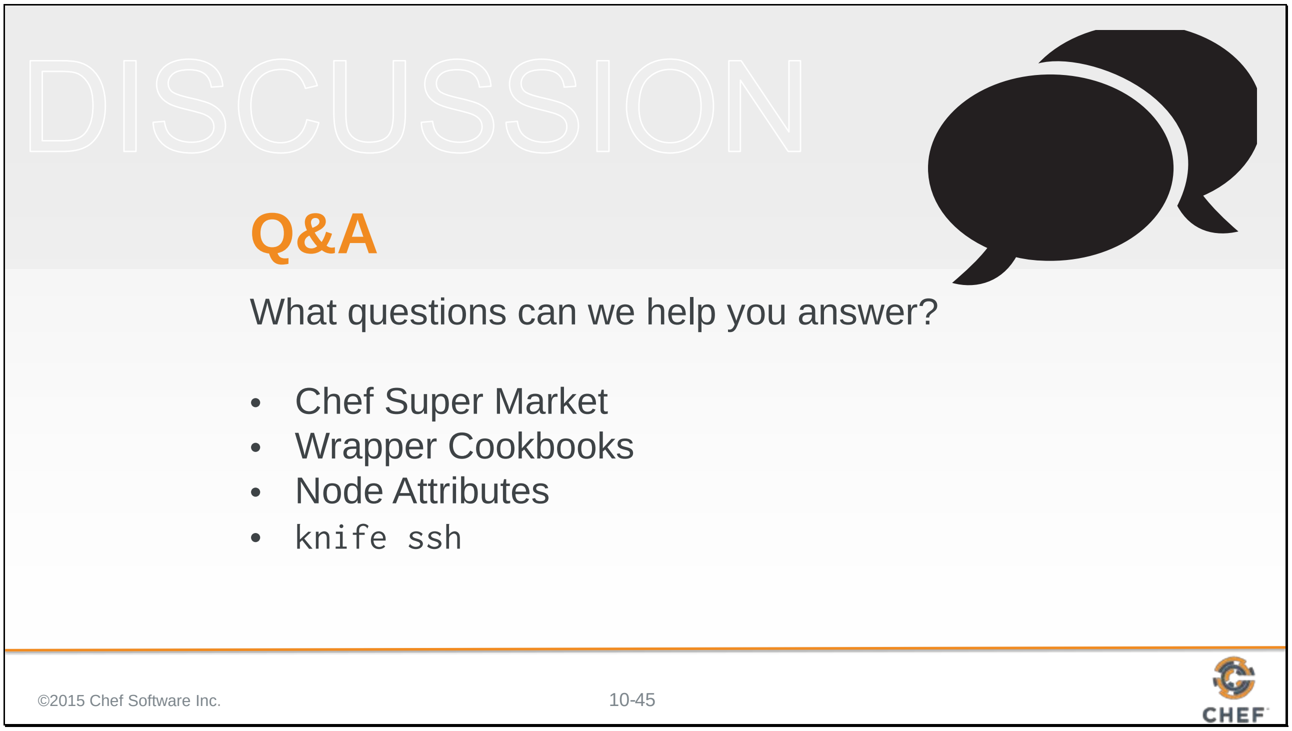
Slide 44



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 45



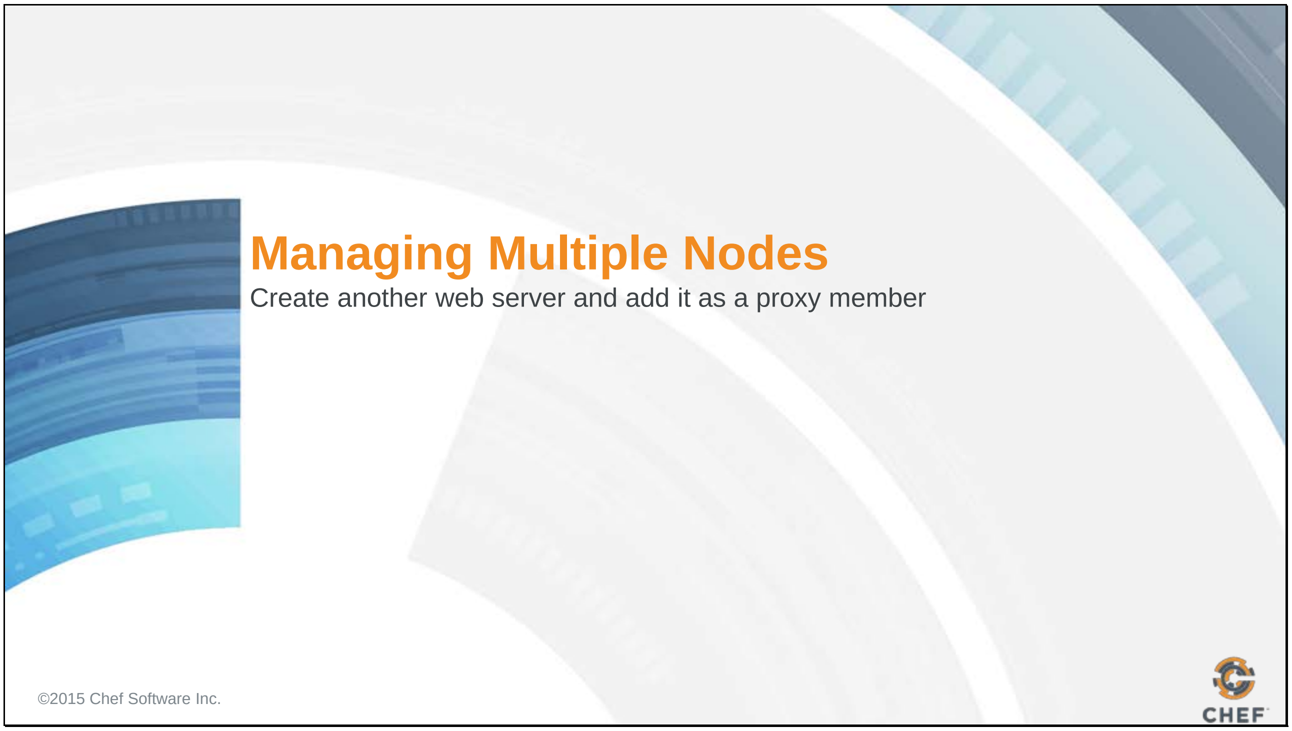
What questions can we help you answer?

In general or about specifically about Chef Super Market, wrapper cookbooks, node attributes, the 'knife ssh' command.

Slide 46

CHEF™

11: Managing Multiple Nodes



This section's goal is to have you bootstrap another node, this time a web server, and add it to the proxy members.

Slide 2

Objectives

After completing this module, you should be able to

- Bootstrap, update the run_list, and run chef-client on a node
- Append values to an attribute within a recipe
- Version a cookbook and upload it to the Chef Server

CHEF

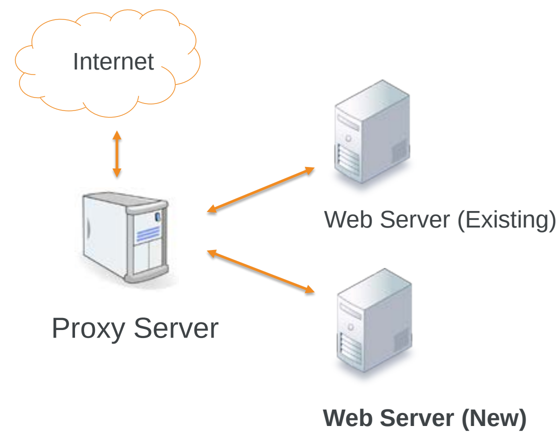
In this module you will learn how to bootstrap, update the run list, and run chef-client on a node. You will also learn how to update a default attribute within a recipe, version and upload a cookbook.

Slide 3

Managing User Traffic

You already configured the proxy server and one web server node.

In this module you'll add another node to the proxy server's list a web server's it is serving.



After completing this module, you will have configured three nodes:

- Node 1: A web server
- Node 2: The proxy server
- Node 3: Another web server

Slide 4

Lab: Another Web Node

- ❑ Bootstrap a new node
- ❑ Update the run list of the new node to include the web server cookbook
- ❑ Login to that system and run chef-client
- ❑ Verify that the node's web server is functional

CHEF

Now its time to create a third node. The third node will be the second web server node.

We will provide you with a new node for the following exercise.

Slide 5

Bootstrap the new node and name it node3.

Slide 6

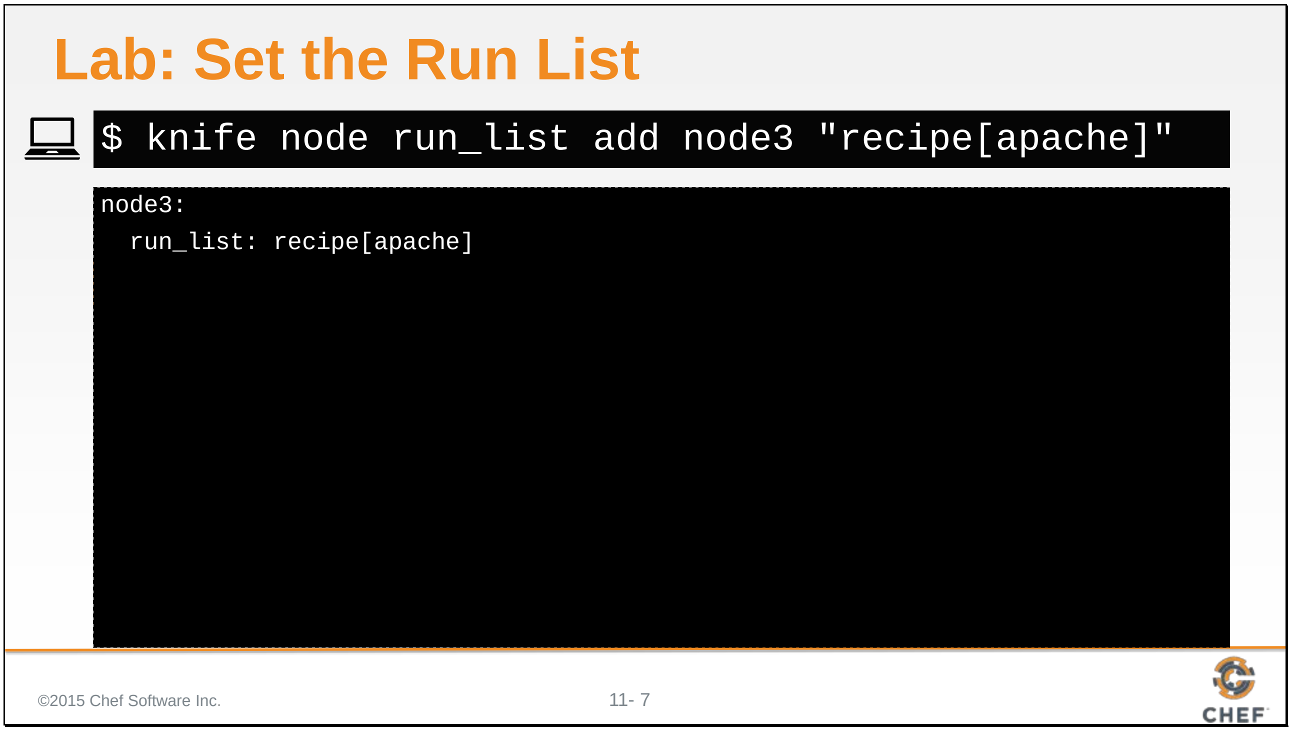
Lab: Verify the New Node

```
$ knife node show node3
```

```
Node Name:   node3
Environment: _default
FQDN:        ip-172-31-0-127.ec2.internal
IP:          54.210.86.164
Run List:
Roles:
Recipes:
Platform:    centos 6.6
Tags:
```


Verify that you bootstrapped the node.

Slide 7



Set the run list for this node by running the apache cookbook's default recipe.

Slide 8

Lab: Converge the Run List

```
$ knife ssh "*:*" -x USERNAME -P PWD "sudo chef-client"
```

```
ec2-54-175-46-24.compute-1.amazonaws.com  Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-175-46-24.compute-1.amazonaws.com  resolving cookbooks for run list: ["apache"]
ec2-54-210-86-164.compute-1.amazonaws.com resolving cookbooks for run list: ["apache"]
ec2-54-210-86-164.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list: ["myhaproxy"]
ec2-54-175-46-24.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com    - apache
ec2-54-175-46-24.compute-1.amazonaws.com  Compiling Cookbooks...
ec2-54-175-46-24.compute-1.amazonaws.com  Converging 3 resources
ec2-54-175-46-24.compute-1.amazonaws.com  Recipe: apache::server
```


Apply that run list by logging into that node and running sudo chef-client or remotely administer the node with the 'knife ssh' command as shown here.

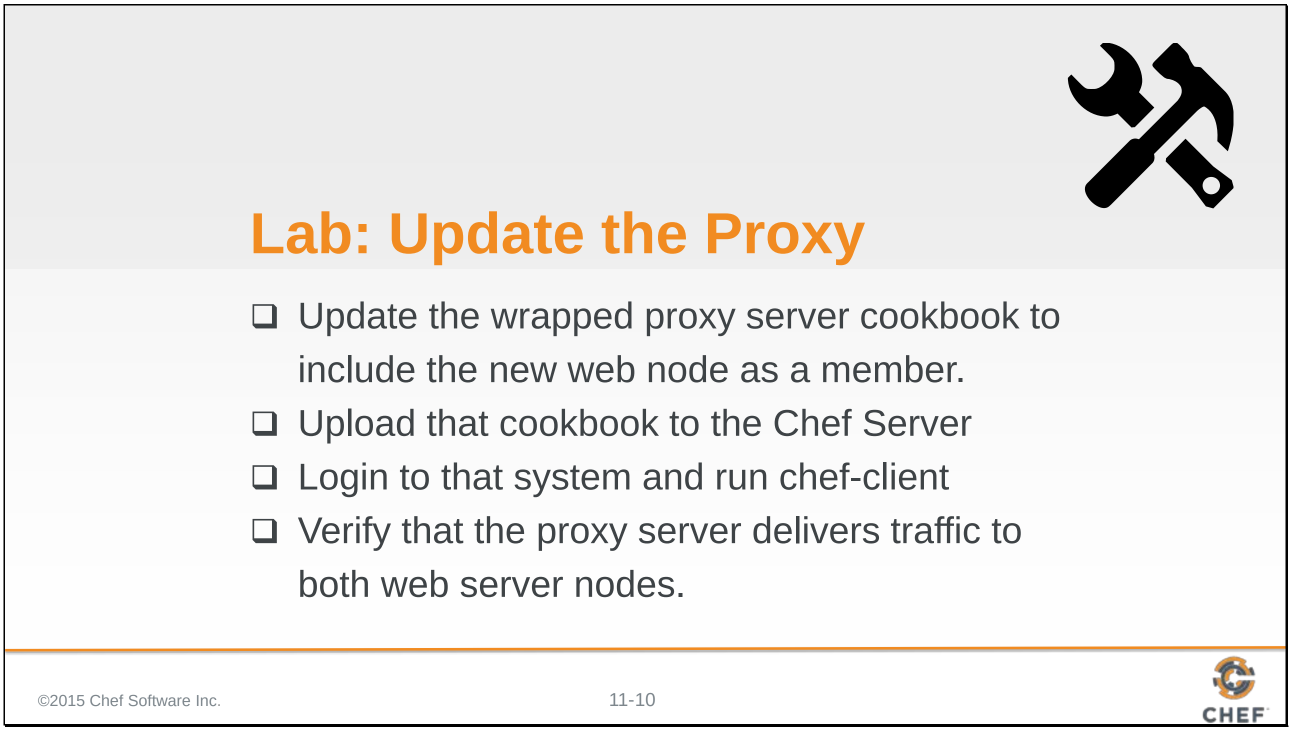
Slide 9

Verify that the New Node Serves the Page

 11- 9

Verify that the node serves up the default html page that contains the node's internal IP address and hostname.

Slide 10



Now that you have the third node. It is time to add that node to the member's list for the proxy server.

Slide 11

Lab: Capture Node's Public Host Name and IP

```
$ knife node show node3 -a cloud
```

```
node1:
  cloud:
    local_hostname:  ip-172-31-8-64.ec2.internal
    local_ipv4:      172.31.8.64
    private_ips:     172.31.8.64
    provider:        ec2
    public_hostname: ec2-54-176-64-173.us-west-1.compute.amazonaws.com
    public_ips:      54.175.46.48
    public_ipv4:     54.175.46.48
```


If you use 'knife node show' to display the 'cloud' attribute for node3, you will see the local, private, and public connection information.

Capture and write down the public hostname and the public ipv4 address of node3. You will need this in the recipe you are going to write.

Slide 12

Lab: Add the Other Web Server to Proxy

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node.default['haproxy']['members'] = [{
    "hostname" => "ec2-204-236-155-223.us-west-1.compute.amazonaws.com",
    "ipaddress" => "ec2-204-236-155-223.us-west-1.compute.amazonaws.com",
    "port" => 80,
    "ssl_port" => 80
  },{
    "hostname" => "ec2-54-176-64-173.us-west-1.compute.amazonaws.com",
    "ipaddress" => "ec2-54-176-64-173.us-west-1.compute.amazonaws.com",
    "port" => 80,
    "ssl_port" => 80
  }
]

include_recipe "haproxy::default"
```


Add the second web server (node3) to the Proxy server's members list. You may need to run 'knife node show node3 -a cloud' to get the hostname and ipaddress values.

Slide 13

Lab: Update the Version

~/chef-repo/cookbooks/myhaproxy/metadata.rb

```
name             'myhaproxy'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures myhaproxy'
long_description 'Installs/Configures myhaproxy'
version          '0.2.0'

depends 'haproxy', '~> 1.6.6'
```


Update the version number in myhaproxy cookbook's metadata.

Slide 14

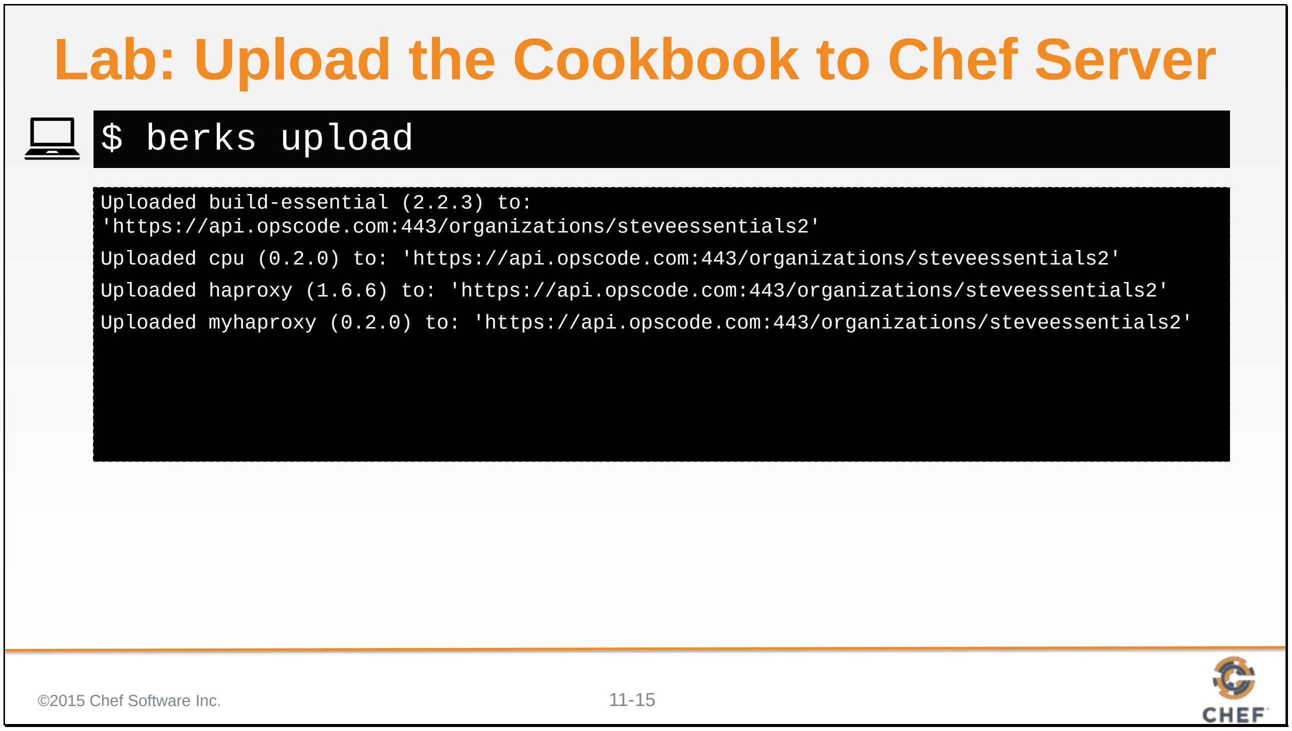
Lab: CD and Then Run berks install

```
$ cd ~/chef-repo/cookbooks/myhaproxy
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'myhaproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using build-essential (2.2.3)
Using cpu (0.2.0)
Using haproxy (1.6.6)
Using myhaproxy (0.2.0) from source at .
```


Change into the 'myhaproxy' cookbook directory and then run 'berks install' to install any dependencies for the 'myhaproxy' cookbook.

Slide 15



Run 'berks upload' to upload the myhaproxy cookbook to Chef Server.

Slide 16

Lab: Converge the Cookbook

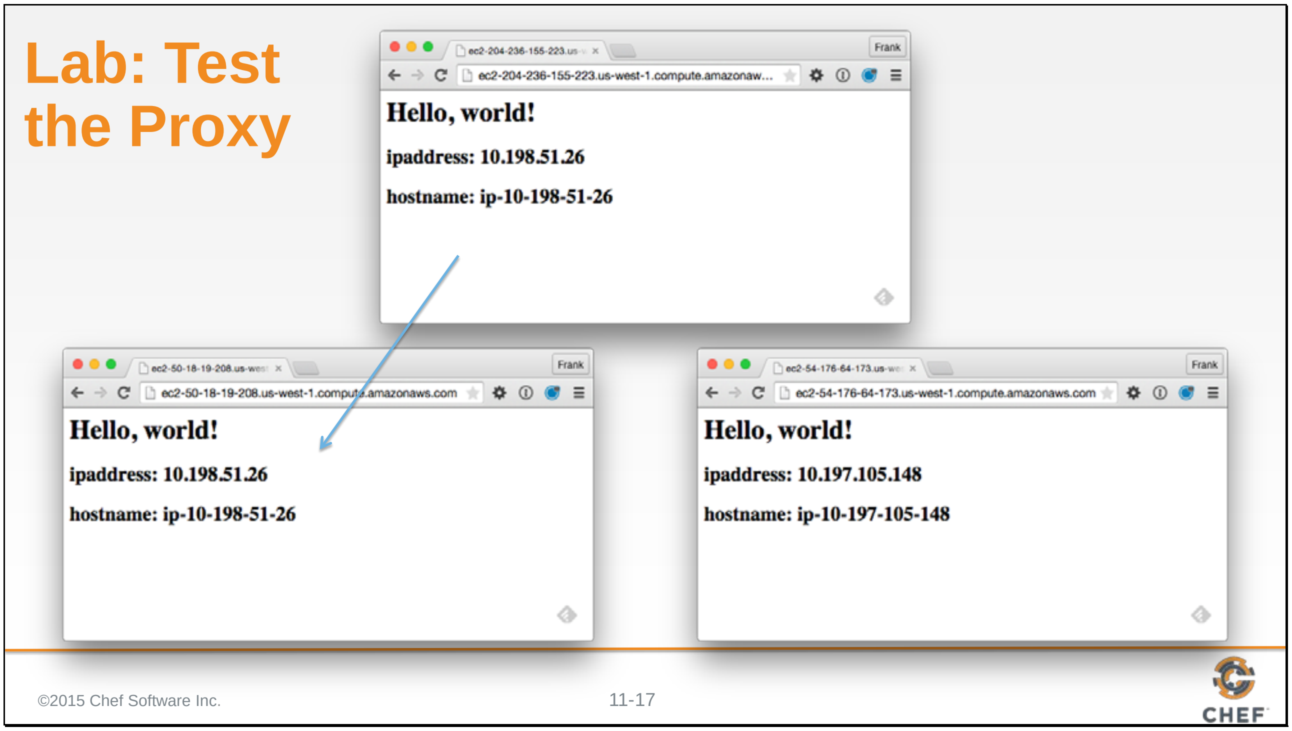
```
$ knife ssh "*:*" -x USERNAME -P PWD "sudo chef-client"
```

```
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-175-46-24.compute-1.amazonaws.com  Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-54-175-46-24.compute-1.amazonaws.com  resolving cookbooks for run list:
["apache"]
ec2-54-175-46-24.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com    - apache
ec2-54-175-46-24.compute-1.amazonaws.com  Compiling Cookbooks...
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com  Converging 3 resources
ec2-54-175-46-24.compute-1.amazonaws.com  Recipe: apache::server
ec2-54-210-192-12.compute-1.amazonaws.com   - build-essential
```


Converge the cookbook by logging into that node and running 'sudo chef-client' or remotely administer the node with the 'knife ssh' command as shown here.

Within the output you should see the haproxy configuration file will update with a new entry that contains the information of the second member (node3).

Slide 17

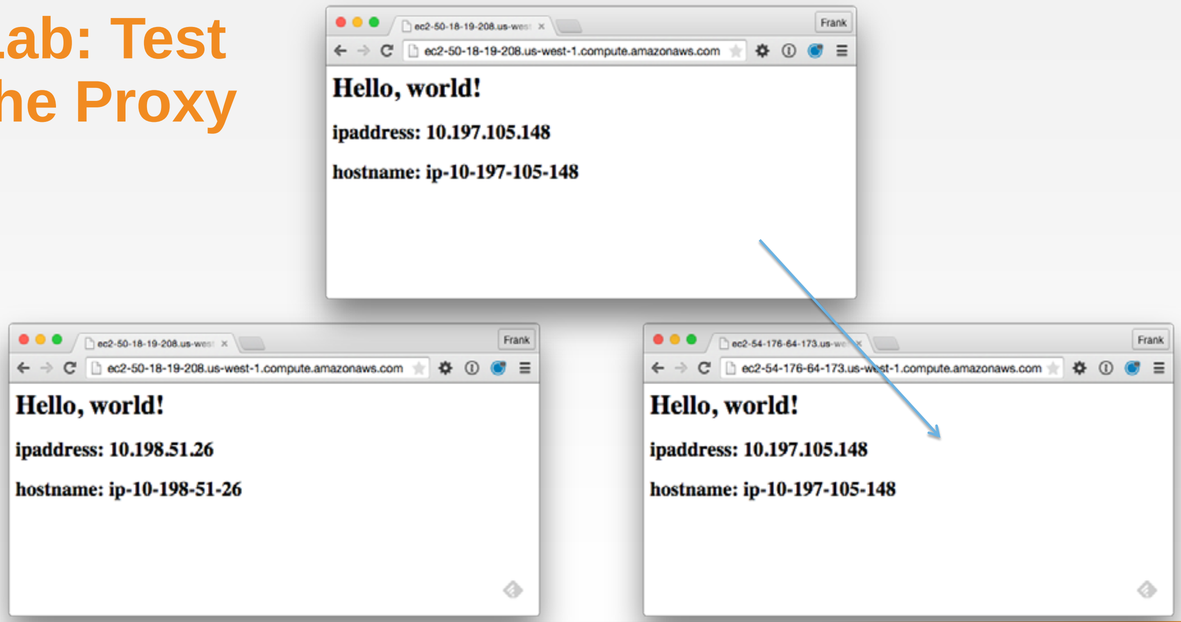


Point a web browser to the URL of your Proxy server and then click Refresh a few times. You should see each of your web server's HTML page as the Proxy server switches between each web server.

This is not a very scientific way of seeing that the proxy server is balancing requests between these two web nodes.

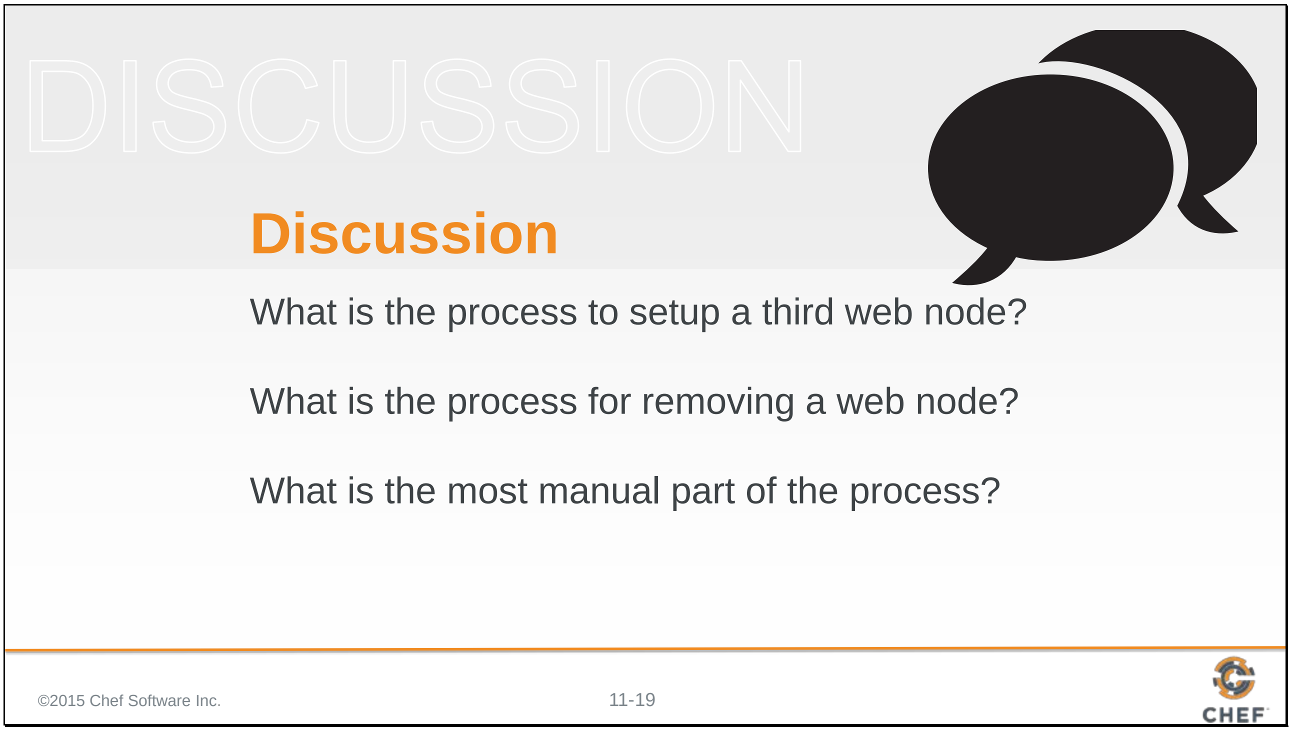
Slide 18

Lab: Test the Proxy



CHEF

Slide 19



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

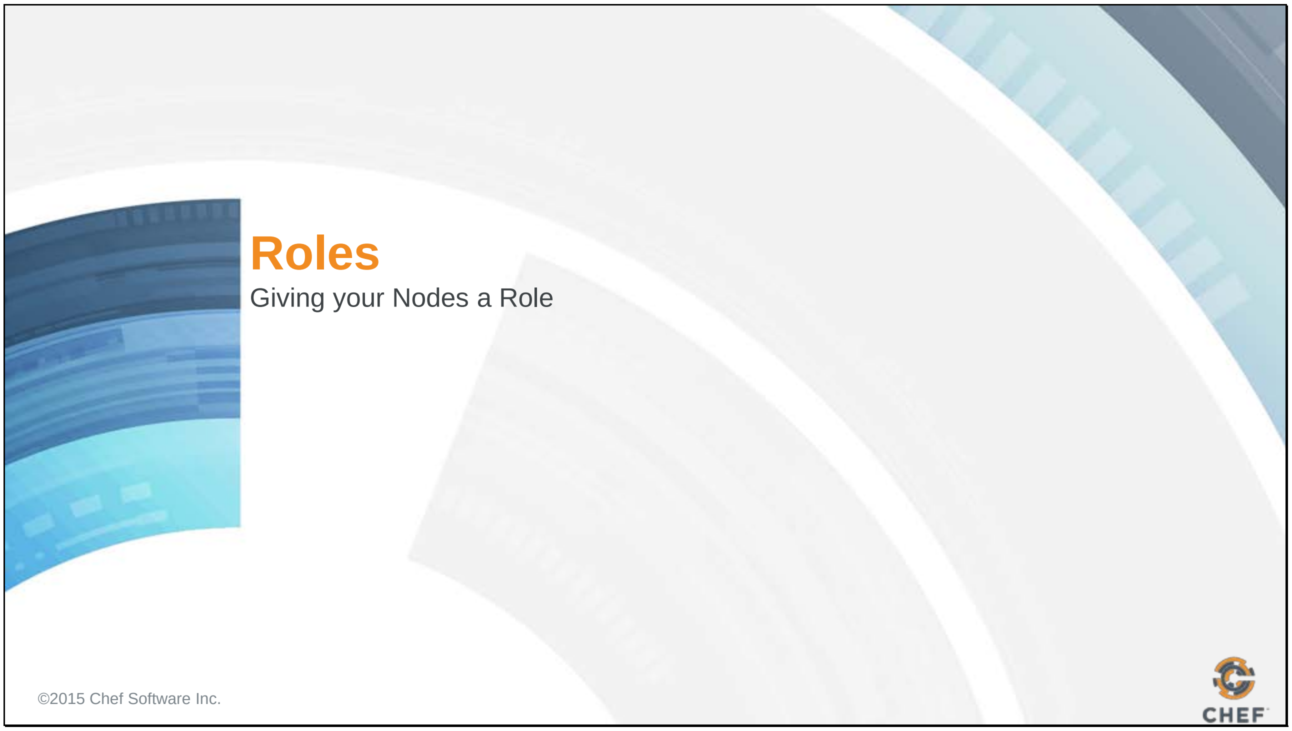
Slide 20


DISCUSSION
Q&A
What questions can we help you answer?
©2015 Chef Software Inc.
11-20
CHEF

Slide 21

CHEF™

12: Roles



Slide 2

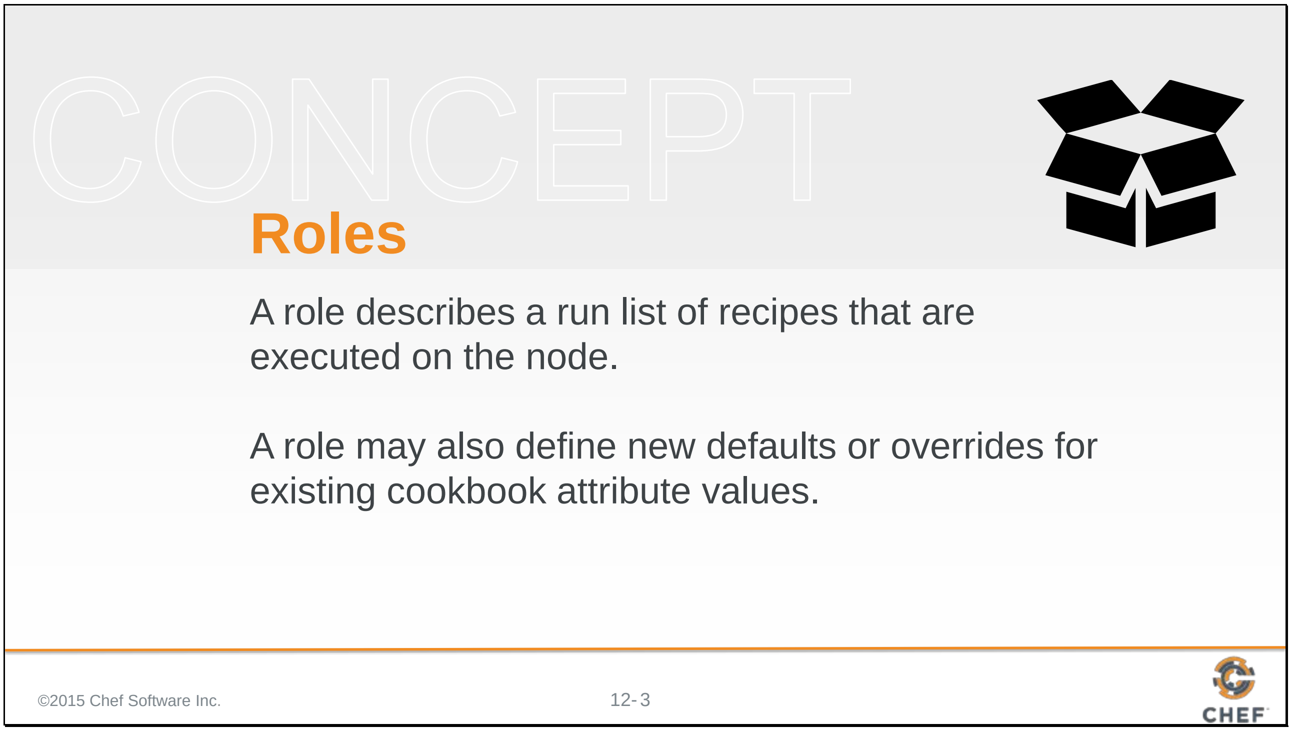
Objectives

After completing this module, you should be able to

- Assign roles to nodes so you can better describe them and configure them in a similar manner.

In this module you will give your nodes a role to better describe them so you can configure them in a similar manner.

Slide 3

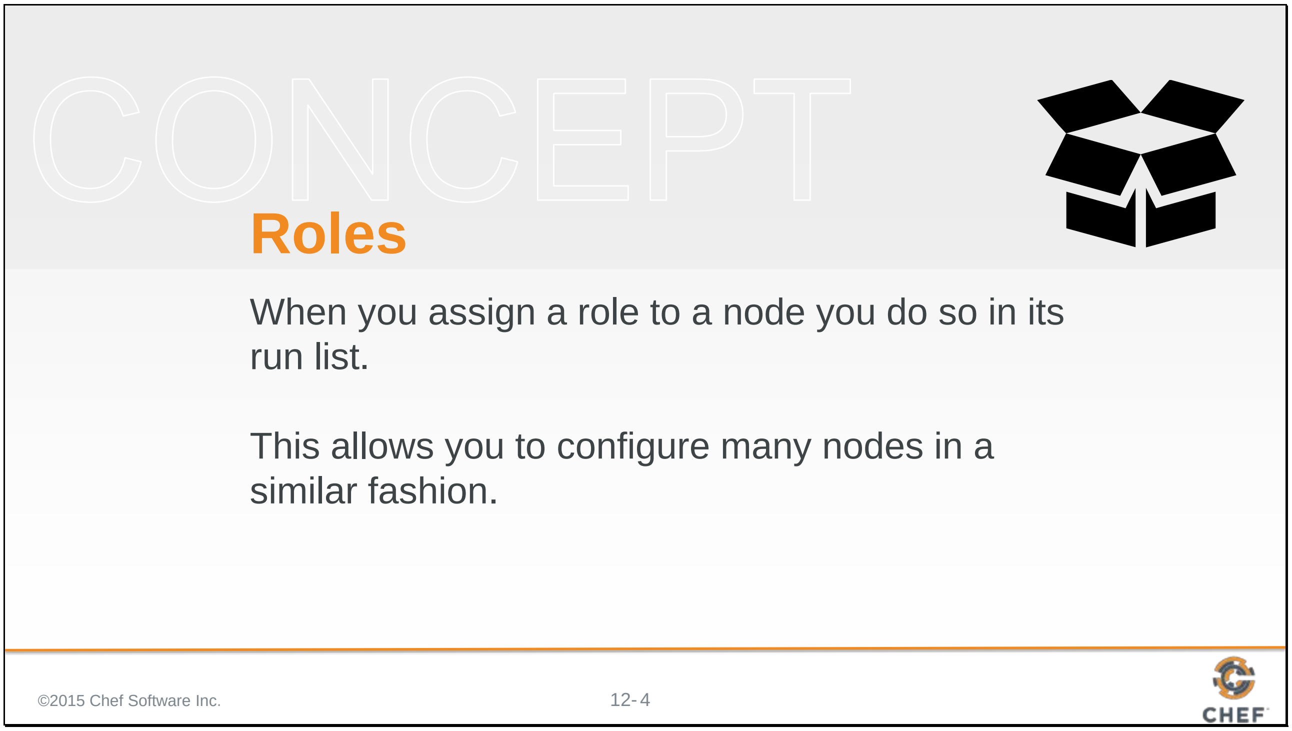


Up until this point it has been a mouthful to describe the nodes within our organization. We have two nodes, node1 and node3, that have the apache cookbook's default recipe in their run list. We have one node, node2, that has the myhaproxy cookbook's default recipe in its run list.

The Chef Server allows us to create and manage roles. A role describes a run list of recipes that are executed on the node. A role may also define new defaults or overrides for existing cookbook attribute values. Similar to what we accomplished with the wrapper cookbook.

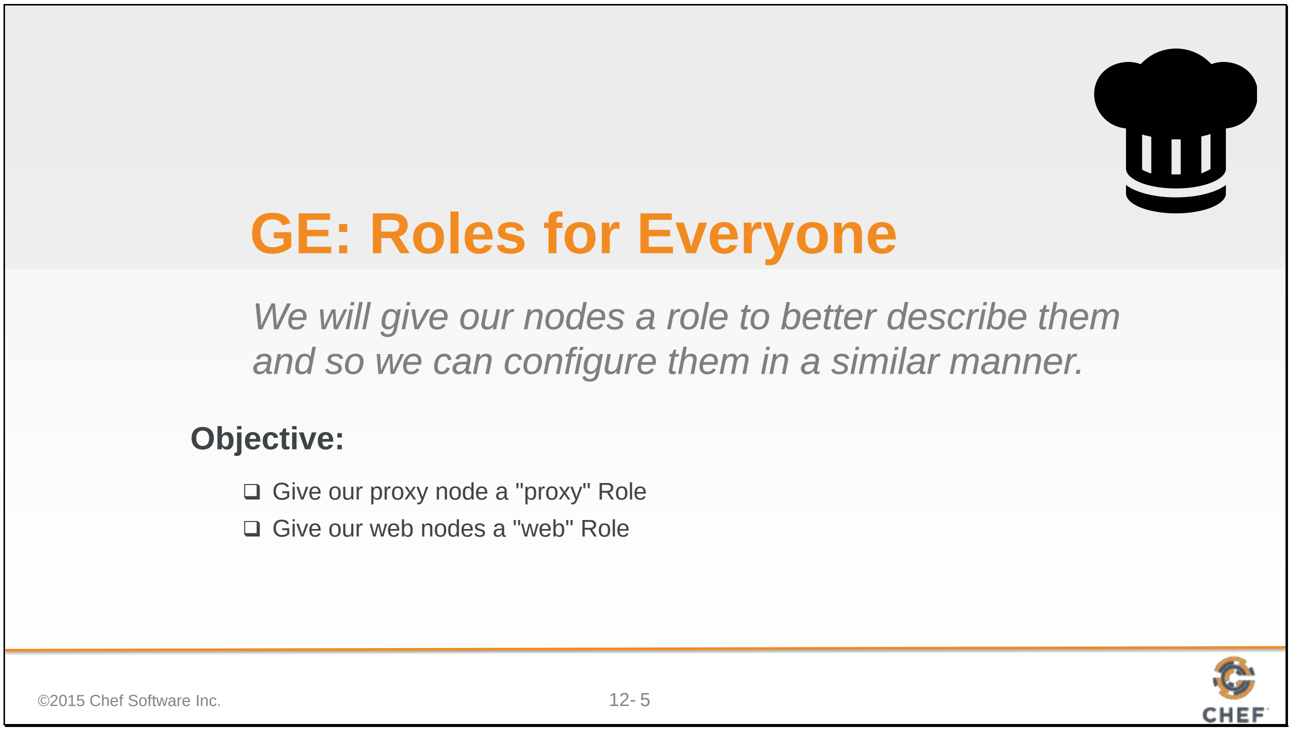
A node may have zero or roles assigned to it.

Slide 4



When you assign a role to a node you do so in its run list. This allows us to configure many nodes in a similar fashion because we no longer need to re-create a long run list for each node--we simply give it a role or all the roles it needs to accomplish its desired function.

Slide 5



In this section you will create a proxy role and assign it to the run list of node2. You will also will create a web role and assign it to the run list of node1 and node3.

This is particularly powerful because we will no longer have to manage each of these identical nodes individually, instead we can make changes to the role that they share and all of the nodes that have this role will update accordingly.

Slide 6

GE: What Can 'knife role' Do?

```
$ cd ~/chef-repo
$ knife role --help
```

```
** ROLE COMMANDS **
knife role bulk delete REGEX (options)
knife role create ROLE (options)
knife role delete ROLE (options)
knife role edit ROLE (options)
knife role env_run_list add [ROLE] [ENVIRONMENT] [ENTRY[,ENTRY]] (options)
knife role env_run_list clear [ROLE] [ENVIRONMENT]
knife role env_run_list remove [ROLE] [ENVIRONMENT] [ENTRIES]
knife role env_run_list replace [ROLE] [ENVIRONMENT] [OLD_ENTRY] [NEW_ENTRY]
knife role env_run_list set [ROLE] [ENVIRONMENT] [ENTRIES]
knife role from file FILE [FILE..] (options)
```


Return to the base of your Chef repository and then run 'knife role --help' to see the available commands. Similar to other commands, you can see that 'knife role' supports the ability to list currently-defined roles.

Slide 7

When you run 'knife role list' you can see from its lack of response that you have no roles defined.

Slide 8

GE: Create a Roles Directory

```
$ mkdir roles
```


Create a **roles** directory if necessary. If you are using the Chef Starter Kit this directory may already exist.

Slide 9

GE: Create the proxy.rb

~/chef-repo/roles/proxy.rb

```
name "proxy"
description "Proxy Server"
run_list "recipe[myhaproxy]"
```

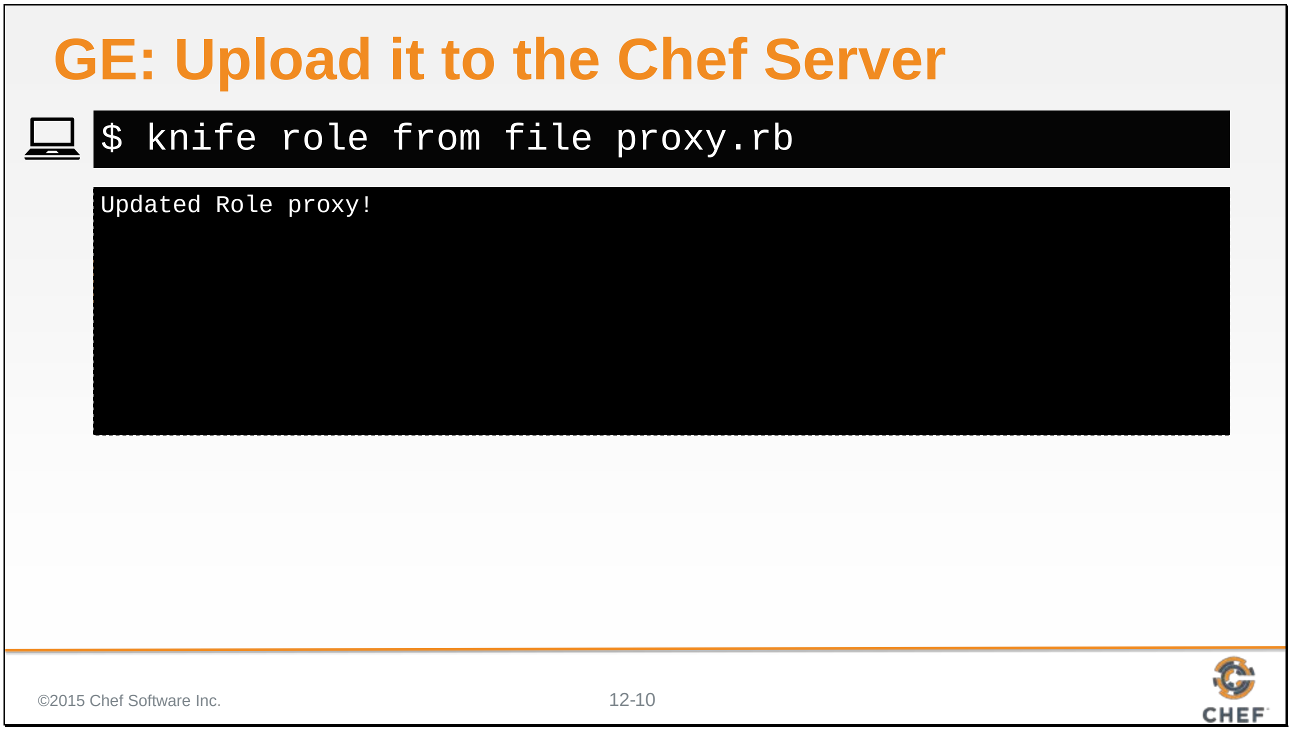

Create a file named proxy.rb. This is a ruby file that contains specific methods that allow you to express details about the role. You'll see that the role has a name, a description, and run list.

The name of the role as a practice will share the name of the ruby file unless it cannot for some reason. The name of the role should clearly describe what it attempts accomplish.

The description of the role helps reinforce or clarify the intended purpose of the role. When selecting a role name that is not clear it is important that a helpful description is provided to help ensure everyone on the team understands its purpose.

The run list defines the list of recipes that give the role its purpose. Currently the proxy role defines a single recipe - the myhaproxy cookbook's default recipe.

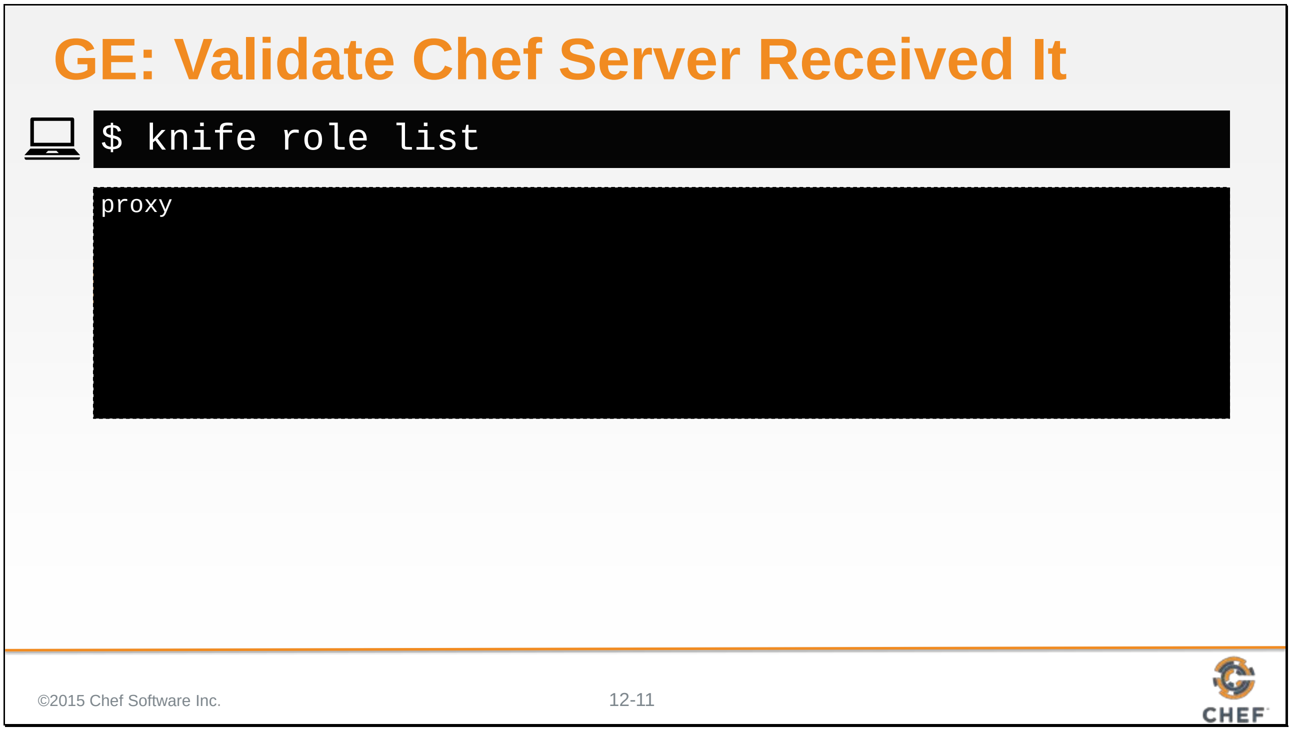
Slide 10



Now you need to upload it to the Chef Server. This is done through the command 'knife role from file proxy.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file proxy.rb.

Slide 11



With the role uploaded, it is time to validate that the Chef Server received it correctly. We can do that by again asking the Chef Server for a list of all the roles on the system.

Slide 12

GE: View Details of the Role

```
$ knife role show proxy
```

```
chef_type:            role
default_attributes:
description:          Proxy Server
env_run_lists:
json_class:           Chef::Role
name:                 proxy
override_attributes:
run_list:             recipe[myhaproxy]
```


You can ask for more details about a specific role using the above command. In this example we are requesting specific details about the role named proxy.

Slide 13

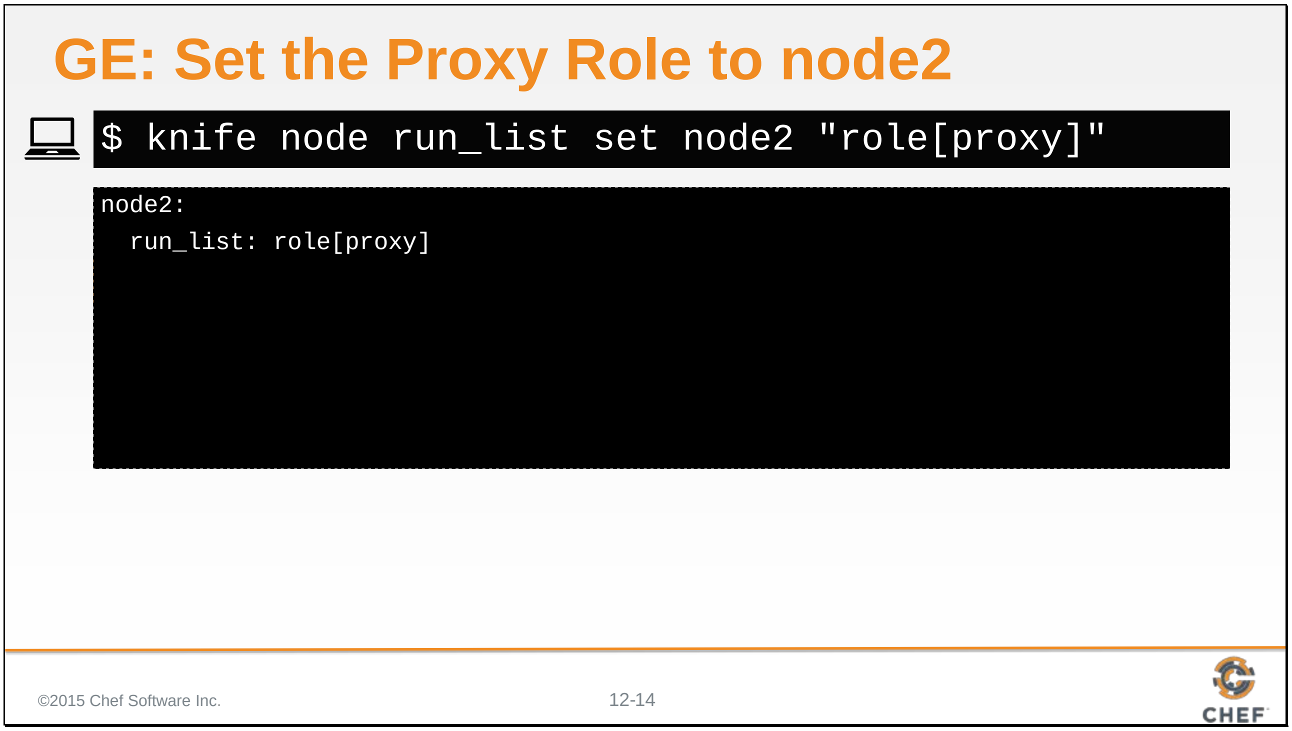
GE: Run 'knife node --help'

```
$ knife node --help
```

```
** NODE COMMANDS **
knife node bulk delete REGEX (options)
knife node create NODE (options)
knife node delete NODE (options)
knife node edit NODE (options)
knife node environment set NODE ENVIRONMENT
knife node from file FILE (options)
knife node list (options)
knife node run_list add [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list remove [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list set NODE ENTRIES (options)
knife node show NODE (options)
```


Run 'knife node --help' to see its options.

Slide 14



The last step is to redefine the run list for node2. We want the run list to contain only the proxy role.

Previously, we used the command 'knife node run_list add' to append a new item to the existing run list. There is also a command that allows us to remove an item from the run list. There is a command that allows us to set the run list to a value provided. This will replace the existing run list with a new one that we provide.

Slide 15

GE: Verify the Run List

```
$ knife node show node2
```

```
Node Name:   node2
Environment: _default
FQDN:        ip-172-31-0-128.ec2.internal
IP:          54.210.192.12
Run List:    role[proxy]
Roles:
Recipes:     myhaproxy, myhaproxy::default, haproxy::default, haproxy::install_package
Platform:    centos 6.6
Tags:
```


After you update the run list, you can verify that the node has the correctly-defined run list by running 'knife node show node2'.

Slide 16

GE: Converge All the Proxy Nodes

```
$ knife ssh "role:proxy" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com   - build-essential
ec2-54-210-192-12.compute-1.amazonaws.com   - cpu
ec2-54-210-192-12.compute-1.amazonaws.com   - haproxy
ec2-54-210-192-12.compute-1.amazonaws.com   - myhaproxy
ec2-54-210-192-12.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-192-12.compute-1.amazonaws.com Converging 9 resources
ec2-54-210-192-12.compute-1.amazonaws.com Recipe: haproxy::install_package
ec2-54-210-192-12.compute-1.amazonaws.com   * yum_package[haproxy] action install
(up to date) ...
```


You can use 'knife ssh' to run 'sudo chef-client' on all the nodes again to ensure that nothing has changed.

In this instance we only interested in having node2 run the command so we can get a little more creative with the search criteria and find nodes with the role proxy. In this case there is only one result.

Within the results, nothing should change. Switching over to the role did not change the fundamental recipes that were applied to the node.

Slide 17

Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- ✓ Give our proxy node a "proxy" Role
- ❑ Give our web nodes a "web" Role

Now if you want to setup a new node in the future to act as a proxy server, you can now simply set the new node's run list to be the proxy role and it will have identical functionality with all the other nodes that define this role.

Slide 18

Lab: Define a Web Role

- ❑ Create a role named 'web' that has the run list 'recipe[apache]'
- ❑ Set node1's run list to be "role[web]"
- ❑ Set node3's run list to be "role[web]"

In this lab, define a new role named 'web' that has the run list: including the apache cookbook's default recipe.

When you're done defining the role, upload it to the Chef Server, and then set the run list on node1 and node3 to the role that you have defined.

And for good measure, though nothing should have changed, run 'sudo chef-client' on both node1 and node3 to ensure that no functionality has been lost.

Slide 19

Lab: Create the web.rb File

~/chef-repo/roles/web.rb

```
name "web"
description "Web Server"
run_list "recipe[apache]"
```


First we create a file named web.rb in the roles directory.

The name of the role is web. The description should be Web Server. The run list you define should contain the apache cookbook's default recipe.

Slide 20

Lab: Upload the web.rb File

```
$ knife role from file web.rb
```

```
Updated Role web!
```


You need to share the role with the Chef Server so upload that file.

Use the command 'knife role from file web.rb'. 'knife' knows where to look for that role to upload it.

Slide 21

Lab: Verify the Role on the Chef Server

```
$ knife role list
```

```
proxy
web
```


Verify that the role can be found on the Chef Server.

Slide 22

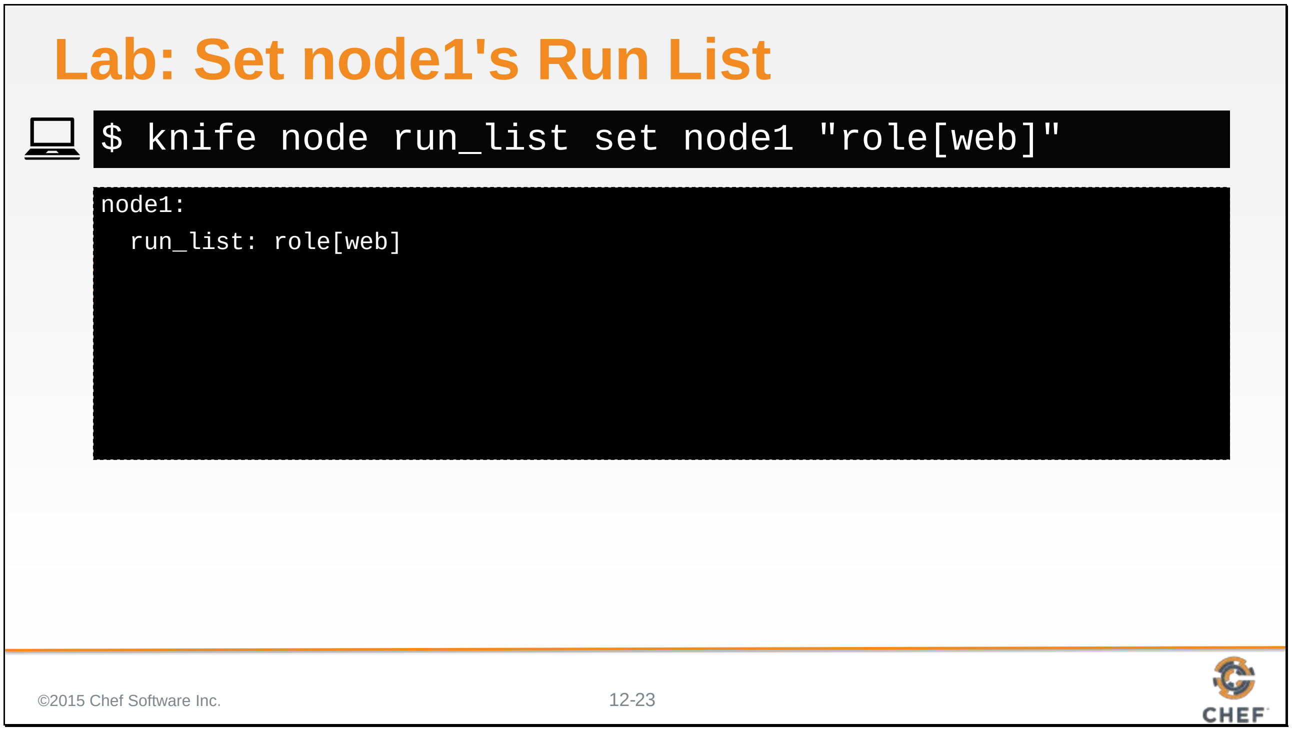
Lab: Verify Specific Information About the Role

```
$ knife role show web
```

```
chef_type:          role
default_attributes:
description:        Web Server
env_run_lists:
json_class:         Chef::Role
name:               web
override_attributes:
run_list:           recipe[apache]
```


Verify specific information about the role. Specifically, does it have the run list that we defined?

Slide 23



Set node1's run list to be the web role.

Slide 24

Lab: Set node3's Run List

```
$ knife node run_list set node3 "role[web]"
```

```
node3:
  run_list: role[web]
```


CHEF

And we then set node3's run list to be the web role.

Slide 25

Lab: Converge All Web Nodes

```
$ knife ssh "role:web" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-175-46-24.compute-1.amazonaws.com  Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com resolving cookbooks for run list:
["apache"]
ec2-54-175-46-24.compute-1.amazonaws.com  resolving cookbooks for run list:
["apache"]
ec2-54-210-86-164.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-86-164.compute-1.amazonaws.com   - apache
ec2-54-210-86-164.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-86-164.compute-1.amazonaws.com Converging 3 resources
ec2-54-210-86-164.compute-1.amazonaws.com Recipe: apache::server
ec2-54-175-46-24.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com    - apache
```


To verify that everything is working the same as before, run 'knife ssh' for both of these nodes. In this instance the query syntax is going to find all nodes with the role set to web.

Slide 26

Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

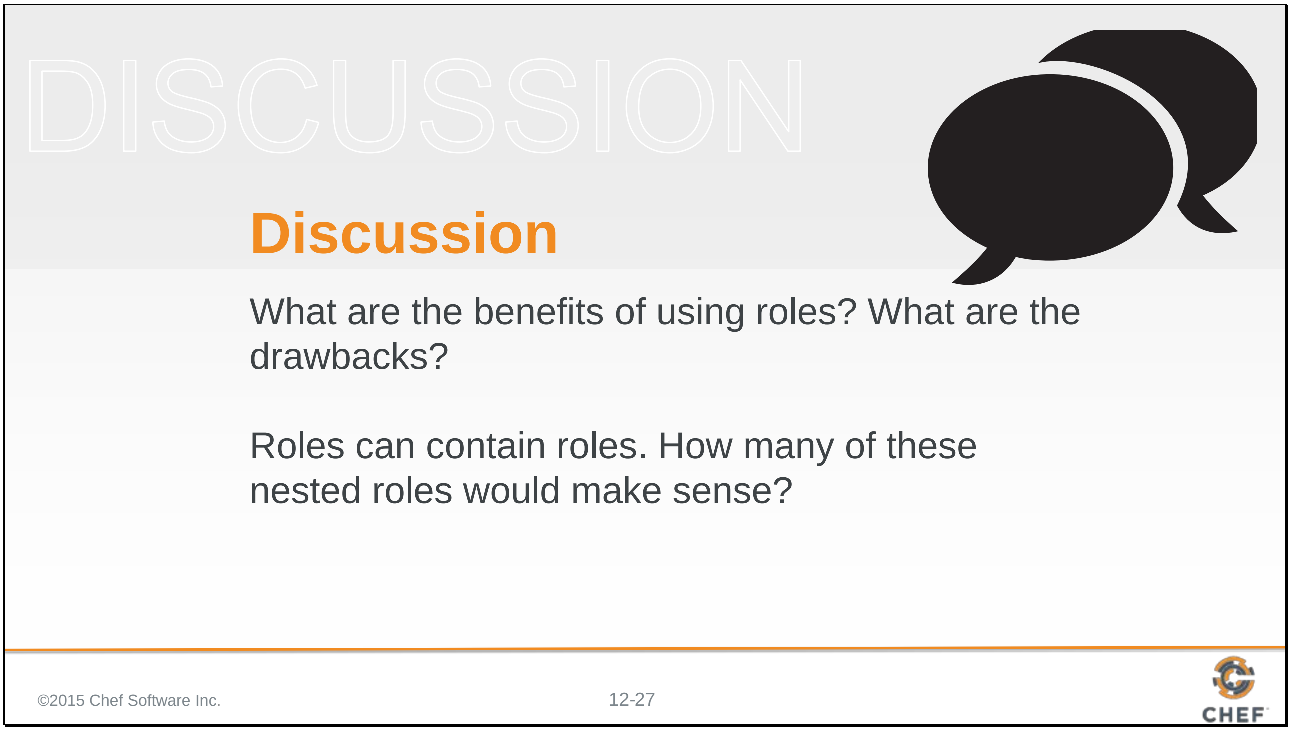
- ✓ Give our proxy node a "proxy" Role
- ✓ Give our web nodes a "web" Role

CHEF

With that we now have made it far easier to talk about our nodes. We can more casually describe a node as a 'web server' node or a 'proxy server' node.

In the future if we needed to ensure that these types of nodes needed to run additional recipes, we could return to the role file, update its run list, and then upload it to the Chef Server again.

Slide 27



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 28

DISCUSSION

Q&A

What questions can we help you answer?

Slide 29

CHEF™

13: Search


Search
Update a Cookbook to Dynamically Use Nodes with the Web Role
©2015 Chef Software Inc.
CHEF

Slide 2

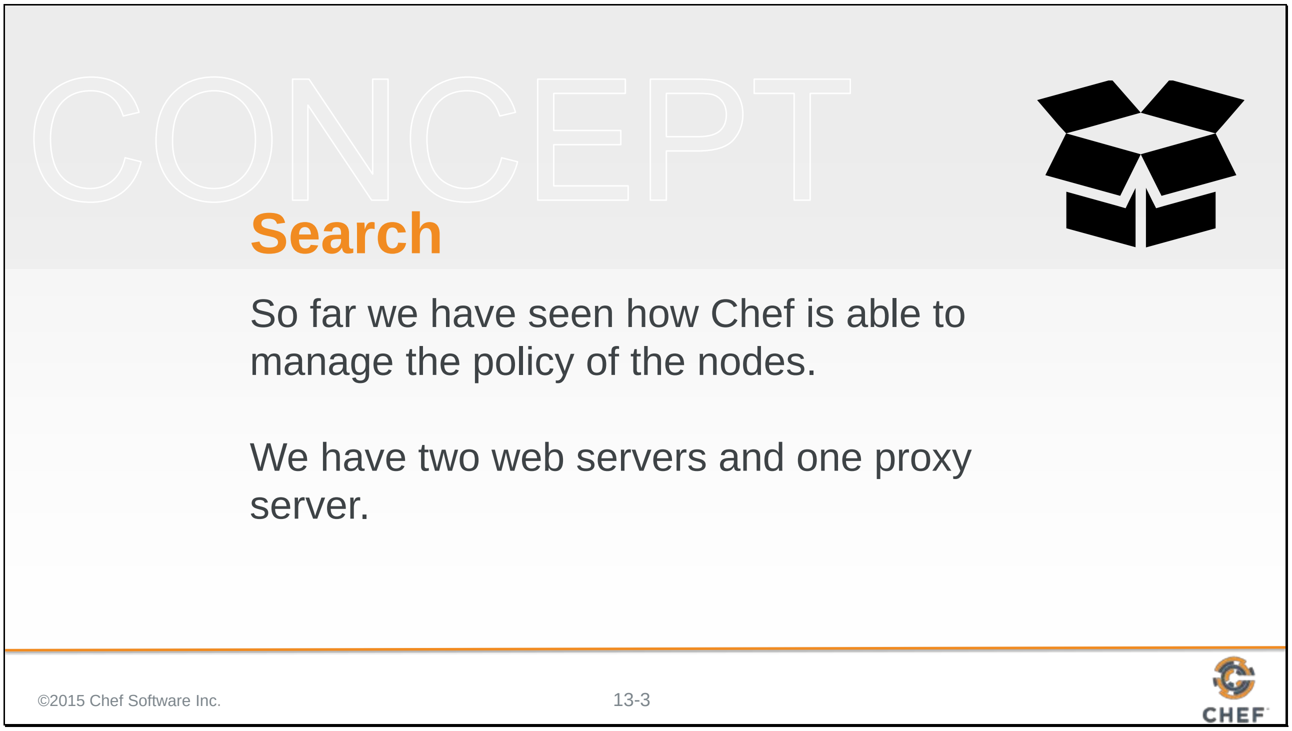
Objectives

After completing this module, you should be able to

- Describe the query syntax used in search
- Build a search into your recipe code
- Create a Ruby Array and Ruby Hash
- Update the wrapped proxy cookbook to dynamically use nodes with the web role

In this module you will learn how to describe the query syntax used in search, build a search into your recipe code, create a ruby array and ruby hash, and update the wrapped proxy cookbook to dynamically use nodes with the web role.

Slide 3



So far we have seen how Chef is able to manage the policy of the nodes within our infrastructure.

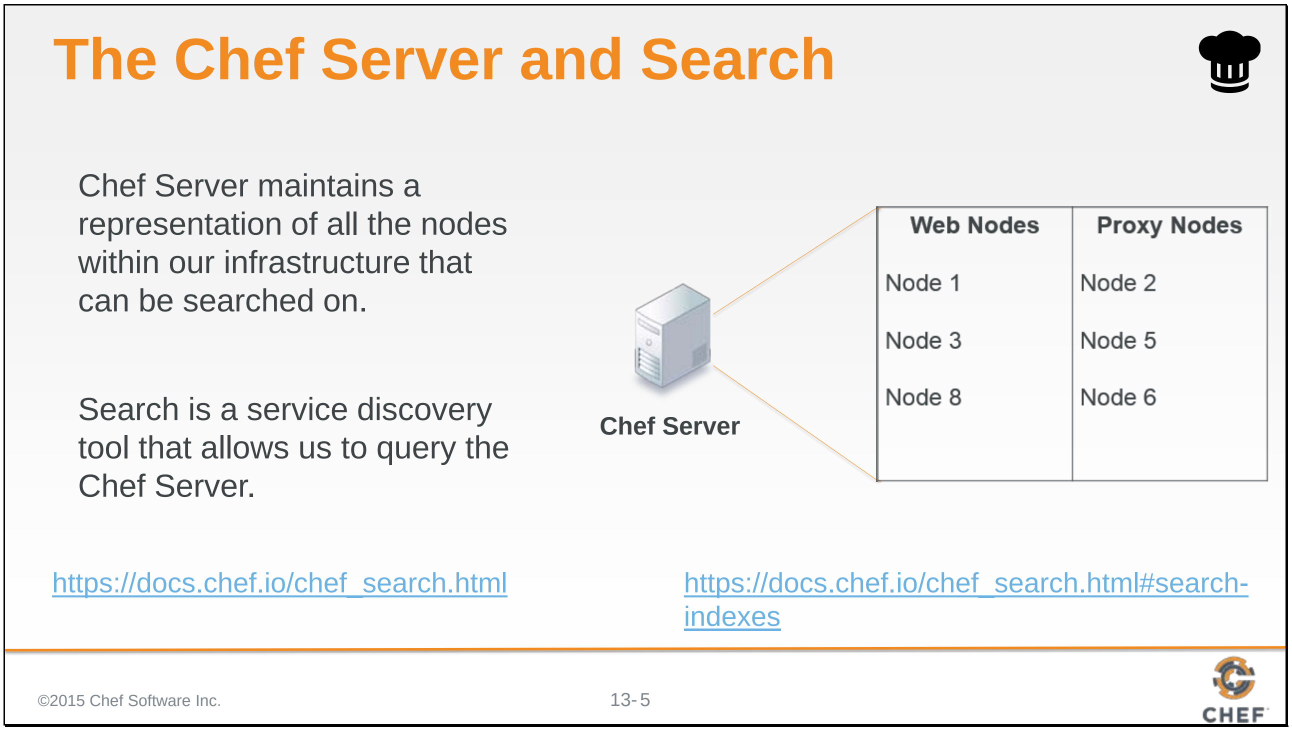
We have two web servers and one proxy server. As more customers come to our website we can continue scale up to meet that demand.

Slide 4

To add new servers as proxy members, we would need to bootstrap a new web server and then update our proxy cookbook's recipe to include that new web server. But that seems dramatically inefficient to have to update a cookbook recipe.

A more ideal solution would be for the recipe to instead discover all of the web servers within our organization and automatically add them list of available members for our proxy.

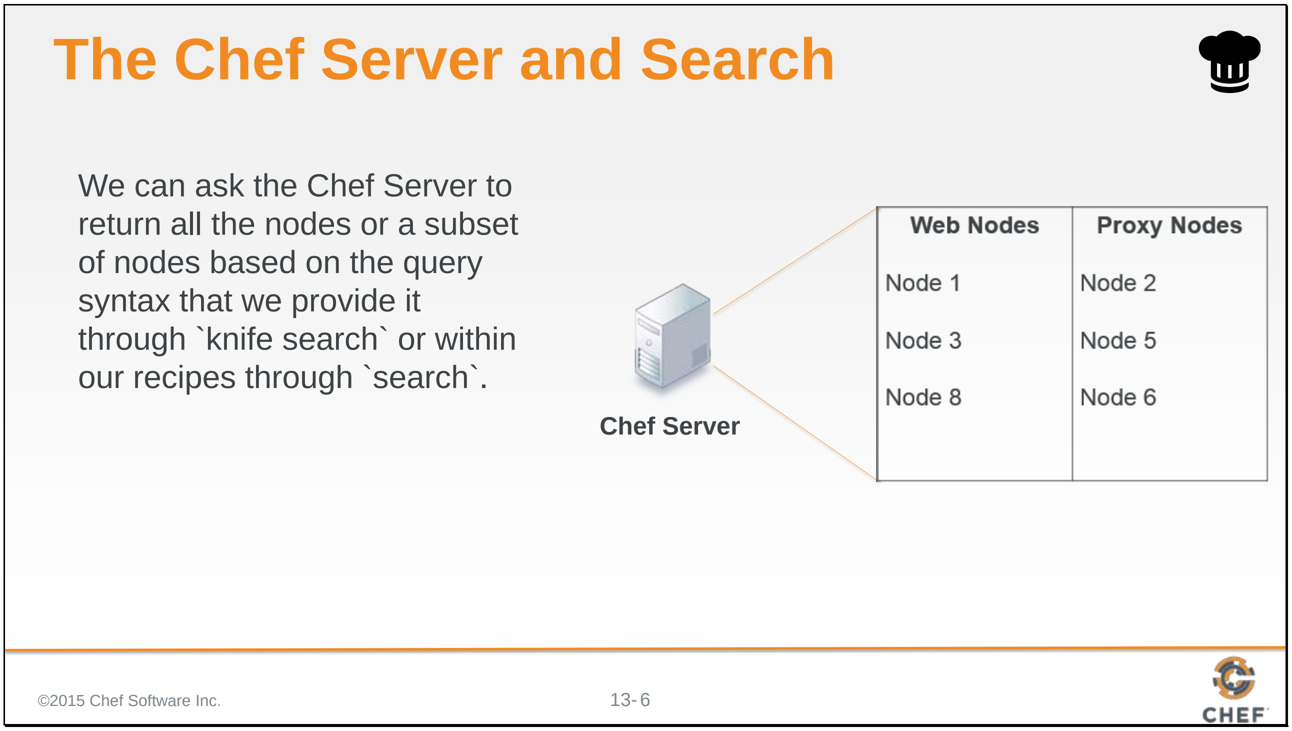
Slide 5



The Chef Server maintains a representation of all the nodes within an infrastructure and provides a way for us to discover these systems through Search.

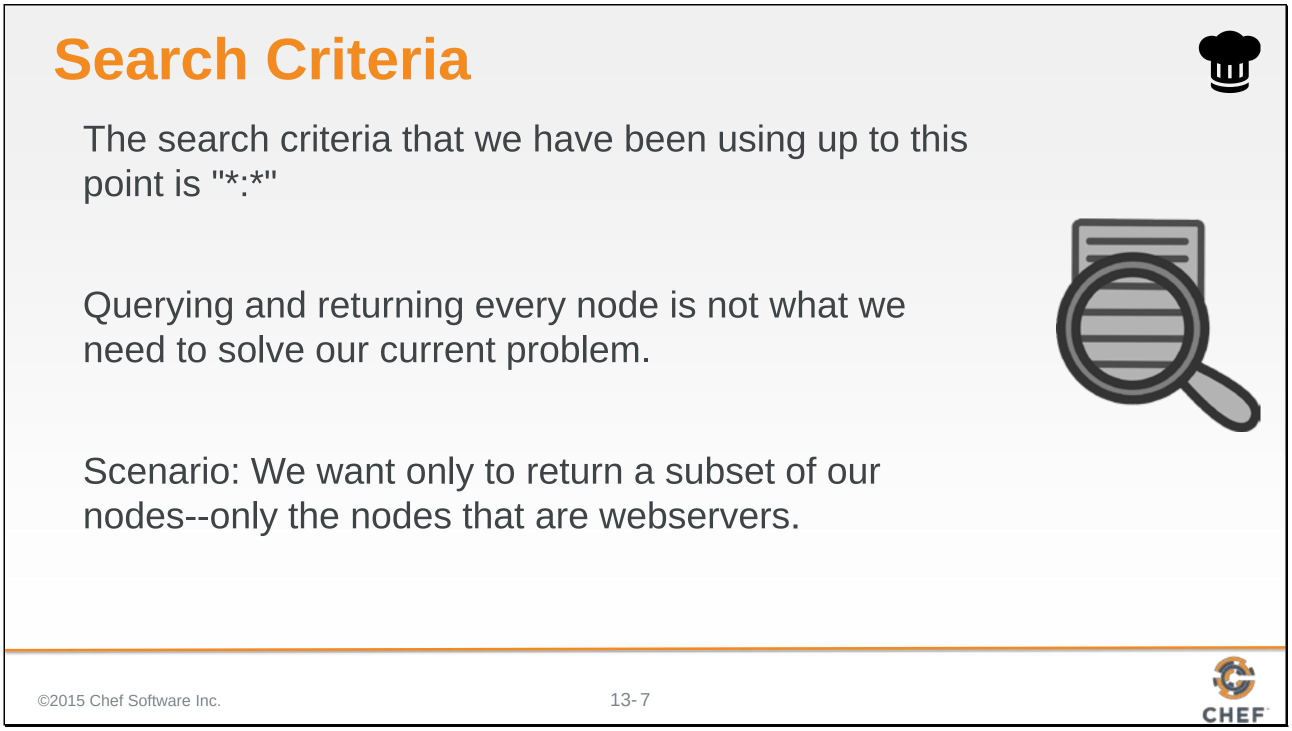
Search is a service discovery tool that allows us to query the Chef Server across a few indexes. One such index is on our nodes.

Slide 6



We can ask the Chef Server to return back to us all the nodes or a subset of nodes based on the query syntax that we provide it through the knife command `knife search` or within our recipes through the `search` method.

Slide 7

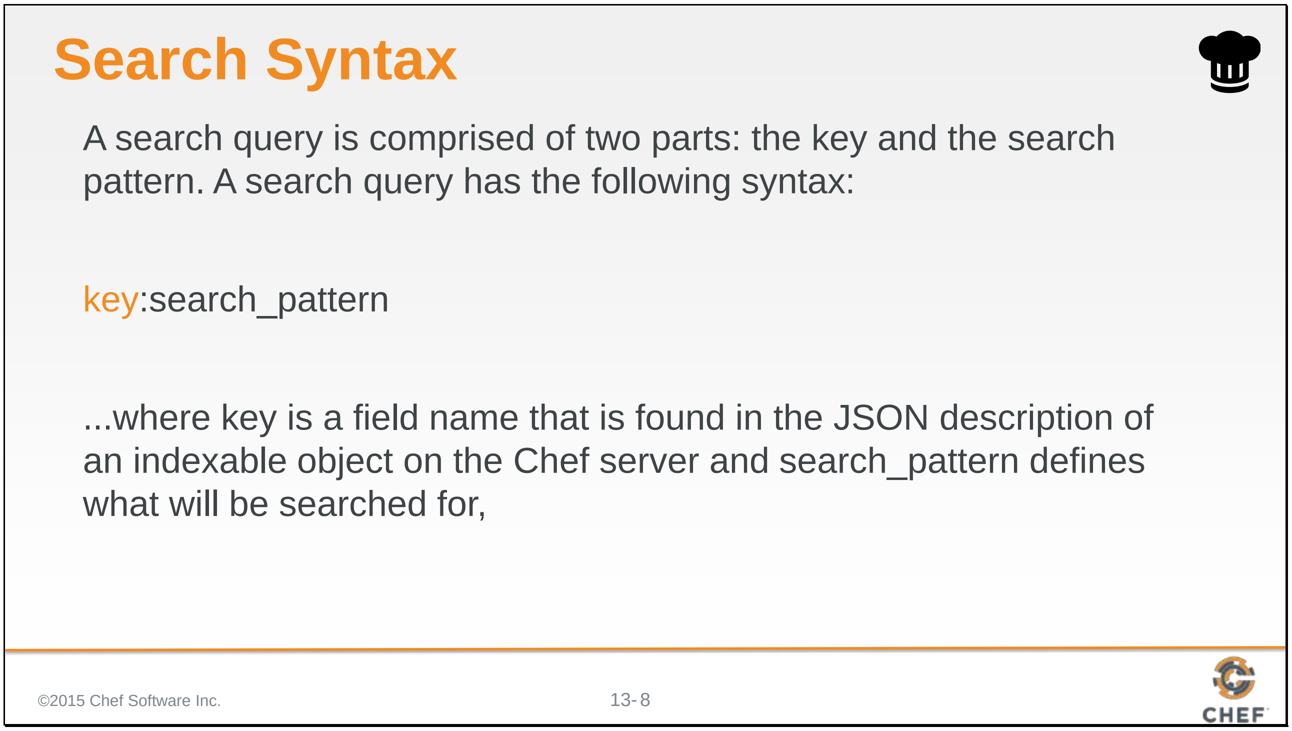


We have been using a form of the search criteria already when we have employed the `knife ssh` command. The search criteria that we have been using up to this point is "*:*" which we explained matched every node within our infrastructure.

Querying and returning every node is not exactly what we need to solve our current problem. Scenario: We want only to return a subset of our nodes--only the nodes that are webservers.

Let's examine the search criteria more so we can understand how it works and how we can use it to find a subset of the nodes--only the nodes that are webservers.

Slide 8

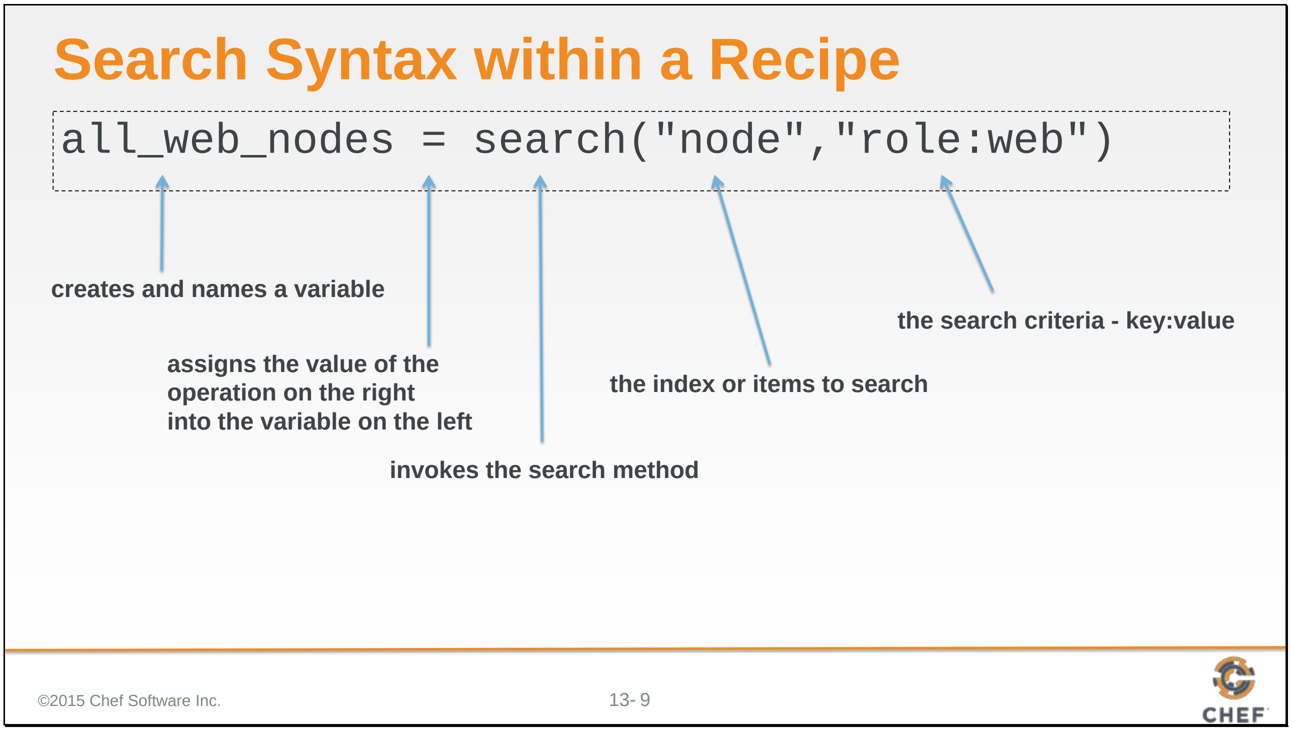


A search query is comprised of two parts: the key and the search pattern. A search query has the following syntax:

key:search_pattern

...where key is a field name that is found in the JSON description of an indexable object on the Chef server (a role, node, client, environment, or data bag) and search_pattern defines what will be searched for.

Slide 9



Search within a recipe is done through a `search` method that is available within the recipe.

The `search` method accepts two arguments. The first argument is a string or variable that contains the index or item to search on the Chef Server. These are: nodes; roles; and environments. The second argument is a string or variable that contains the search criteria to scope the results. This is using the notation "key:value".

The result of the search method is stored in a local variable that is named 'all_web_nodes'. Variables within Ruby are created immediately when you assign them.

Slide 10

Search Syntax within a Recipe

```
all_web_nodes = search("node","role:web")
```

Search the Chef Server for all node objects that have the role equal to 'web' and store the results into a local variable named "all_web_nodes'.

This example syntax could be translated to mean: Search the Chef Server for all node objects that have the role equal to 'web' and store the results into a local variable named 'all_web_nodes'.

Slide 11

Hard Coding Example

~/chef-repo/cookbooks/myhaproxy/default.rb

```
node.default['haproxy']['members'] = [{
    "hostname" => "ec2-204-236-155-223.us-west-1.compute.amazonaws.com",
    "ipaddress" => "ec2-204-236-155-223.us-west-1.compute.amazonaws.com",
    "port" => 80,
    "ssl_port" => 80
  },
  {
    "hostname" => "ec2-54-176-64-173.us-west-1.compute.amazonaws.com",
    "ipaddress" => "ec2-54-176-64-173.us-west-1.compute.amazonaws.com",
    "port" => 80,
    "ssl_port" => 80
  }
]
include_recipe "haproxy::default"
```


Previously, we had been hard coding the hostname and ipaddress values in our wrapped haproxy recipe. We can request these values from the Chef Server through the `knife node show` command.

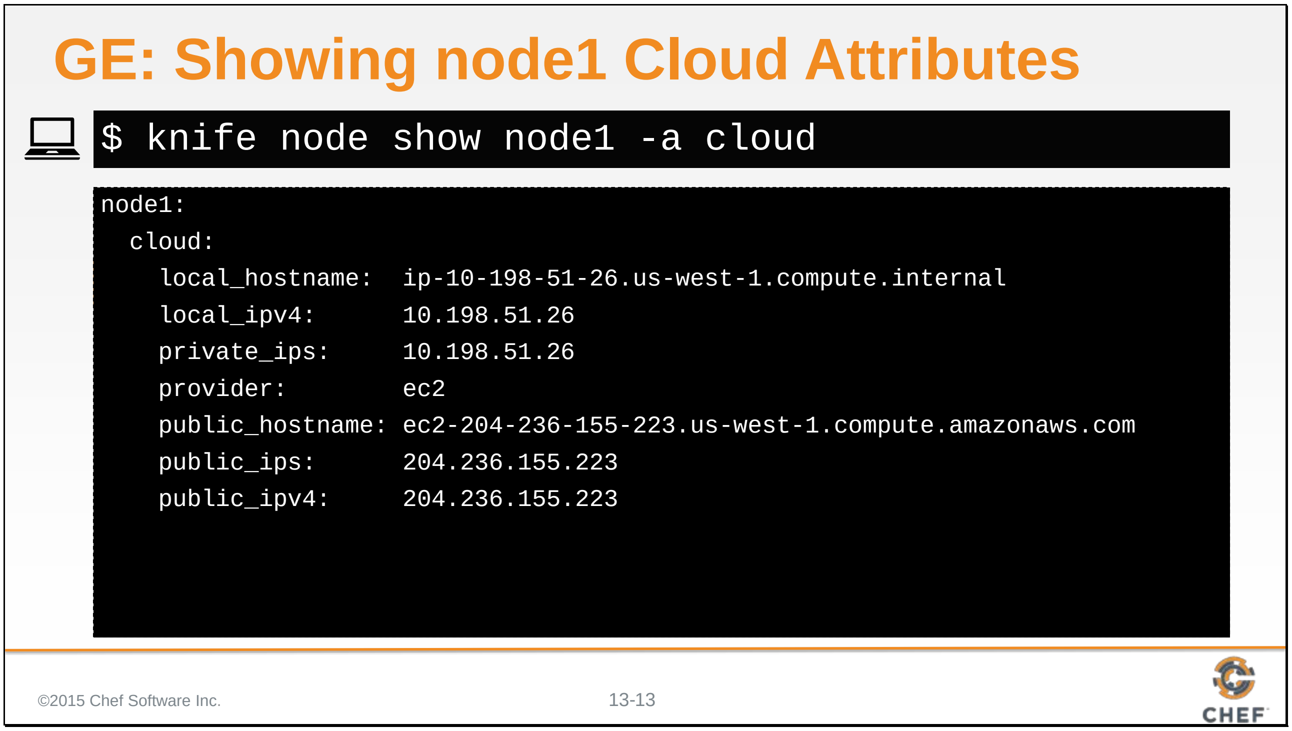
The hostname and ipaddress values are captured by Ohai and sent to the Chef Server. On the Chef Server we can query those values when we ask about specific attribute about the node.

We do that by providing the `-a` flag with the name of the attribute. Because the nodes that we manage are hosted in the cloud, these attributes are stored under a parent attribute named 'cloud'.

Slide 12

In this section we'll update the wrapped proxy cookbook to dynamically use nodes with the web role.

Slide 13



Here we are asking for all the 'cloud' attributes for 'node1'.

Slide 14

GE: Showing node3 Cloud Attributes

```
$ knife node show node3 -a cloud
```

```
node3:
  cloud:
    local_hostname:  ip-10-197-105-148.us-west-1.compute.internal
    local_ipv4:      10.197.105.148
    private_ips:     10.197.105.148
    provider:        ec2
    public_hostname: ec2-54-176-64-173.us-west-1.compute.amazonaws.com
    public_ips:      54.176.64.173
    public_ipv4:     54.176.64.173
```


Here we are asking for all the 'cloud' attributes for 'node3'.

Slide 15

GE: Remove the Hard-coded Members

~/chef-repo/cookbooks/myhaproxy/default.rb

```
node.default['haproxy']['members'] = [{
    "hostname" => "ec2-204-236-155-223.us-west-1.compute.amazonaws.com",
    "ipaddress" => "ec2-204-236-155-223.us-west-1.compute.amazonaws.com",
    "port" => 80,
    "ssl_port" => 80
  },
  {
    "hostname" => "ec2-54-176-64-173.us-west-1.compute.amazonaws.com",
    "ipaddress" => "ec2-54-176-64-173.us-west-1.compute.amazonaws.com",
    "port" => 80,
    "ssl_port" => 80
  }
]
include_recipe "haproxy::default"
```


Edit the 'myhaproxy' cookbook's default recipe and remove the current default recipe where you hard-coded the members.

Slide 16

GE: Use Search to Identify the Members

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
all_web_nodes = search("node","role:web")

include_recipe "haproxy::default"
```


Replace it with an updated recipe that searches for all nodes that have the 'web' role defined.

The search method's first parameter is asking the Chef Server to look at all the nodes within our organization.

The search method's second parameter is asking the Chef Server to only return the nodes that have been assigned the role web.

All of those nodes are stored in a local variable named `all_web_nodes`. This is an array of node objects. It may contain zero or more nodes that match the search criteria.

Slide 17

Creating an Array to Store the Converted Members

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
all_web_nodes = search("node","role:web")

members = []

#TODO: Convert each web node into an array of
hashes

node.default['haproxy']['members'] = members

include_recipe "haproxy::default"
```


Unfortunately we cannot simply assign our array of web nodes into the haproxy's members attributes because it needs a hash that contains the keys 'hostname', 'ipaddress', 'port', and 'ssl_port'.

We will need to convert each of the web node objects into a structure that the haproxy member's attribute expects.

First we create an empty array and assign that empty array into a local variable named \`members\`. \`members\` is an array that we will populated with the hashes we will create later; until then we will write a TODO for us.

Then we will assign that array into the \`node.default['haproxy']['members']\`.

Slide 18

Populating the Members with Each New Member

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
all_web_nodes = search("node","role:web")

members = []

all_web_nodes.each do |web_node|
  member = {}
  # TODO: add populate the hash with hostname, ipaddress ...
  # TODO: add the hash to the new array of members
  members.push(member)
end

node.default['haproxy']['members'] = members

include_recipe "haproxy::default"
```


So we need to loop through the array of all the web nodes stored in `all_web_nodes`. We do that through a method available on every array object named 'each'. With the each method a block of code is provided -- you see it here from the first 'do' right after the each to the 'end' later in the file.

A block of code is an operation that you want perform on every item in the array. In our case we want to take each of the node objects and convert them into a hash object.

So every member of the array is visited and every member of the array runs through the block of code.

Slide 19

Populating the Hash with Node Details

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
all_web_nodes.each do |web_node|
  member = {
    "hostname" =>
web_node["cloud"]["public_hostname"],
    "ipaddress" => web_node["cloud"]["public_ipv4"],
    "port" => 80,
    "ssl_port" => 80
  }
  members.push(member)
end
```


Between the pipes we see a local variable that we are defining that exists only in the block `web_node`. This local variable, `web_node`, is a name we came up with to refer to each node in our array of `all_web_nodes`.

Each web node in the array is sent through the block. When inside the block of code it is referred to as `web_node`.

Inside the block the first thing that is created is another local variable named `member` which is assigned a hash that contains the web_node's hostname and the web_node's ipaddress.

Then the local variable `member`, which contains that hash is pushed into the array of members. This adds the member to the end of the array.

When we are done looping through every web node the `members` array contains a list of all these hash objects.

Slide 20

The Final Recipe

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
all_web_nodes = search("node","role:web")

members = []

all_web_nodes.each do |web_node|
  member = {
    "hostname" => web_node["cloud"]["public_hostname"],
    "ipaddress" => web_node["cloud"]["public_ipv4"],
    "port" => 80,
    "ssl_port" => 80
  }
  members.push(member)
end

node.default['haproxy']['members'] = members

include_recipe "haproxy::default"
```


This is the complete recipe source code.

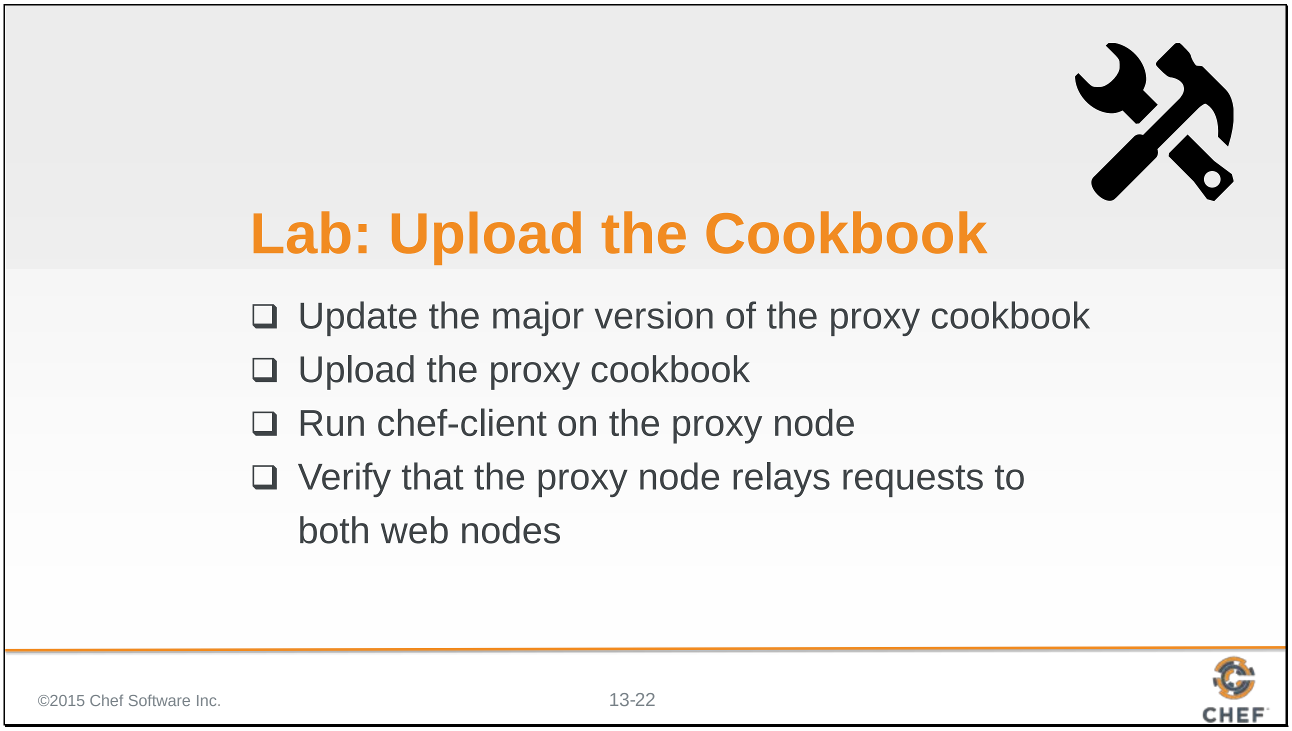
Slide 21

The default recipe of the myhaproxy recipe is now dynamic. Every time a proxy server checks in with the Chef Server, when you run `chef-client`, it will ask the Chef Server if there are any new nodes that are web servers.

As you add nodes, your proxy server will dynamically grow to accommodate them, returning them as node objects, which are then converted to hashes, and then assigned as members.

As you remove nodes, your proxy server will dynamically shrink to accommodate them, returning a smaller set of node objects, which are then converted to hashes, and then assigned as members.

Slide 22



As a lab exercise:

* Update the major version of the cookbook
* Update the cookbook to the Chef Server
* Run `chef-client` on the proxy node
* Verify that the proxy node still relays requests to both of our web servers

Slide 23

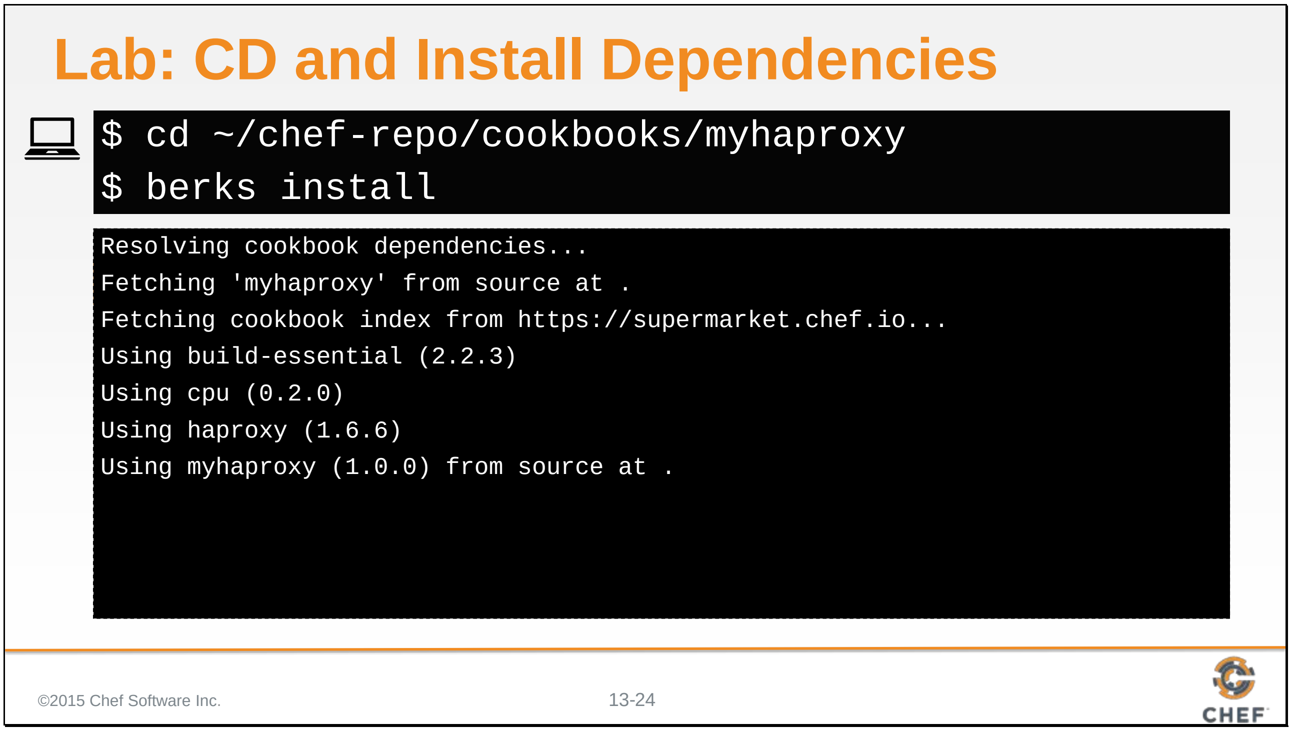
Lab: Update the Version Number

~/chef-repo/cookbooks/myhaproxy/metadata.rb

```
name             'myhaproxy'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures myhaproxy'
long_description 'Installs/Configures myhaproxy'
version          '1.0.0'

depends 'haproxy', '~> 1.6.6'
```


First we update the version to the next major release. We set the version number to 1.0.0.

Slide 24



Change into the cookbook's directory and then install any new dependencies that your cookbook may need at version 1.0.0.

We have no new dependencies but this is required by berkshelf whenever you update the version of the cookbook.

Slide 25

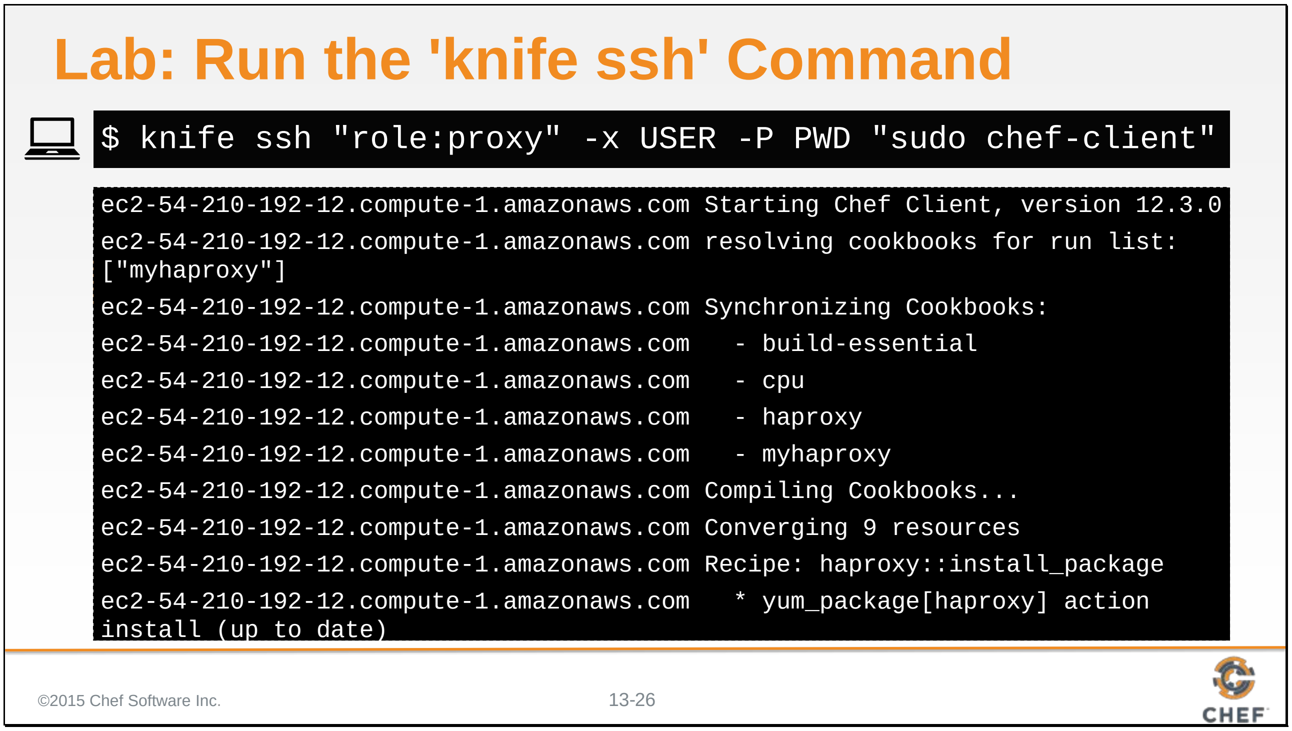
Lab: Upload the Cookbook

```
$ berks upload
```

```
Uploaded build-essential (2.2.3) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded cpu (0.2.0) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded haproxy (1.6.6) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded myhaproxy (1.0.0) to:
'https://api.opscode.com:443/organizations/steveessentials2'
PS C:\Users\sdelfante\chef-repo\cookbooks\myhaproxy>
```


Upload the cookbook using the `berks upload` command.

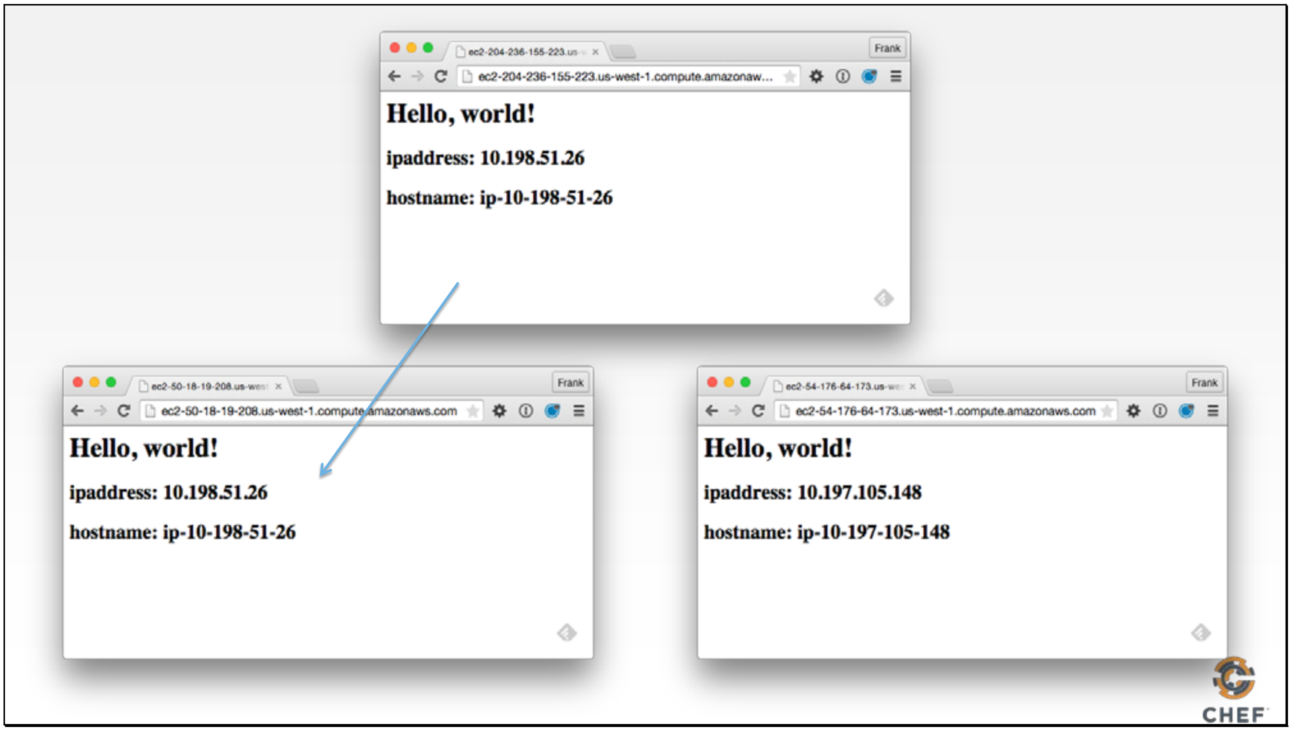
Slide 26



Use `knife ssh` and ask only the nodes with the role 'proxy' to run `sudo chef-client`. This is more efficient than targeted all of the nodes as we did before and more accurate than targeting the node2 "name:node2".

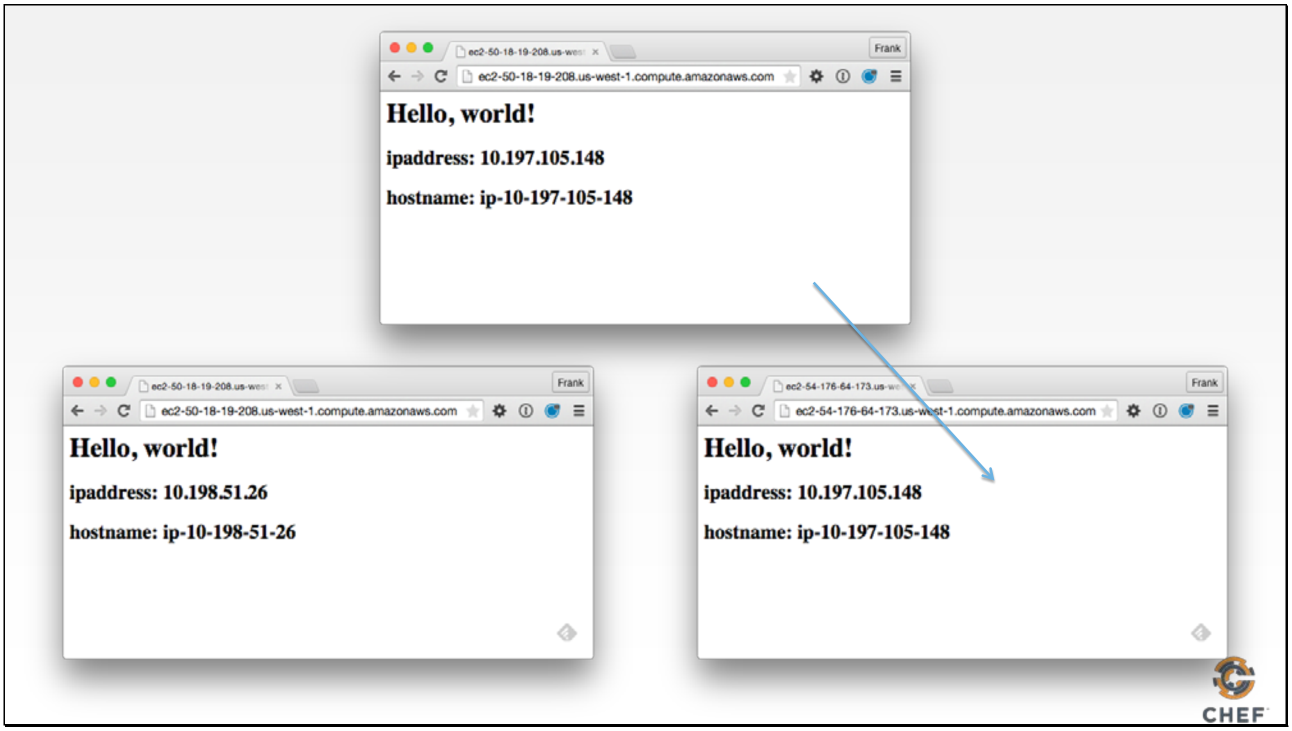
This ensures that all nodes that are also proxy servers to check in with the Chef Server. Similar to how we are targeting only the web server nodes in the recipe.

Slide 27

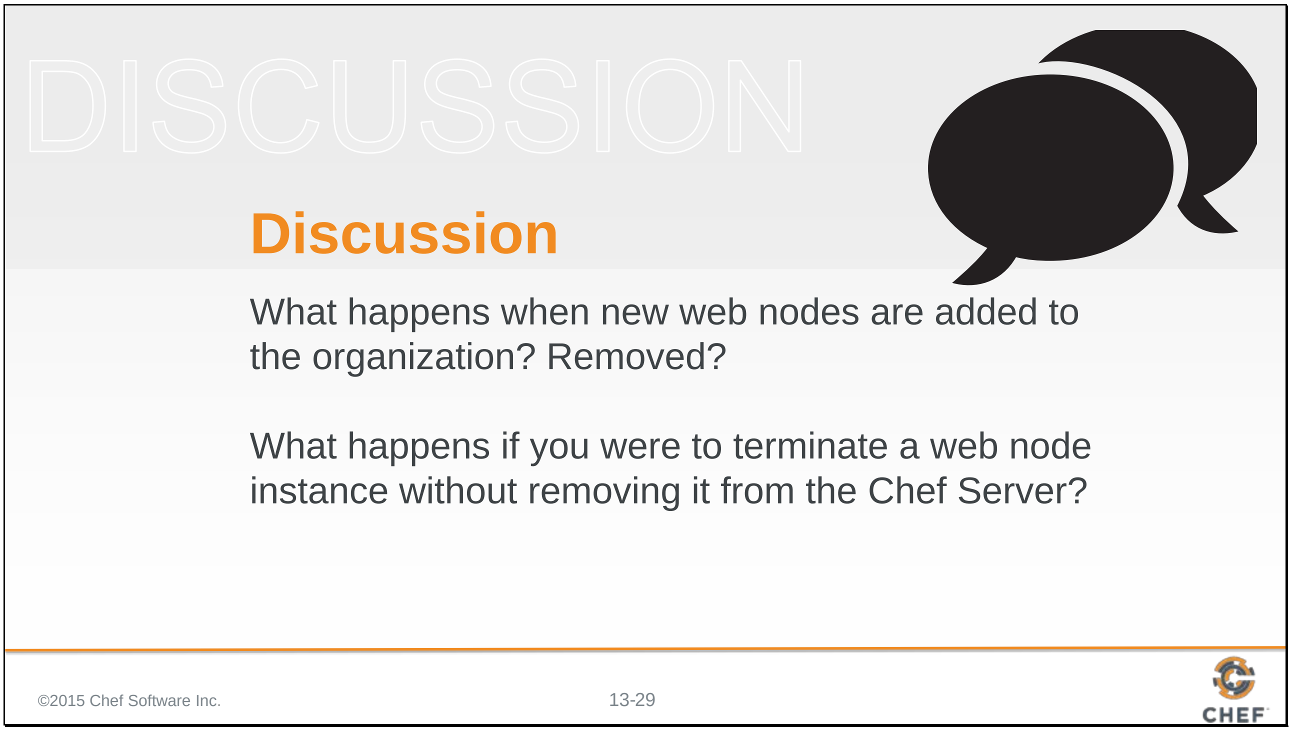


Nothing should change externally. You may see some differences in the logs as the proxy configuration file might change the order of the two entries but the end results is that our proxy server node is still delivering traffic to our two web server nodes.

Slide 28


ec2-50-18-19-208.us-west-1.compute.amazonaws.com
Hello, world!
ipaddress: 10.197.105.148
hostname: ip-10-197-105-148
ec2-50-18-19-208.us-west-1.compute.amazonaws.com
Hello, world!
ipaddress: 10.198.51.26
hostname: ip-10-198-51-26
ec2-54-176-64-173.us-west-1.compute.amazonaws.com
Hello, world!
ipaddress: 10.197.105.148
hostname: ip-10-197-105-148
CHEF

Slide 29



Answer these questions.

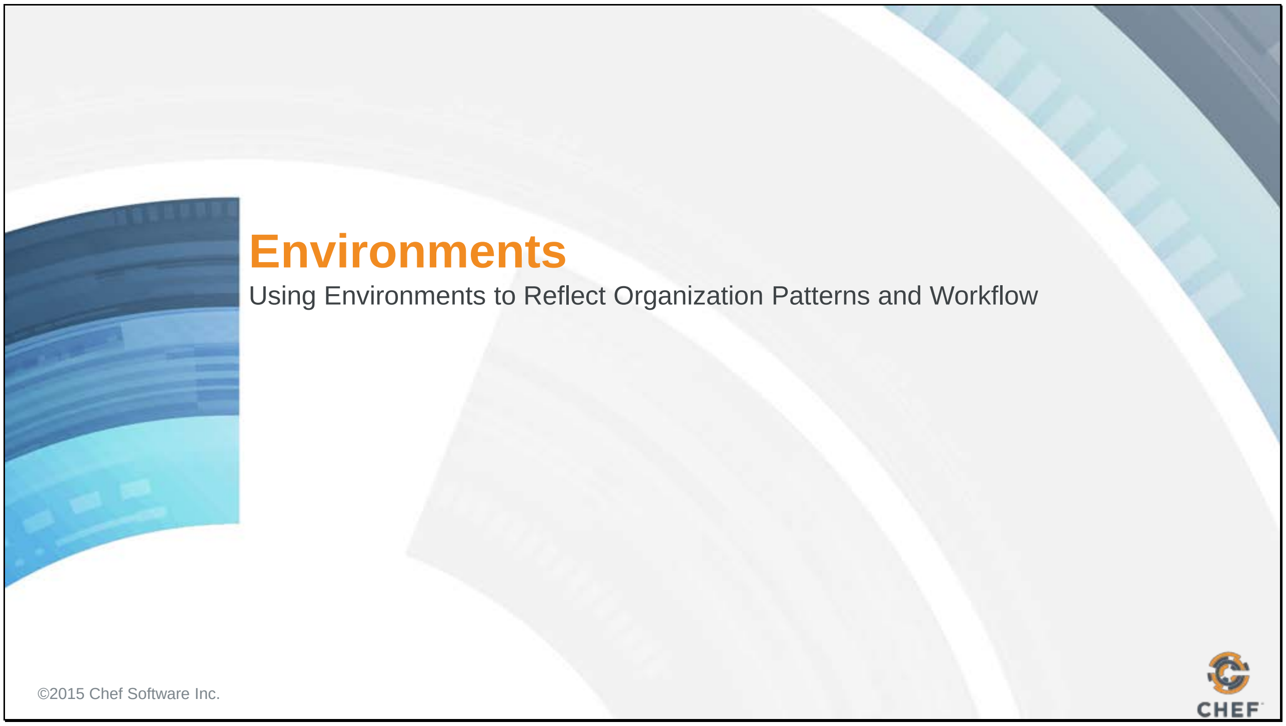
With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 30

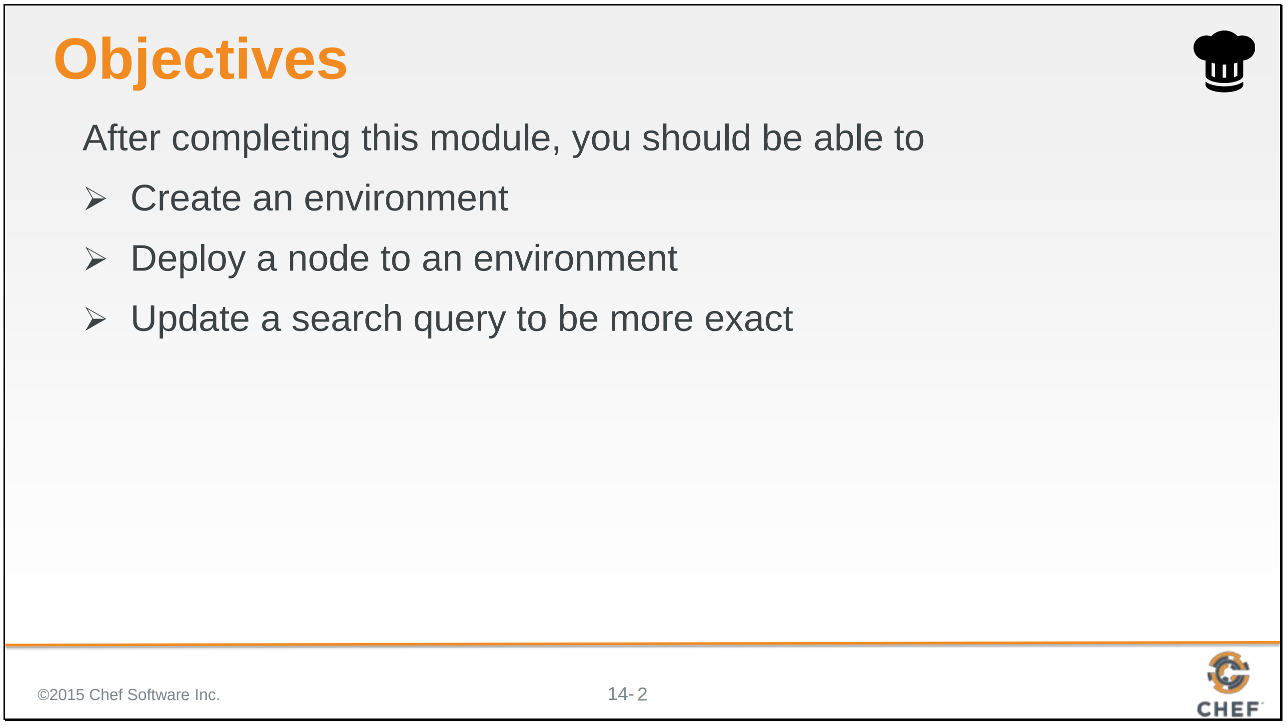

DISCUSSION
Q&A
What questions can we help you answer?
©2015 Chef Software Inc.
13-30
CHEF

Slide 31

14: Environments

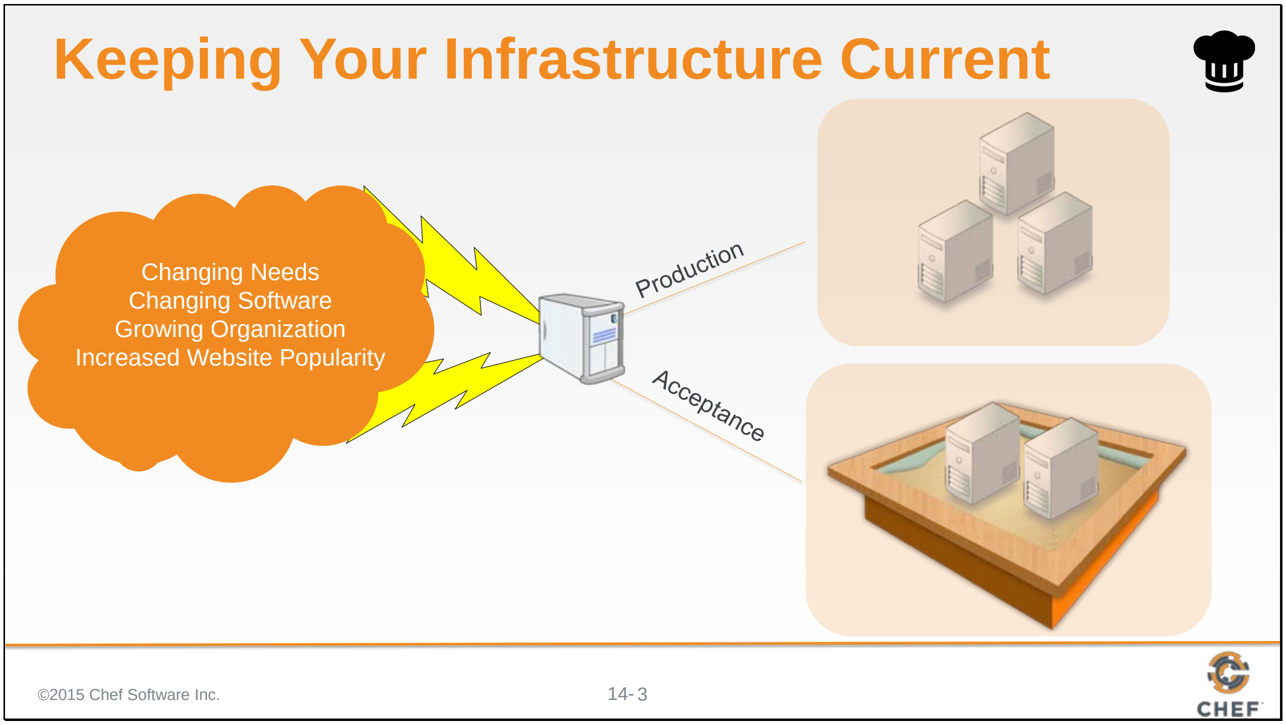


Slide 2



In this section, you will learn how to create an environment, deploy a node to an environment, and update a search query to be more exact.

Slide 3



So, we have updated our myhaproxy cookbook to dynamically search for and update nodes. Everything is as it should be.

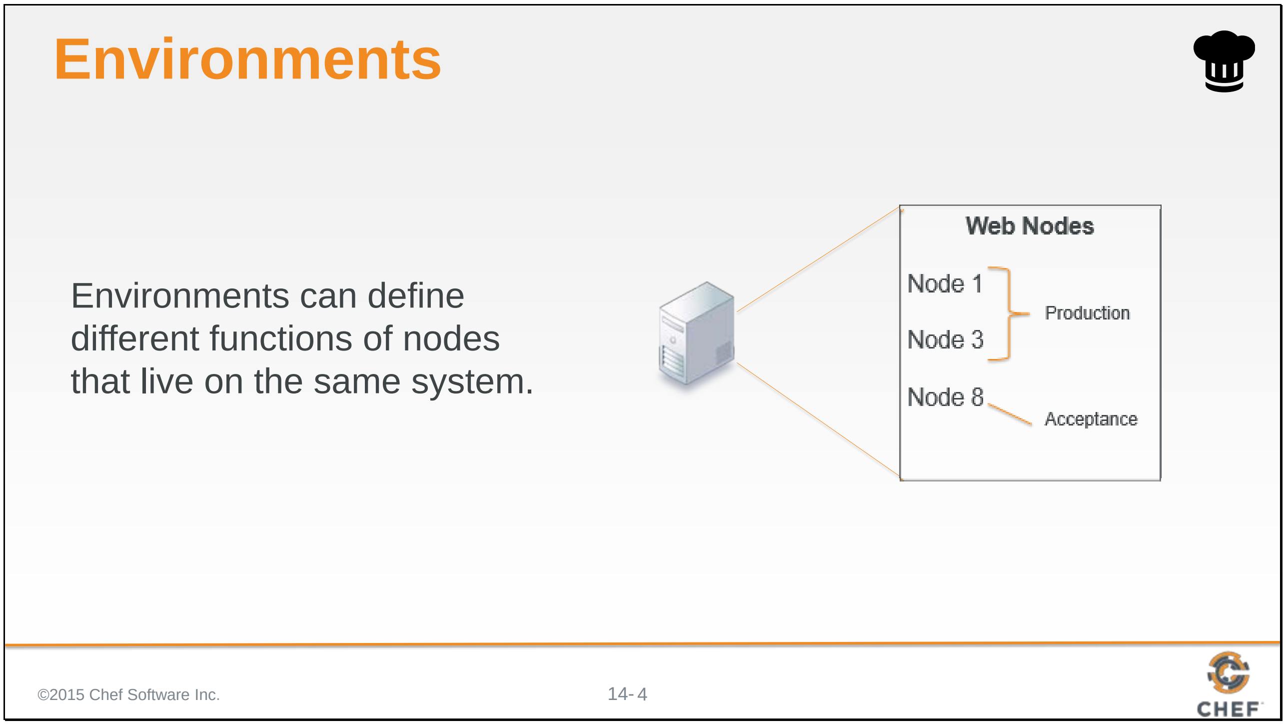
But our system is like a living, breathing thing that must grow and be updated to fit our changing needs. We need to find a way to update and test new tools, features and settings without impacting our current production system.

Of course, we have local testing tools like Test Kitchen to help us verify that our individual cookbooks work before we upload them to the Chef Server. But, that is not always enough. We may want to build, test, and release new features to our cookbooks but we do not immediately want all of our nodes to immediately use them.

For example, what if we had a requirement to update our apache cookbook with a new front page for our application? The release date of our new service with the sign up page does not go live for a week. So, we want to build, test, and upload that cookbook to the Chef Server without actually applying the cookbook until the release date. How would we accomplish that?

This is where environments are useful.

Slide 4



You likely are familiar with the concept of environments. An environment can best be defined as a logical separation of nodes that most often describe the life-cycle of an application.

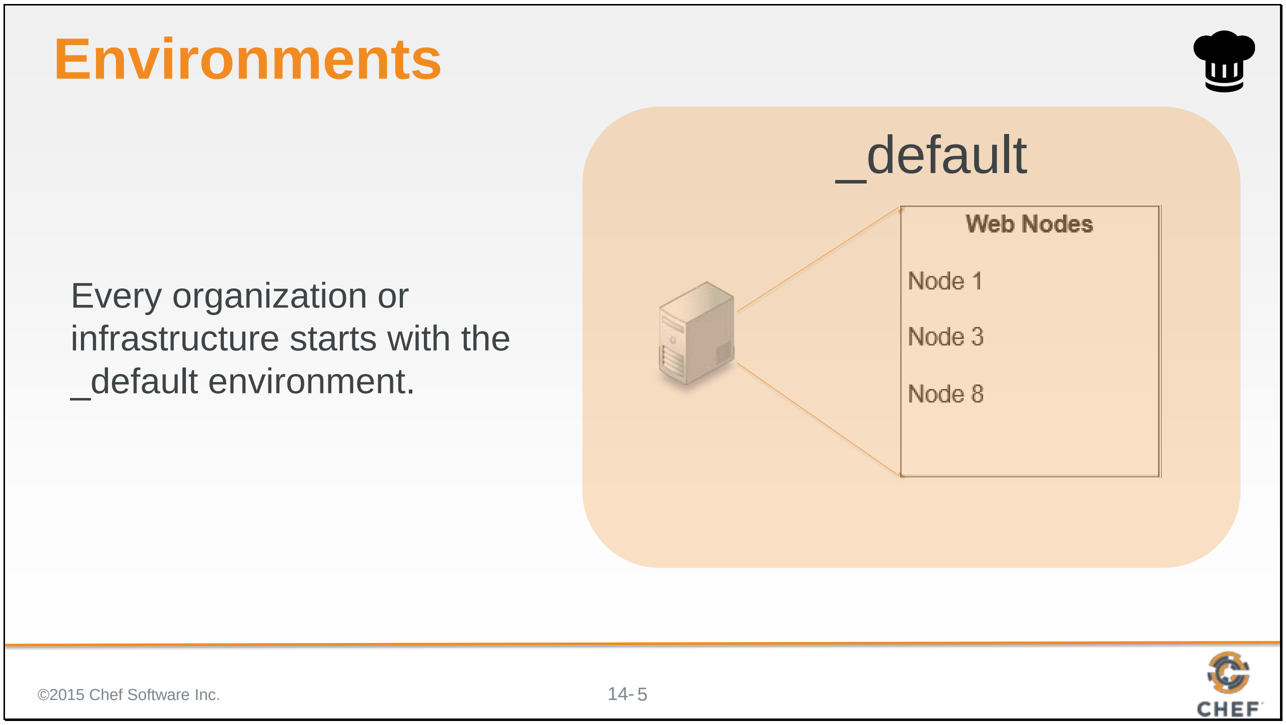
Each environment signifies different behaviors and policies to which a node adheres for a given application or platform. For example, environments can be separated into 'acceptance' and 'production'.

"Acceptance" would be where we may make allowances for constant change and updates and for applications to be deployed with each release.

"Production" might be where we lock down our infrastructure and policies. Production would be what the outside world sees, and would remain unaffected by changes and upgrades until you specifically release them.

Chef also has a concept of an environment. A Chef environment allows us to define a list of policies that we will allow by defining a cookbook.

Slide 5

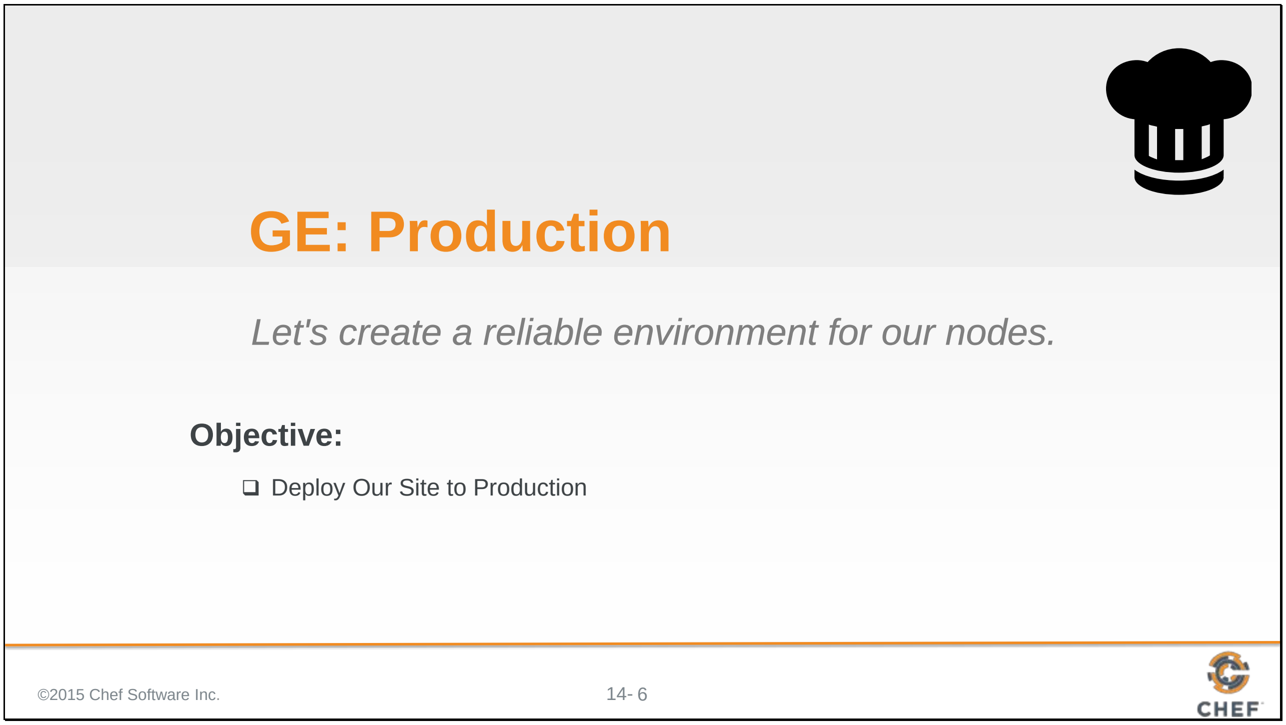


Chef also has a concept of an environment. Chef uses environments to map an organizations real-life workflow to what can be configured and managed using the Chef server.

Every organization begins with a single environment called the _default (underscore default) environment, which cannot be modified or deleted.

Therefore, you must create custom environments to define your organization’s workflow.

Slide 6



First, we need to create a Production environment. This is where we lock down our infrastructure and policies to a specific version of the myhaproxy cookbook.

Slide 7

GE: Using 'knife environment --help'

```
$ cd ~/chef-repo
$ knife environment --help
```

```
** ENVIRONMENT COMMANDS **
knife environment compare [ENVIRONMENT..] (options)
knife environment create ENVIRONMENT (options)
knife environment delete ENVIRONMENT (options)
knife environment edit ENVIRONMENT (options)
knife environment from file FILE [FILE..] (options)
knife environment list (options)
knife environment show ENVIRONMENT (options)
```


Because we still are communicating with the Chef server, let's ask Chef for help regarding available environment commands.

So change into chef-repo and then run 'knife environment --help'.

Slide 8

Remember, we use 'list' to view existing environments.

As previously stated, we see the _default environment has already been created.

Slide 9

GE: Viewing the _default Environment

```
$ knife environment show _default
```

```
chef_type:            environment
cookbook_versions:
default_attributes:
description:          The default Chef environment
json_class:           Chef::Environment
name:                 _default
override_attributes:
```


Let’s see how this environment looks.

Slide 10

GE: Searching All of Our Nodes

```
$ knife search node "*:*"
```

```
3 items found

Node Name:   node1
Environment: _default
FQDN:        ip-172-31-8-68.ec2.internal
IP:          54.175.46.24
Run List:    role[web]
Roles:       web
Recipes:     apache, apache::default, apache::server
Platform:    centos 6.7
Tags:
```


If we search our nodes, we see that all three nodes have been set to the _default environment. How do we change this?

Slide 11

GE: Create an environments Directory

```
$ mkdir environments
```


First, we need to make a new environments directory. (Be sure you are still in the chef-repo before you do this.)

Slide 12

GE: Create the production.rb

~/chef-repo/environments/production.rb

```
name "production"
description "Where we run production code"

cookbook "apache", "= 0.2.1"
cookbook "myhaproxy", "= 1.0.0"
```


Then we need to create a production.rb file. Like in the roles.rb files, we must provide a name and description.

Additionally, we need to define cookbook restrictions to lock down specific versions of both the apache and myhaproxy cookbooks. By adding this information to production.rb, we are telling our nodes to use these specific versions of these specific cookbooks.

Obviously, what this means is that as we work on newer versions of these cookbooks, we won’t break anything in the production environment.

Okay, so now that we have captured our “good” environment in this file, let’s save it and upload it.

Slide 13

GE: Upload the production.rb File

```
$ knife environment from file production.rb
```

```
Updated Environment production
```


Using the knife environment command, let's upload the production.rb file. This should be familiar because it is just like the command we used to upload roles.

Slide 14

Okay, let’s use our list command to make sure the file uploaded correctly.

Slide 15

GE: View the Production Environment

```
$ knife environment show production
```

```
chef_type:           environment
cookbook_versions:
  apache:     = 0.2.1
  myhaproxy: = 1.0.0
default_attributes:
description:         Where we run production code
json_class:          Chef::Environment
name:                production
override_attributes:
```


CHEF

If we use the knife environment show command, we can see how the production.rb file looks.

Note the cookbook versions that we set are shown here.

Slide 16

GE: Viewing 'knife node --help'

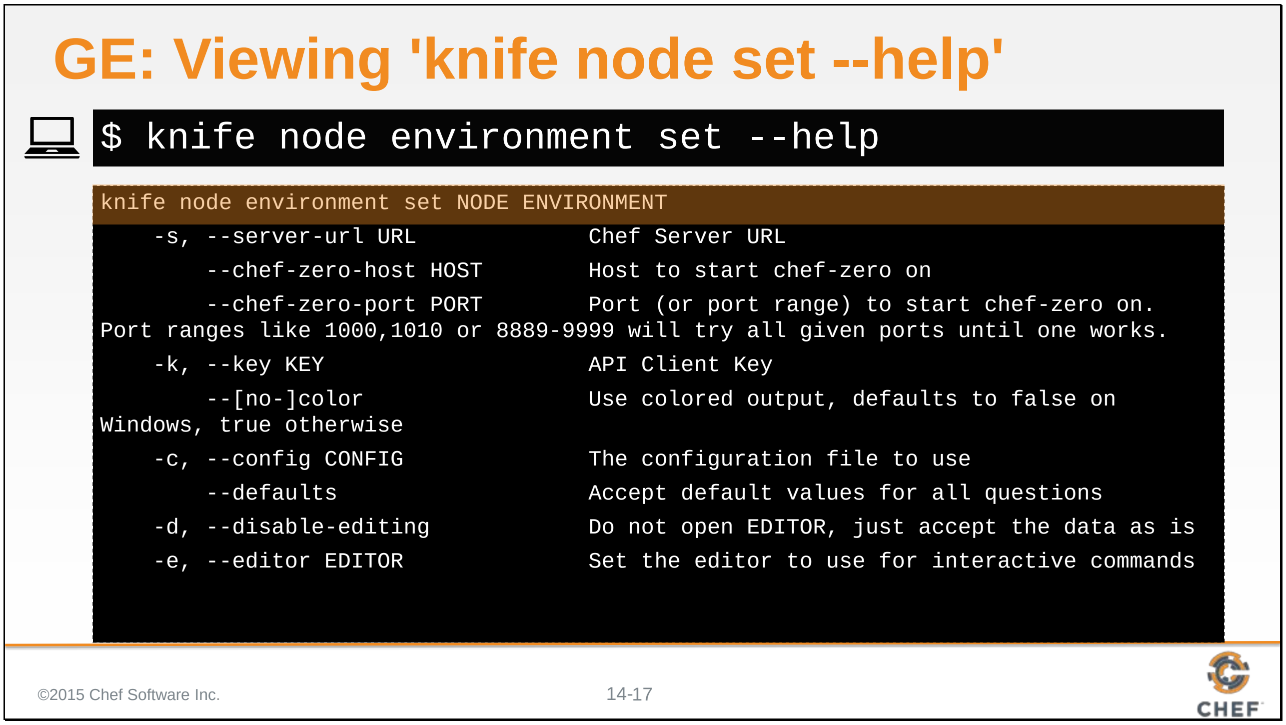
```
$ knife node --help
```

```
** NODE COMMANDS **
knife node bulk delete REGEX (options)
knife node create NODE (options)
knife node delete NODE (options)
knife node edit NODE (options)
knife node environment set NODE ENVIRONMENT
knife node from file FILE (options)
knife node list (options)
knife node run_list add [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list remove [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list set NODE ENTRIES (options)
knife node show NODE (options)
```


Now, we need to set the environments for our nodes. Let’s ask Chef for help on that as well.

Let’s use the knife node environment set command.

Slide 17



But how does that command work, exactly?

It looks like we just add the environment name at the end of the command to set that environment on a node.

Slide 18

GE: Using 'knife environment node set'

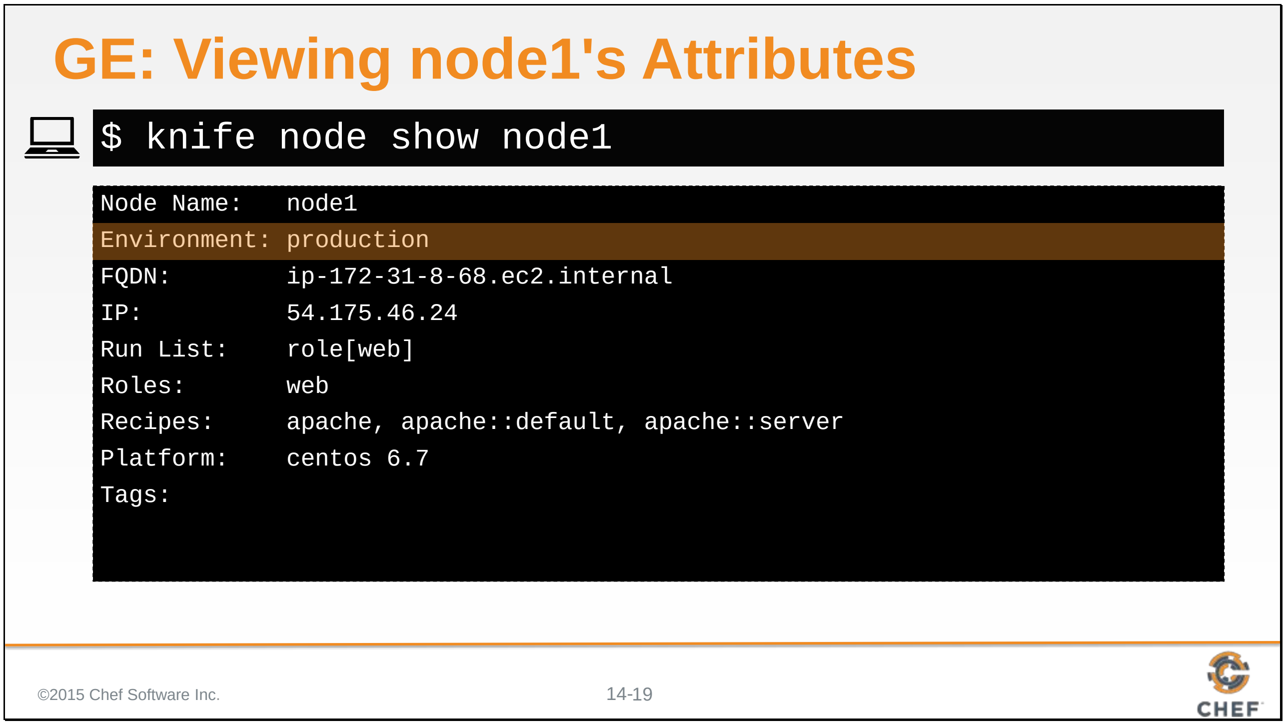
```
$ knife node environment set node1 production
```

```
node1:
  chef_environment:
```


So, let’s do that for node1.

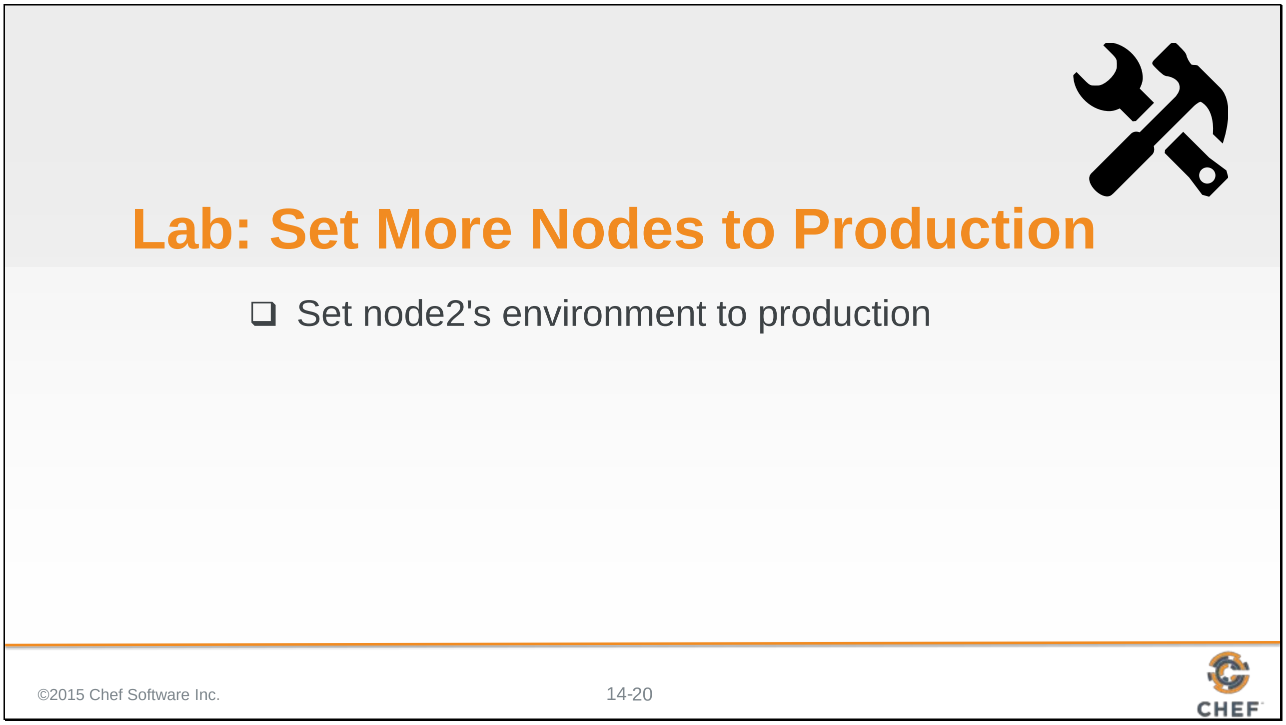
The results don’t really tell us much, so let’s take a look at node1.

Slide 19



Using knife node show, we can see node1’s attributes. Note that it has indeed been set to the production environment.

Slide 20



Let’s do the same thing for node2.

Slide 21

Lab: Set node2's Environment to Production

```
$ knife node environment set node2 production
```

```
node2:
  chef_environment:
```

Slide 22

Lab: Verify node2 is Set to Production

```
$ knife node show node2
```

```
Node Name:   node2
Environment: production
FQDN:        ip-172-31-0-128.ec2.internal
IP:          54.210.192.12
Run List:    role[proxy]
Roles:       proxy
Recipes:     myhaproxy, myhaproxy::default, haproxy::default,
haproxy::install_package
Platform:    centos 6.6
Tags:
```


And, it looks like node2 was successfully set to the production environment.

Slide 23

Production

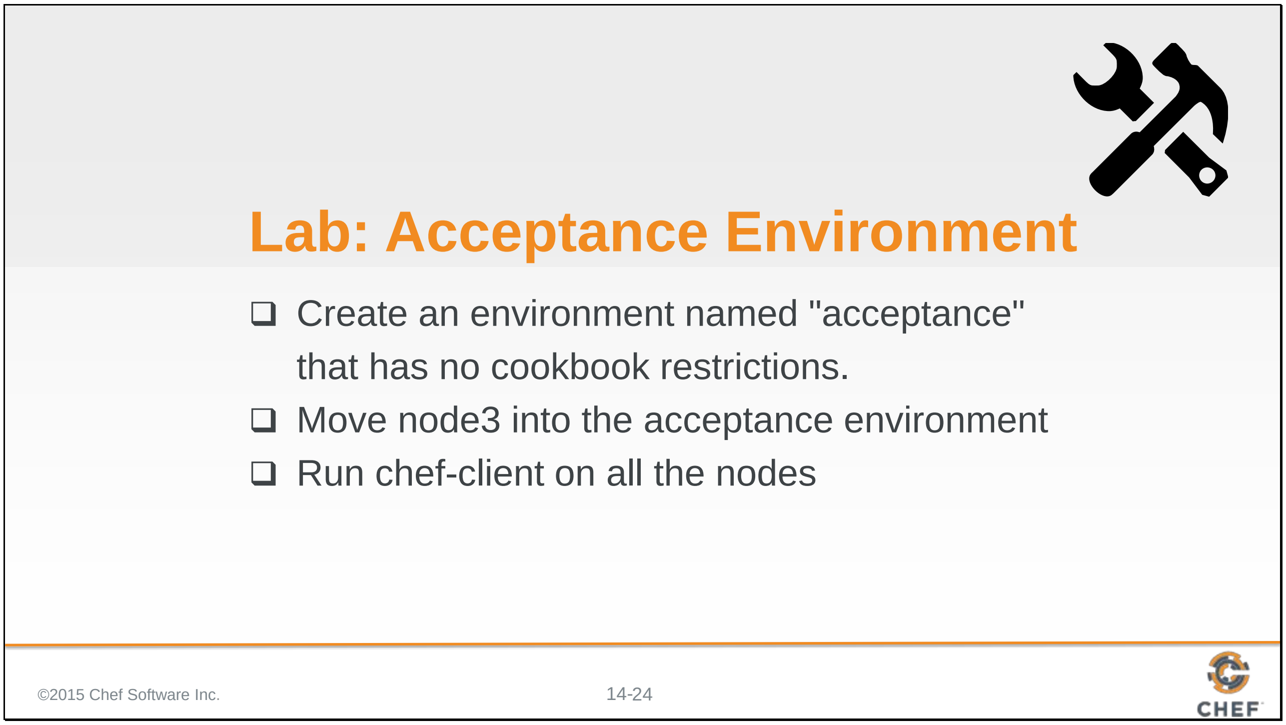
Let's create a reliable environment for our nodes.

Objective:

✓ Deploy our site to Production

CHEF

Slide 24



Now, let’s create the environment we can use to change and update the cookbooks without affecting our production environment. A sandbox, if you will.

Let’s call this our “Acceptance environment”.

Slide 25

Lab: Create a New Environment File

~/chef-repo/environments/acceptance.rb

```
name "acceptance"
description "Where code and applications are
tested"
# No Cookbook Restrictions
```


First, let’s create a new rb file in our chef-repo/environments directory. Let’s name it acceptance.

In the Acceptance environment, we don’t want to lock-down the cookbook versions, so we are not going to place restrictions on the cookbooks.

Slide 26

Let’s upload that .rb file to the Chef server.

Slide 27

Lab: Verify that the Environment was Set

```
$ knife environment list
```

```
_default
production
acceptance
```


And let's make sure that this environment file was added properly.

Slide 28

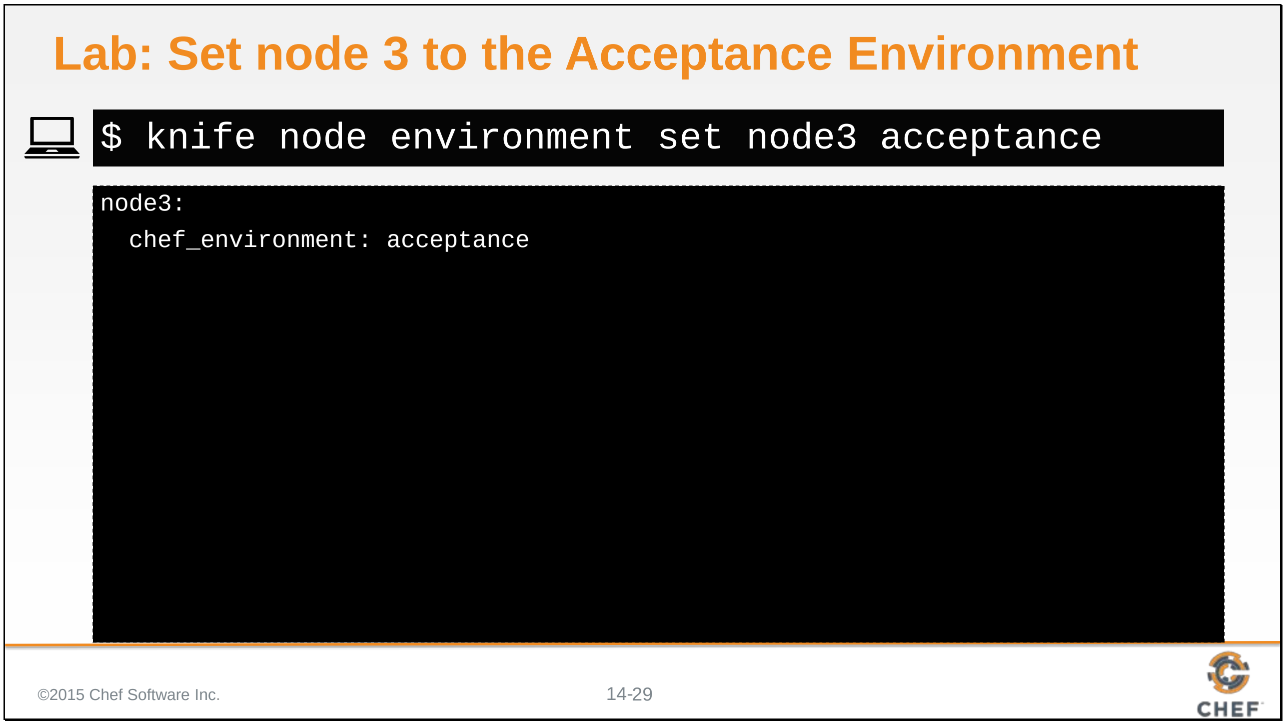
Lab: Verify the Contents of the Environment

```
$ knife environment show acceptance
```

```
chef_type:            environment
cookbook_versions:
default_attributes:
description:          Where code and applications are tested
json_class:           Chef::Environment
name:                 acceptance
override_attributes:
```


And last, but not least, let’s ask the Chef Server to show us the acceptance environment.

Slide 29



Okay, let's set node3 to the acceptance environment.

Slide 30

Lab: Verify that the Environment Was Set

```
$ knife node show node3
```

```
Node Name:   node3
Environment: acceptance
FQDN:        ip-172-31-0-127.ec2.internal
IP:          54.210.86.164
Run List:    role[web]
Roles:       web
Recipes:     apache, apache::default, apache::server
Platform:    centos 6.6
Tags:
```


And confirm that it has been set properly.

Slide 31

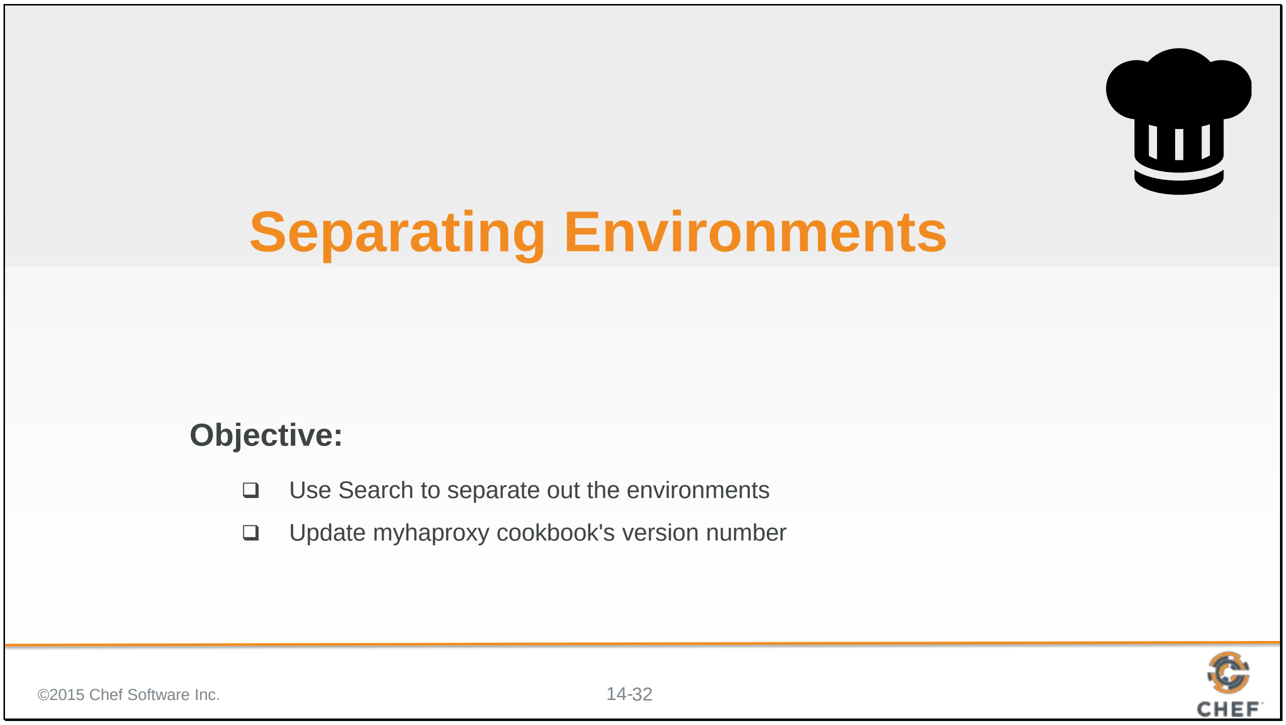
Lab: Converge All the Nodes

```
$ knife ssh "*:*" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-175-46-24.compute-1.amazonaws.com  Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com resolving cookbooks for run list: ["apache"]
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-54-210-86-164.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-86-164.compute-1.amazonaws.com   - apache
ec2-54-210-86-164.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-86-164.compute-1.amazonaws.com Converging 3 resources
ec2-54-210-86-164.compute-1.amazonaws.com Recipe: apache::server
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com   - build-essential
```


Using the knife ssh let’s run chef client on all the nodes.

Slide 32



Now that we have created our two environments and set each node to a specific environment, we need to separate the environments to ensure that the proxy server only communicates with the production nodes.

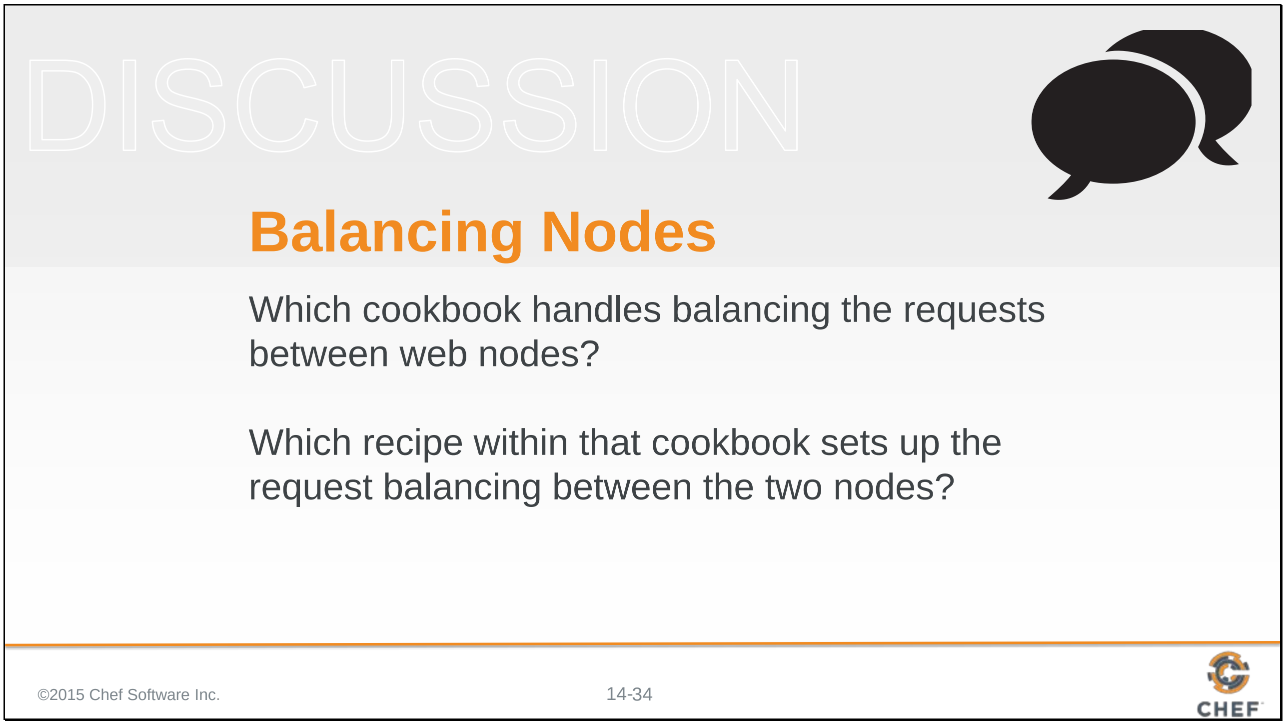
Slide 33

So we set our web nodes to specific environments. As we manage our nodes, making changes to our cookbooks and recipes, what do you think is going to happen to Node1?

What about Node 3?

Setting the nodes is not enough. Chef does not automatically know to separate the environments. So, we have to tell it how to do that.

Slide 34



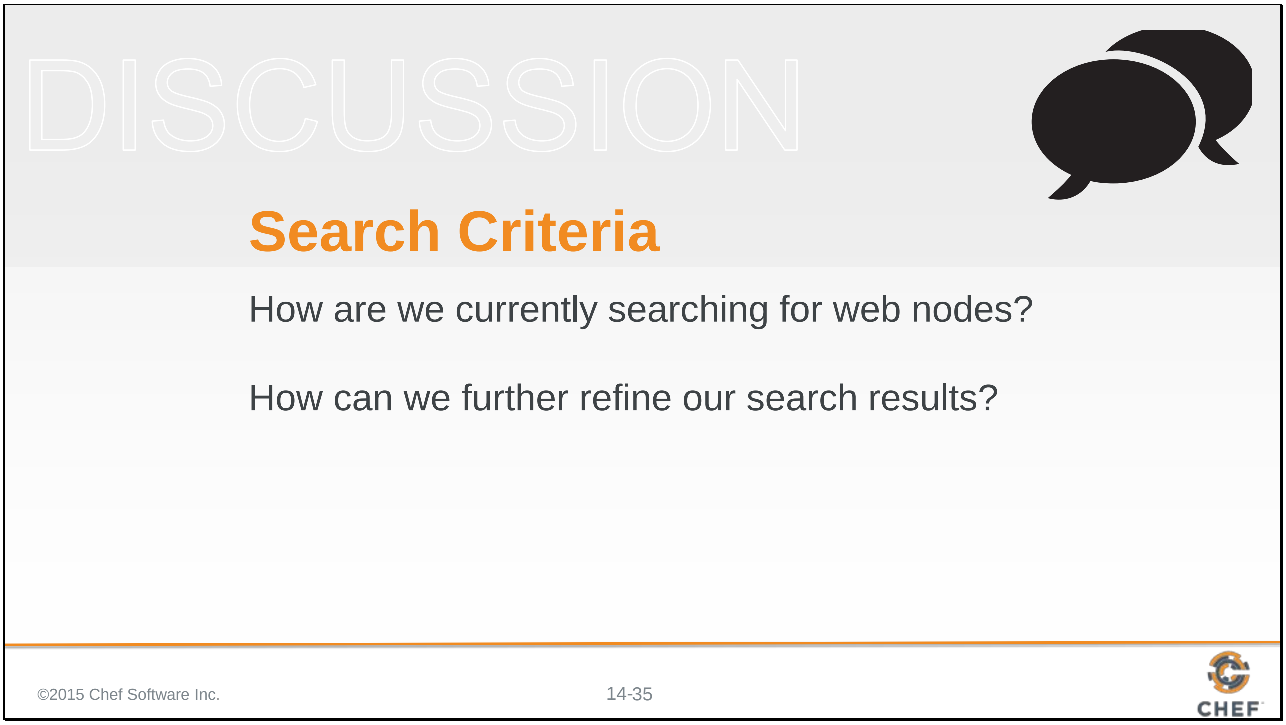
How do we do that?

First, let's answer a couple of question. As you think about the infrastructure we have created, which cookbook handles balance requests between nodes?

So if we want to make changes to that cookbook, which recipe would we change?

Answer 1: myhaproxy Answer 2: default.rb

Slide 35



In our last module, we talked about searching our nodes using Chef. Do you recall what we used to search for web nodes?

Answer: all_web_nodes = search(“node”,”role:web”)

So, considering our search syntax, how can we further refine that syntax to search for a specific web node by environment?

Let’s take a look.

Slide 36

Search Criteria

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
#
# Cookbook Name:: myhaproxy
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.

all_web_nodes = search("node","role:web")

members = []

#...
```


Looking at the default.rb file in the myhaproxy cookbook, we can review the original search syntax. If we want to search by environments, what would we need to add here?

Slide 37

GE: Modify the myhaproxy default.rb

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
#
# Cookbook Name:: myhaproxy
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.

all_web_nodes = search("node","role:web AND
chef_environment:#{node.chef_environment}")

members = []

#...
```


Search the Chef Server for all node objects that have the role equal to 'web' and also share the same environment as the current node applying this recipe. The nodes currently applying this recipe are the nodes with the role set to proxy.

Now that we’ve made our changes, let’s save this file.

Slide 38

GE: Separate Environments

- ✓ Use Search to separate out the environments
- ❑ Update myhaproxy version number

 38 14-

Now that we have created our two environments and set each node to a specific environment, we need to separate the environments to ensure that the proxy server only communicates with the production nodes.

Slide 39

GE: Version the myhaproxy metadata.rb

~/chef-repo/cookbooks/myhaproxy/metadata.rb

```
name             'myhaproxy'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures myhaproxy'
long_description 'Installs/Configures myhaproxy'
version          '1.0.1'

depends 'haproxy', '~> 1.6.6'
```


Before we upload the new myhaproxy cookbook to the server, we probably want to update the version number. What type of change have we made here?

Answer: Patch

Because we are performing a patch, let's set the version number to 1.0.1.

Slide 40

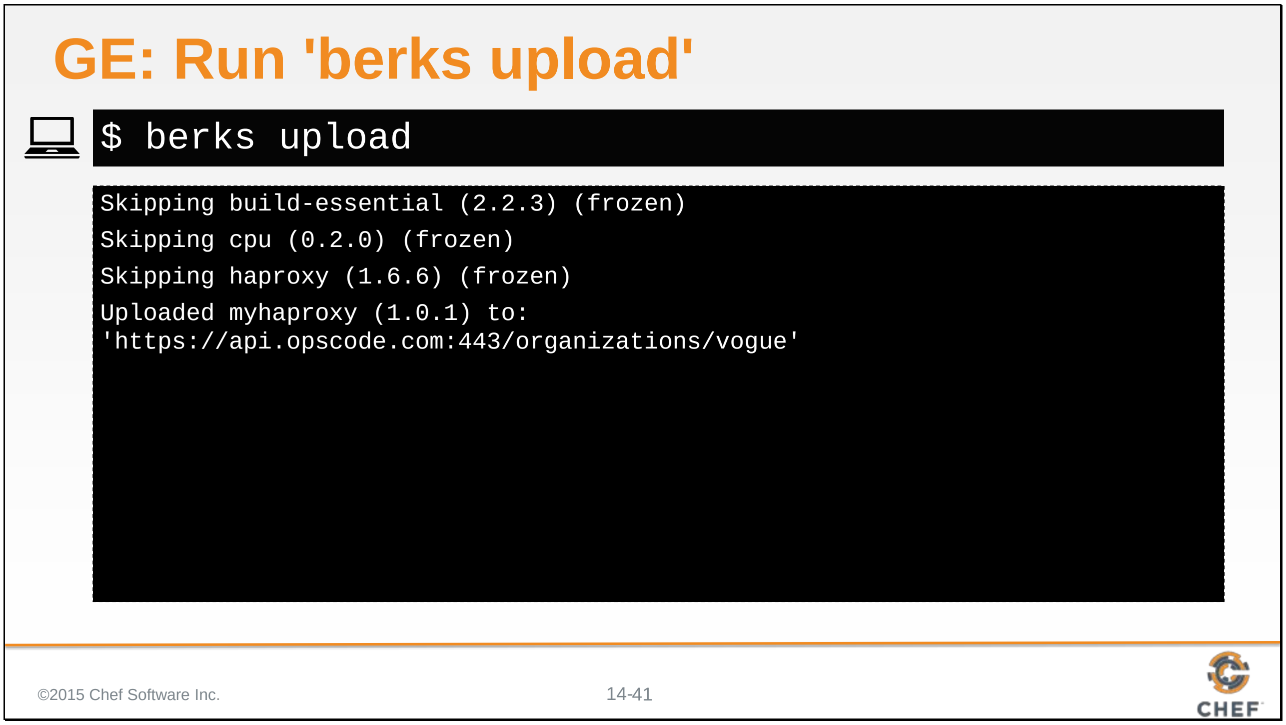
GE: Run 'berks install'

```
$ cd cookbooks/myhaproxy
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'myhaproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using build-essential (2.2.3)
Installing haproxy (1.6.6)
Using cpu (0.2.0)
Using myhaproxy (1.0.1) from source at .
```


We are going to need to use Berks to upload this cookbook because it has dependencies.
So first we need to cd into the cookbook.
Then run 'berks install'.

Slide 41



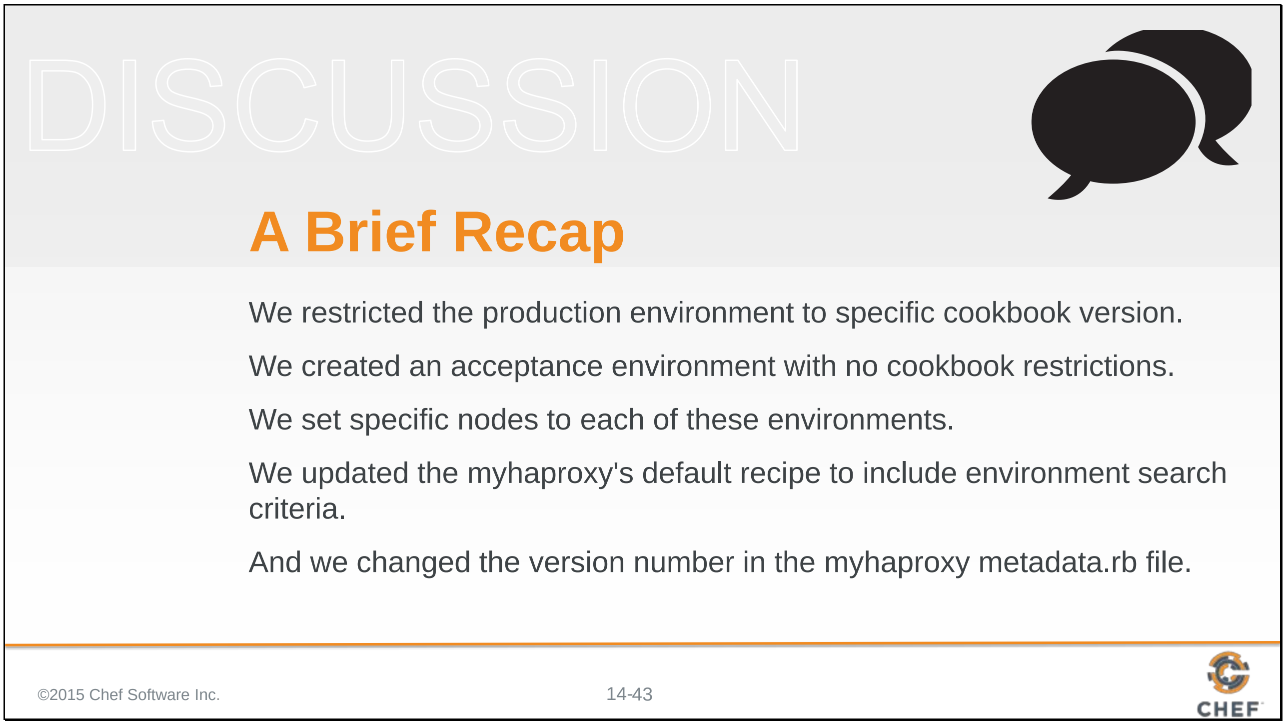
And finally berks upload.

Slide 42

GE: Separate Environments

- ✓ Use Search to separate out the environments
- ✓ Update myhaproxy version number

Slide 43



Before we run 'chef-client' to bring everything up to date, let’s think about what we’ve done.

First, in the production environment, we restricted our cookbooks to a specific version.

Second, we created an acceptance environment with no cookbook restrictions.

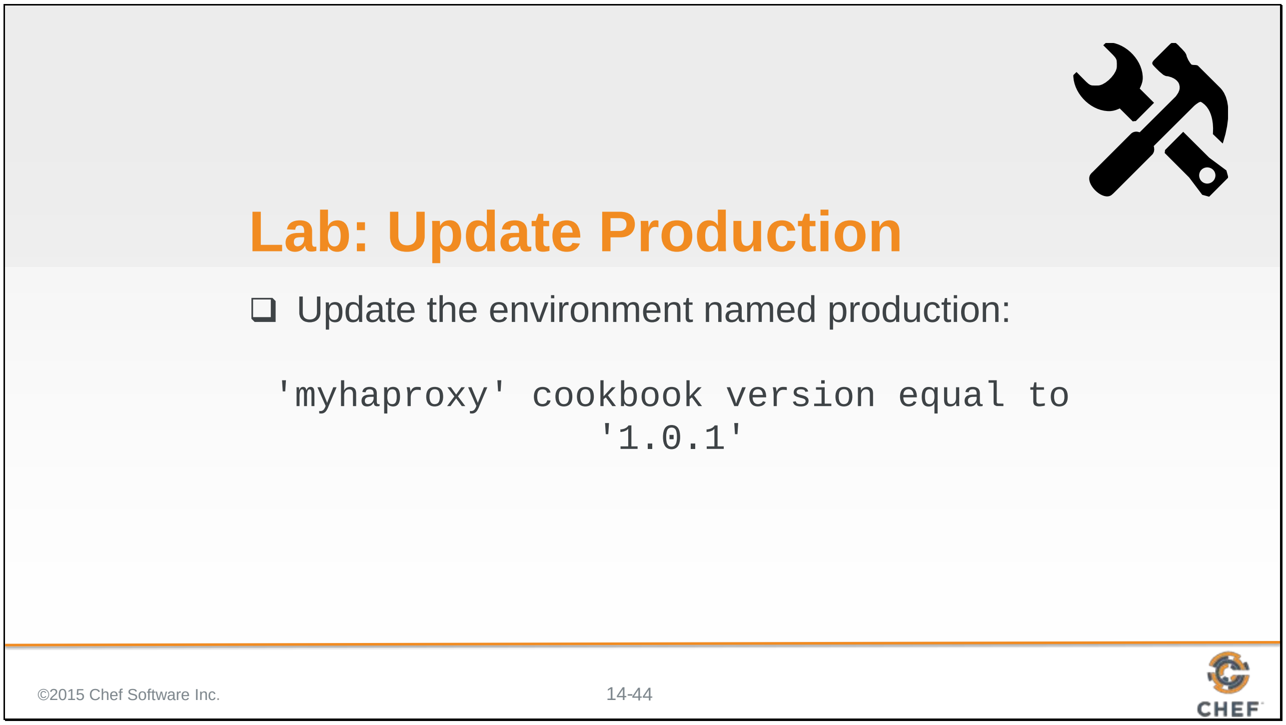
Third, we set specific nodes to each of these environments.

Fourth, we updated the myhaproxy default.rb to include environment search criteria.

And lastly, we changed the version number in the myhaproxy metadata.rb file.

What problems do you think we may encounter, given all that we’ve done here?

Slide 44



Since we changed the version of the myhaproxy cookbook, we need to revise the production.rb file to incorporate the new version.

Slide 45

Lab: Update production.rb

~/chef-repo/environments/production.rb

```
name "production"
description "Where we run production code"

cookbook "apache", "= 0.2.1"
cookbook "myhaproxy", "= 1.0.1"
```


So let’s go back into our production.rb and update it to include the new version number.

Slide 46

Change to ~/chef-repo and then run 'knife environment from file production.rb'.

Slide 47

Lab: Verify the Version Number

```
$ knife environment show production
```

```
chef_type:           environment
cookbook_versions:
  apache:    = 0.2.1
  myhaproxy: = 1.0.1
default_attributes:
description:         Where we run production code
json_class:          Chef::Environment
name:                production
override_attributes:
```


And let's make sure that the production.rb on Chef server has the correct version of myhaproxy designated.

Slide 48

Lab: Converge All Nodes

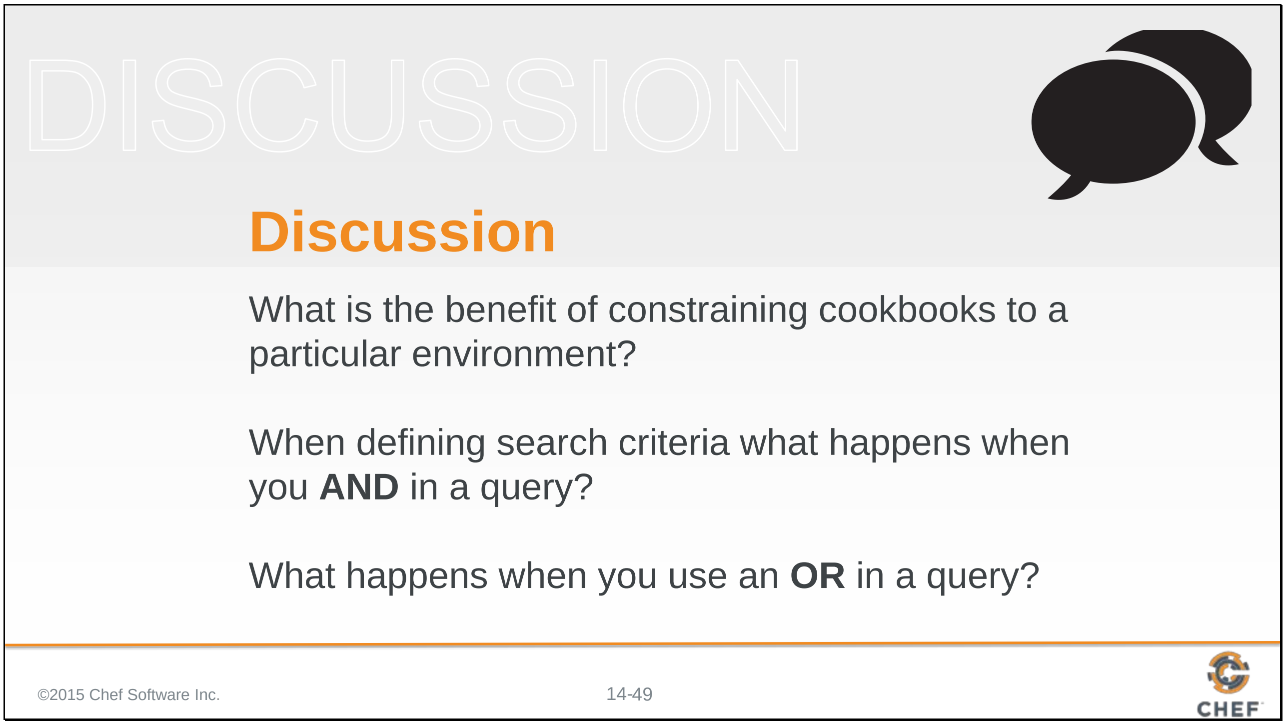
```
$ knife ssh "*:*" -x USER -P PWD "sudo chef-client”
```

```
ec2-54-175-46-24.compute-1.amazonaws.com  Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-86-164.compute-1.amazonaws.com resolving cookbooks for run list: ["apache"]
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-54-210-86-164.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-86-164.compute-1.amazonaws.com   - apache
ec2-54-210-86-164.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-86-164.compute-1.amazonaws.com Converging 3 resources
ec2-54-210-86-164.compute-1.amazonaws.com Recipe: apache::server
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com   - build-essential
```


CHEF

And use 'sudo chef-client' to converge all nodes.

Slide 49



Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 50

DISCUSSION

Q&A

What questions can we help you answer?

CHEF

Slide 51

15: Further Resources

Slide 2

Going Forward

There are many Chef resources available to you outside this class. During this module we will talk about just a few of those resources.

But…remember what we said at the beginning of this class:

The best way to learn Chef is to use Chef

Slide 3


Practice Chef
First, let’s talk about stuff you can read to help you learn Chef.
©2015 Chef Software Inc.
15- 3
CHEF

Slide 4

Another great place to practice Chef is our awesome Learn Chef site. These interactive modules provide you with an opportunity to run through exercises similar to those we did in this class, and it is updated when new Chef features are introduced. This is one of the most robust self-guided tutorial sites out there.

Slide 5

Resources You Can Read

A lot of people in the Chef community have written about Chef.

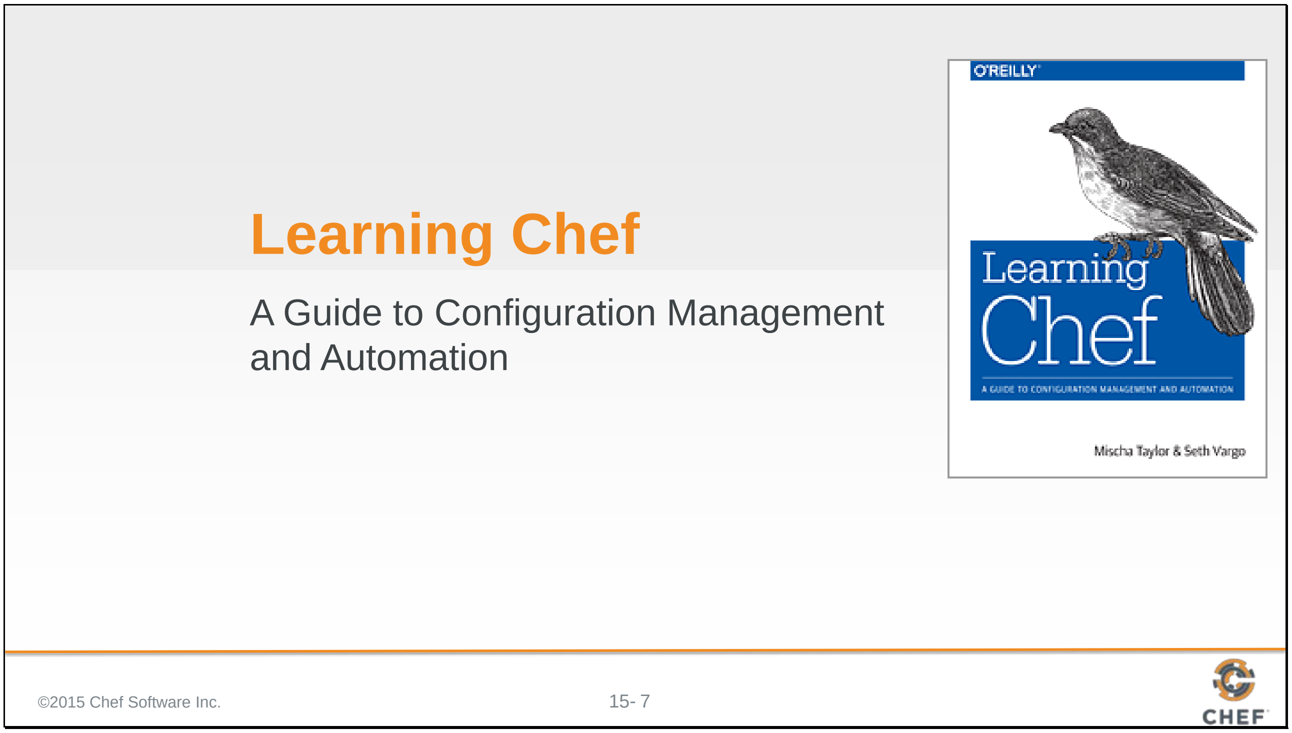
Here are just a few of those resources.

Slide 6

Remember, how often we referred to the Docs site throughout this workshop. That wasn't by accident. We wanted you to become comfortable with using our Docs site to resolve issues and learn about the many Chef tools out there.

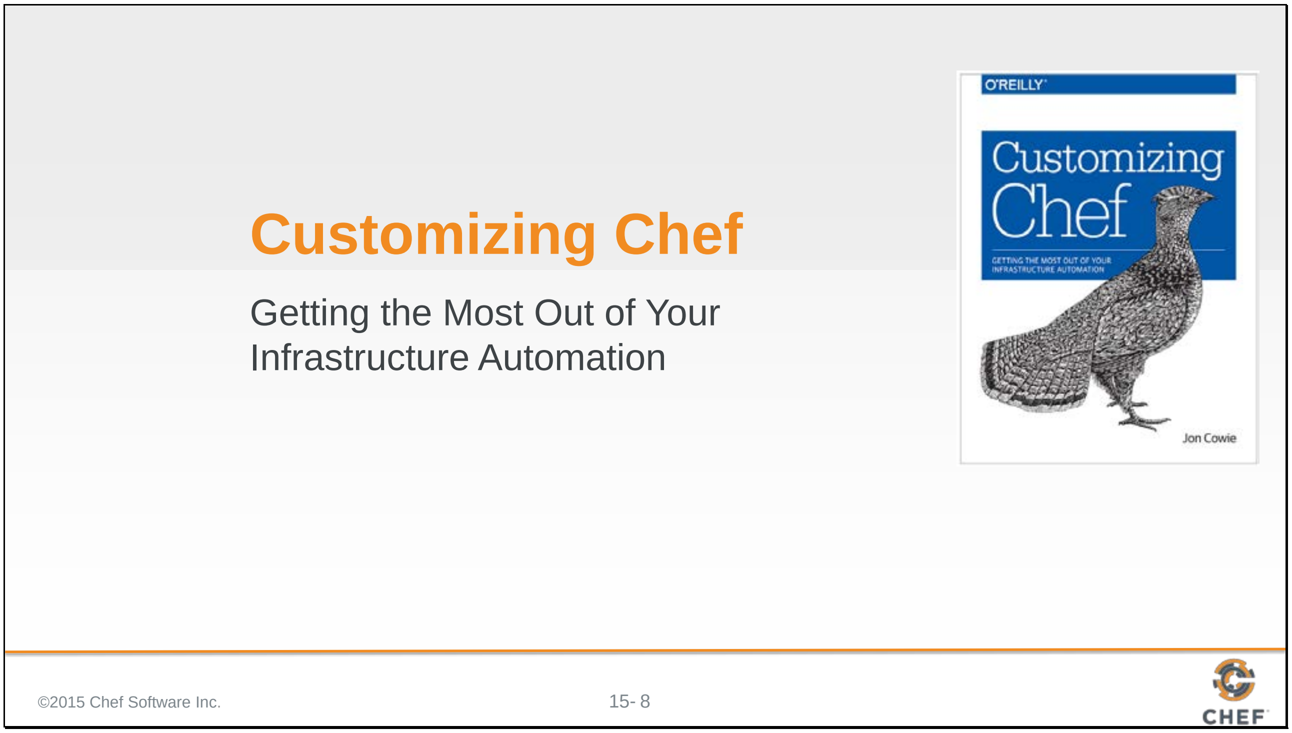
Docs are there, available to you, 24 hours a day, 7 days a week. Any question you have, you probably will find the answer on our Docs site.

Slide 7



Some people who have used Chef for years have written some excellent books about Chef. Check out Mischa Taylor's and Seth Vargo's *Guide to Configuration Management and Automation*. You can find it on O'Reilly. It's a great book.

Slide 8



Additionally, you may want to read Jon Cowie's *Getting the Most Out of Your Infrastructure Automation.* It's also available on O'Reilly.

Slide 9

We have uploaded a number of videos to the Chef YouTube channel, including training videos and talks from past Chef conferences.

Slide 10

theshipshow.com

 15-10 CHEF

The Ship Show is a twice-monthly podcast, featuring discussion on everything from build engineering to DevOps to release management, plus interviews, new tools and techniques, and reviews.

Slide 11

Food Fight is a bi-weekly podcast for the Chef community. We bring together the smartest people in the Chef community and the broader DevOps world to discuss the thorniest issues in system administration.

Slide 12

Chef Developers' IRC Meeting

https://github.com/chef/chef-community-irc-meetings

CHEF

Join members of the Chef Community in a weekly meeting for Chef Developers where we’ll discuss the future of the Chef project and other things pertinent to the community. The agenda and schedule can be found at this link.

Slide 13

ChefConf is a gathering of hundreds of Chef community members. We get together to learn about the latest and greatest in the industry (both the hows and the whys), as well as exchange ideas, brainstorm solutions, and give hugs, which has become the calling card of the DevOps community, and the Chef community in particular.

ChefConf 2016 will be held in Austin, Texas during July.

Slide 14

Chef Community Summit

https://www.chef.io/summit/

 15-14 CHEF

The Chef Community will gather for two days of open space sessions and brainstorm on Chef best practices.

The Chef Community Summit is a facilitated Open Space event. The participants of the summit propose topics, organize an agenda, and discuss and work on the ideas that are most important to the community.

The 2015 summit takes place in two locations–Seattle on October 14 & 15, then in London on November 3 and 4. Dates for the 2016 Summit will be announced in the spring of 2016.

Slide 15

CHEF™

Thank You!