2: Chef Resources

Slide 2

Objectives

After completing this module, you should be able to:

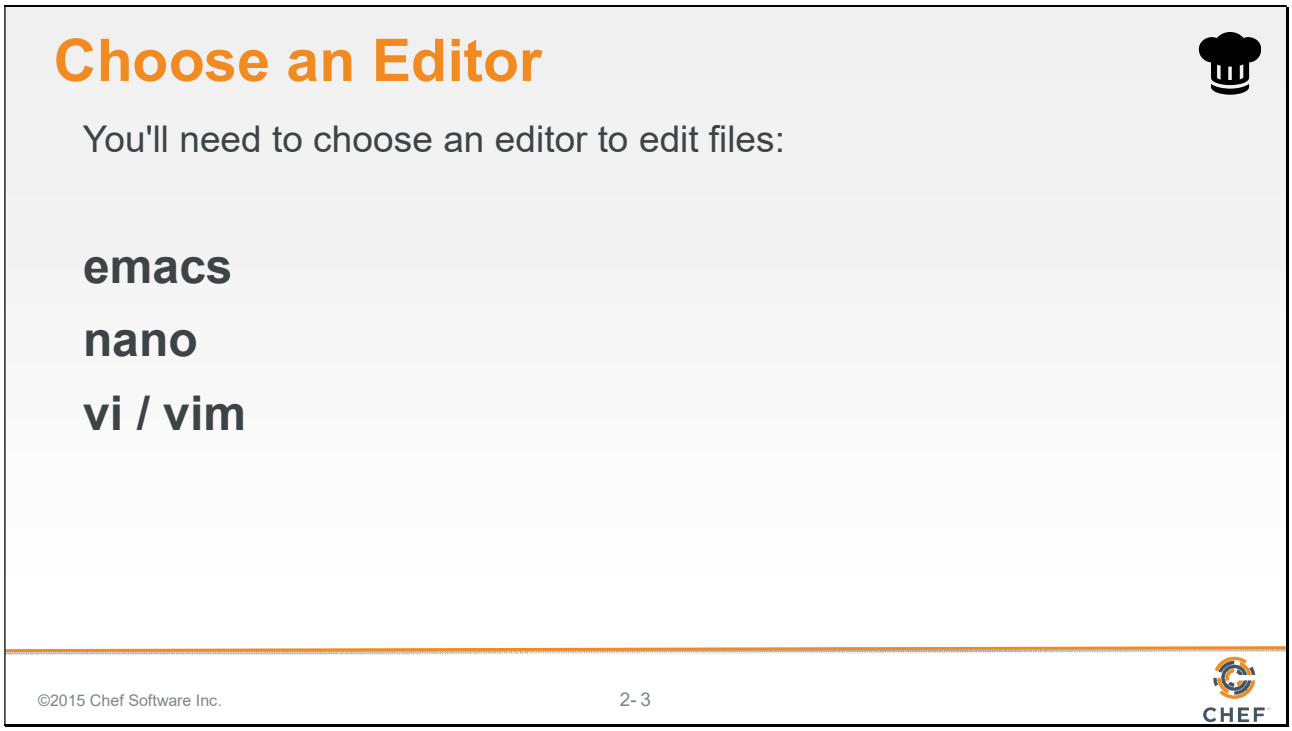
- Use Chef to install packages on your virtual workstation
- Use the chef-apply command
- Create a basic Chef recipe file
- Define Chef Resources

©2015 Chef Software Inc.

CHEF

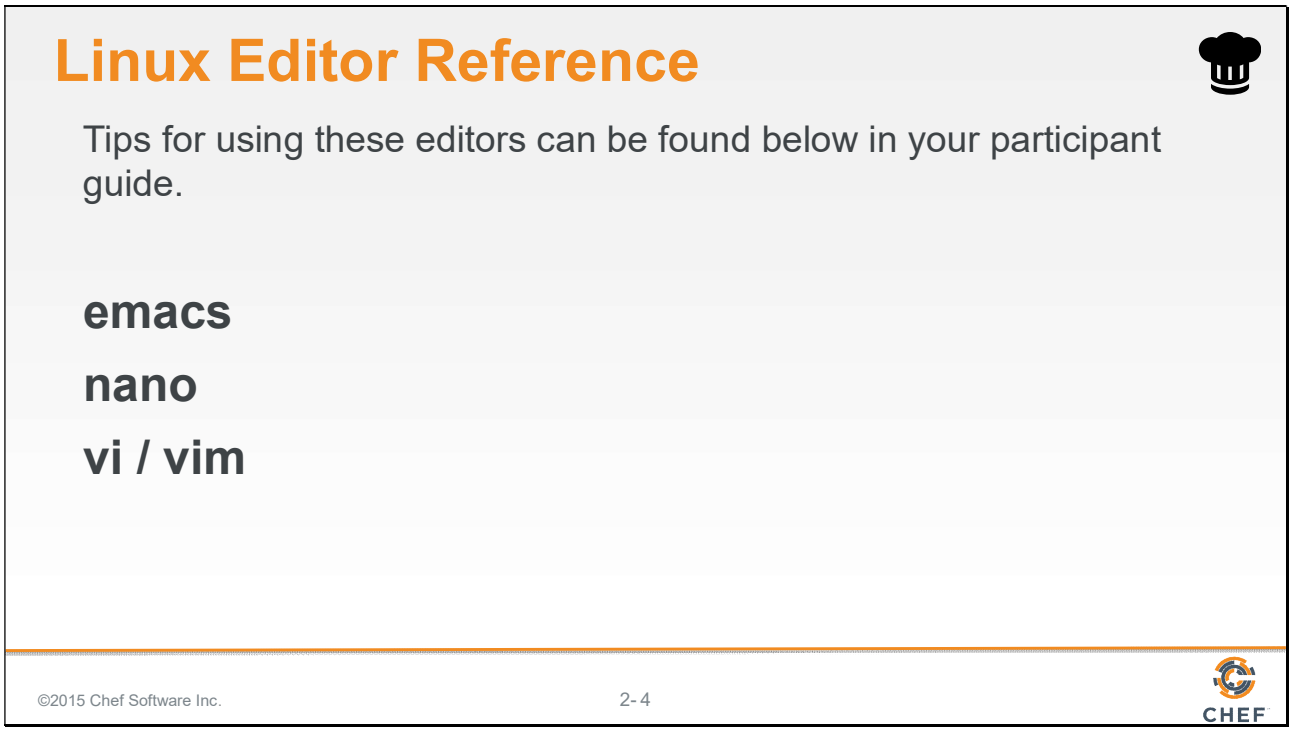
In this module you will learn how to install packages on a virtual workstation, use the 'chef-apply' command, create a basic Chef recipe file and define Chef Resources.

Slide 3



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



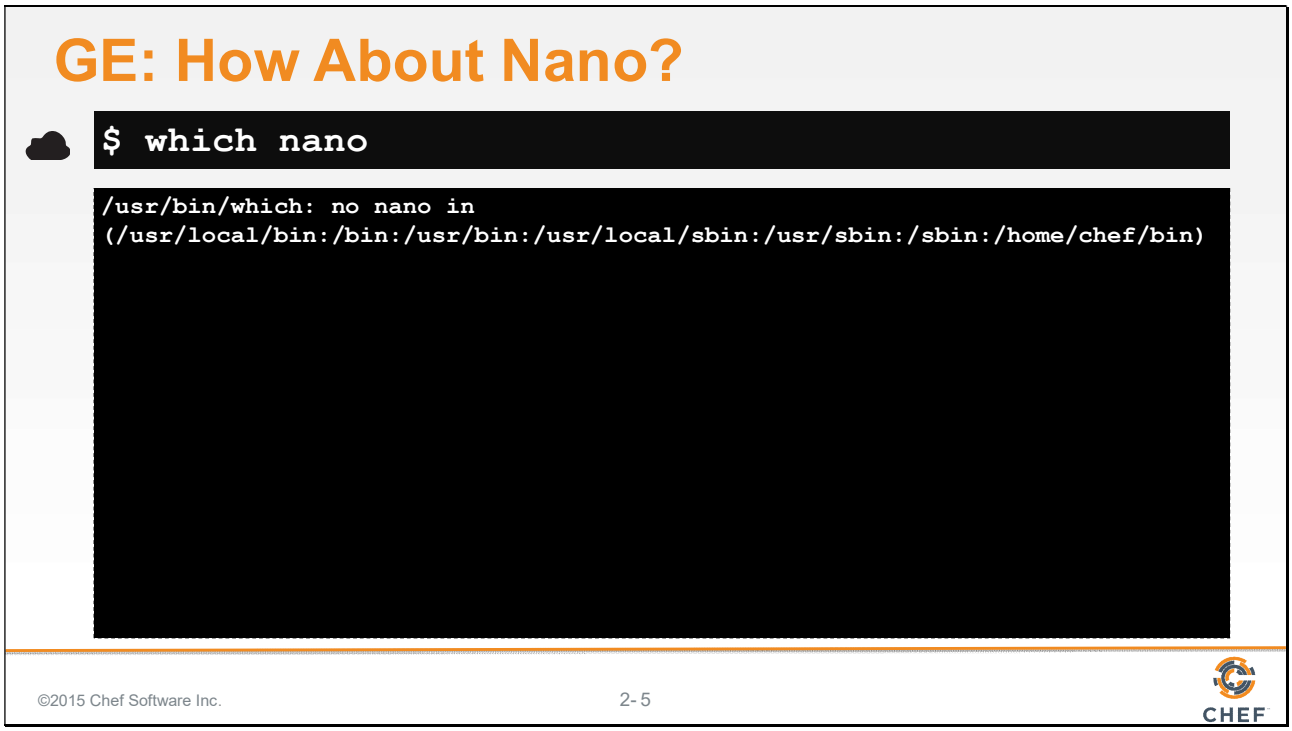
Nano: (Nano is usually touted as the easiest editor to get started with editing through the command-line.)

```
OPEN FILE $ nano FILENAME

WRITE (When exiting) ctrl+x, y, ENTER

EXIT ctrl+x
```

Slide 5

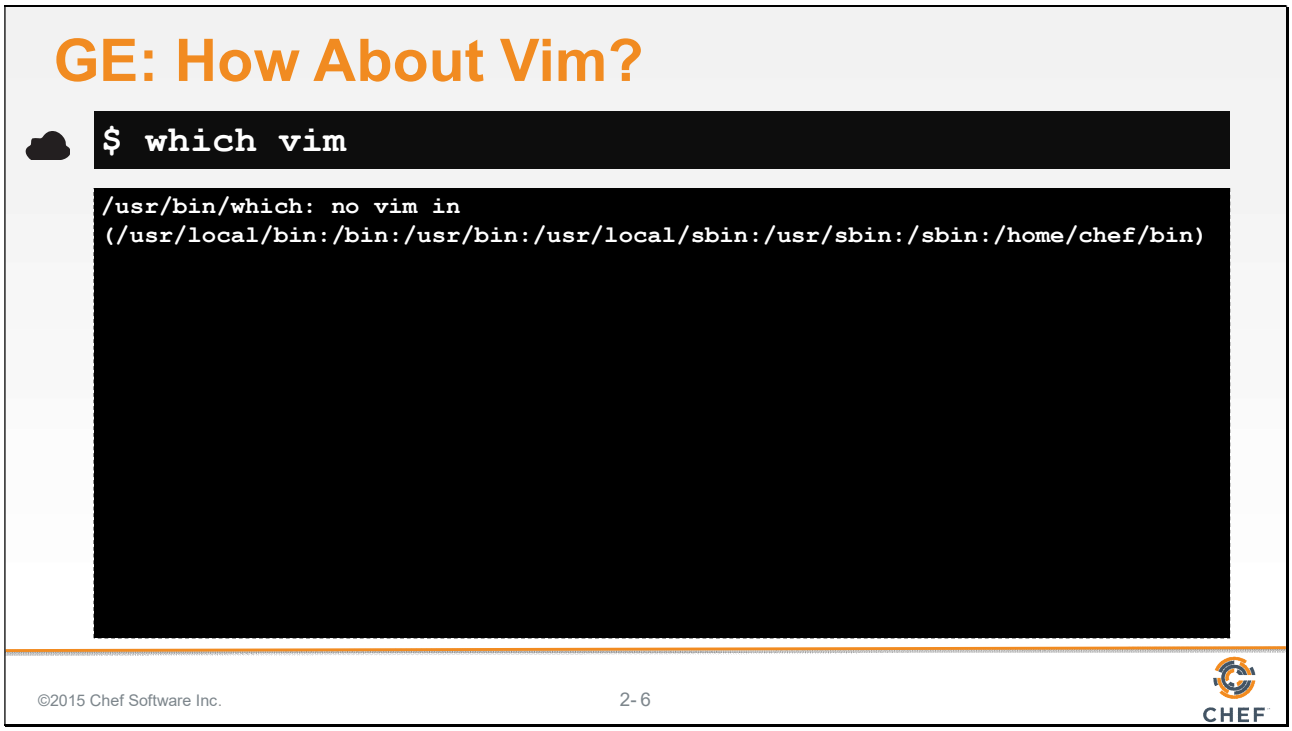


Now that you've picked your editor, you need to find out if it is already installed.

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.

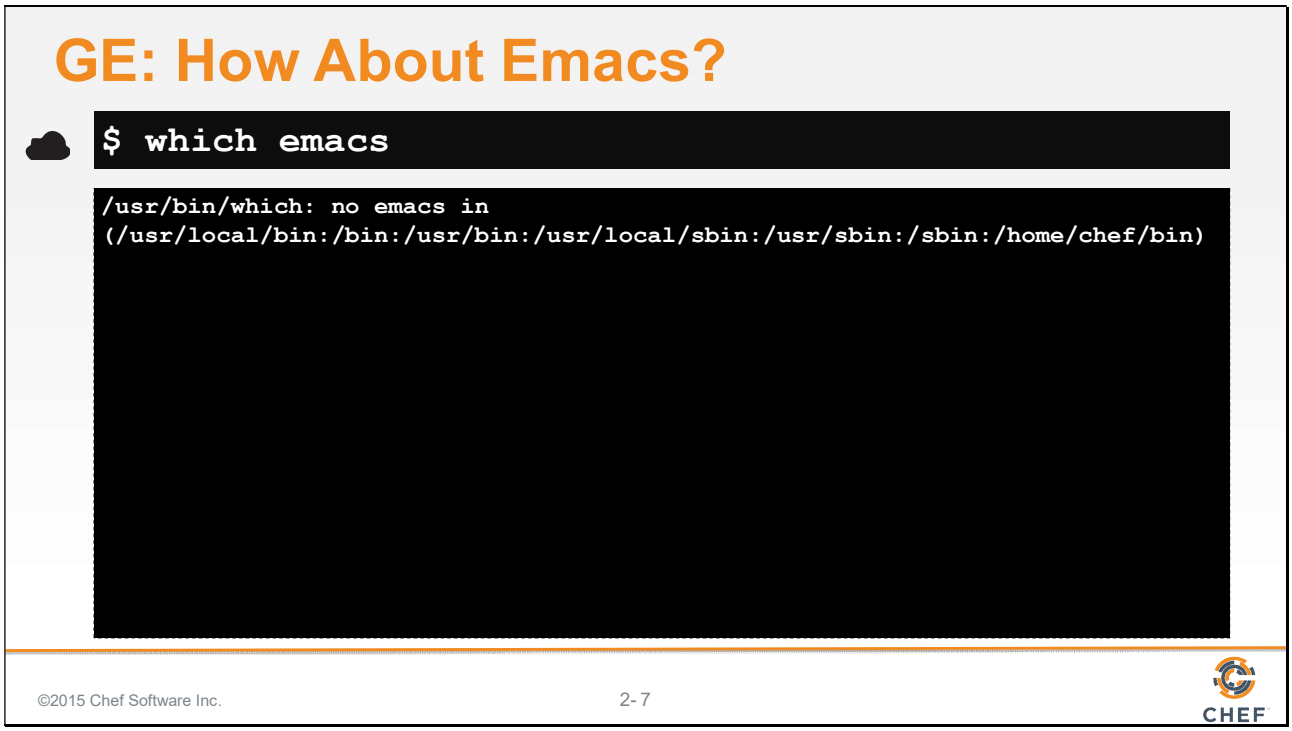
Is nano installed? No, it doesn't look like it.

Slide 6



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

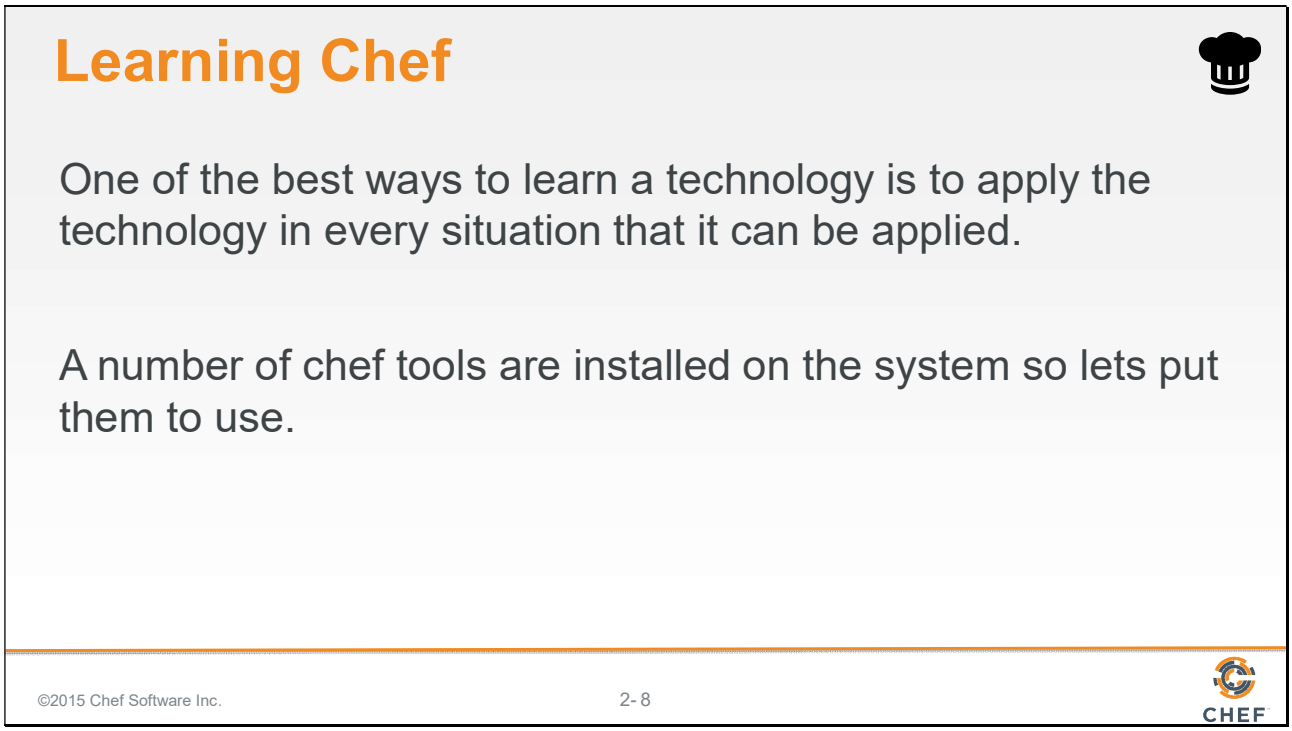
Slide 7



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



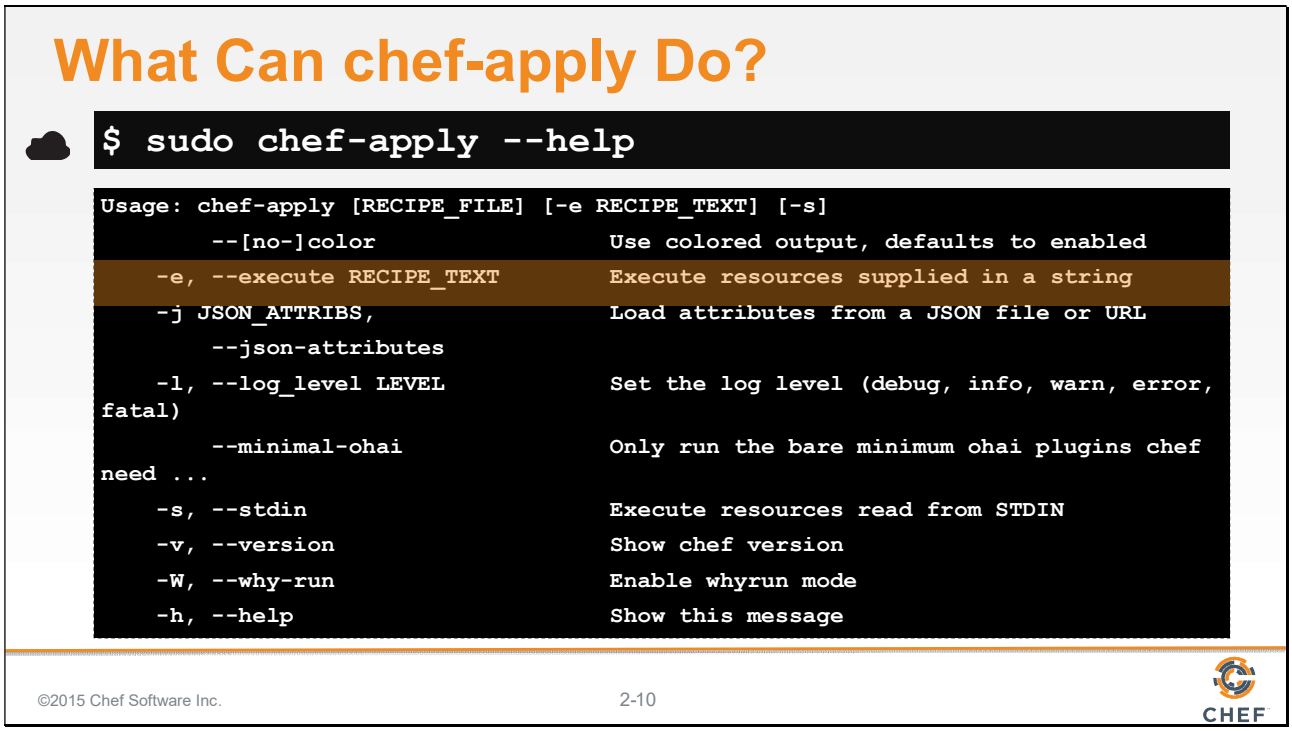
But before you figure out the Linux distribution and start installing packages through the distribution's specific package manager, this seems like a perfect opportunity to experiment with how to solve configuration problems with Chef.

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

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

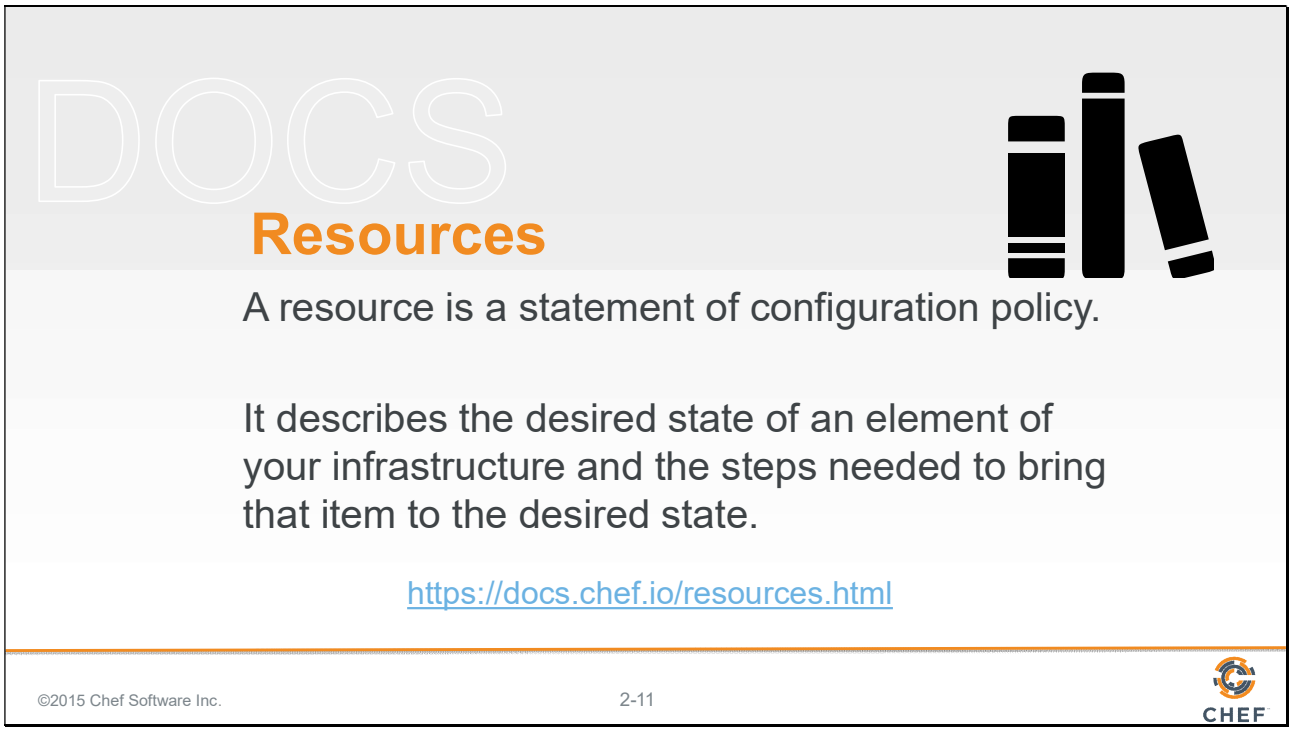


Run the chef-apply application on the workstation with the "--help" flag to learn more about it.

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

©2015 Chef Software Inc.

CHEF

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

©2015 Chef Software Inc.

CHEF

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

Slide 14

Example: File

```
file '/etc/motd' do
  content 'This computer 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

©2015 Chef Software Inc.

CHEF

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.

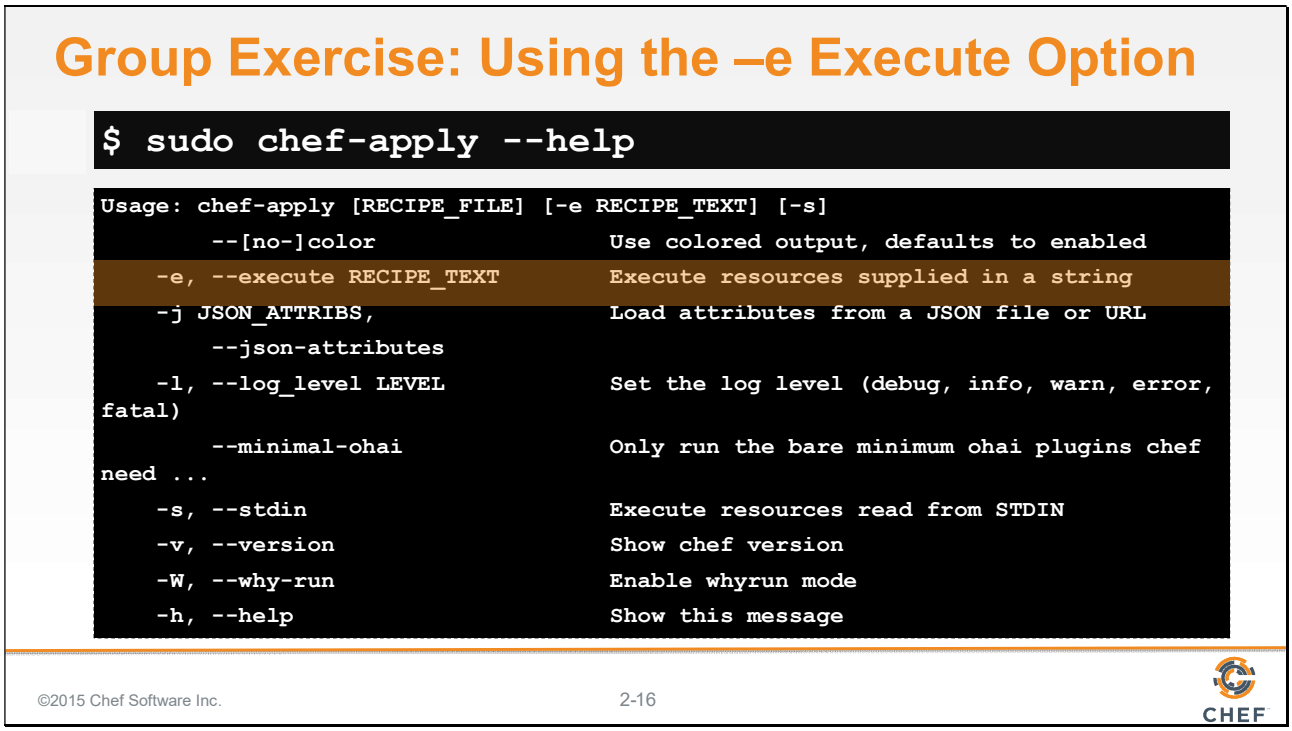
https://docs.chef.io/resource_file.html

©2015 Chef Software Inc.

CHEF

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

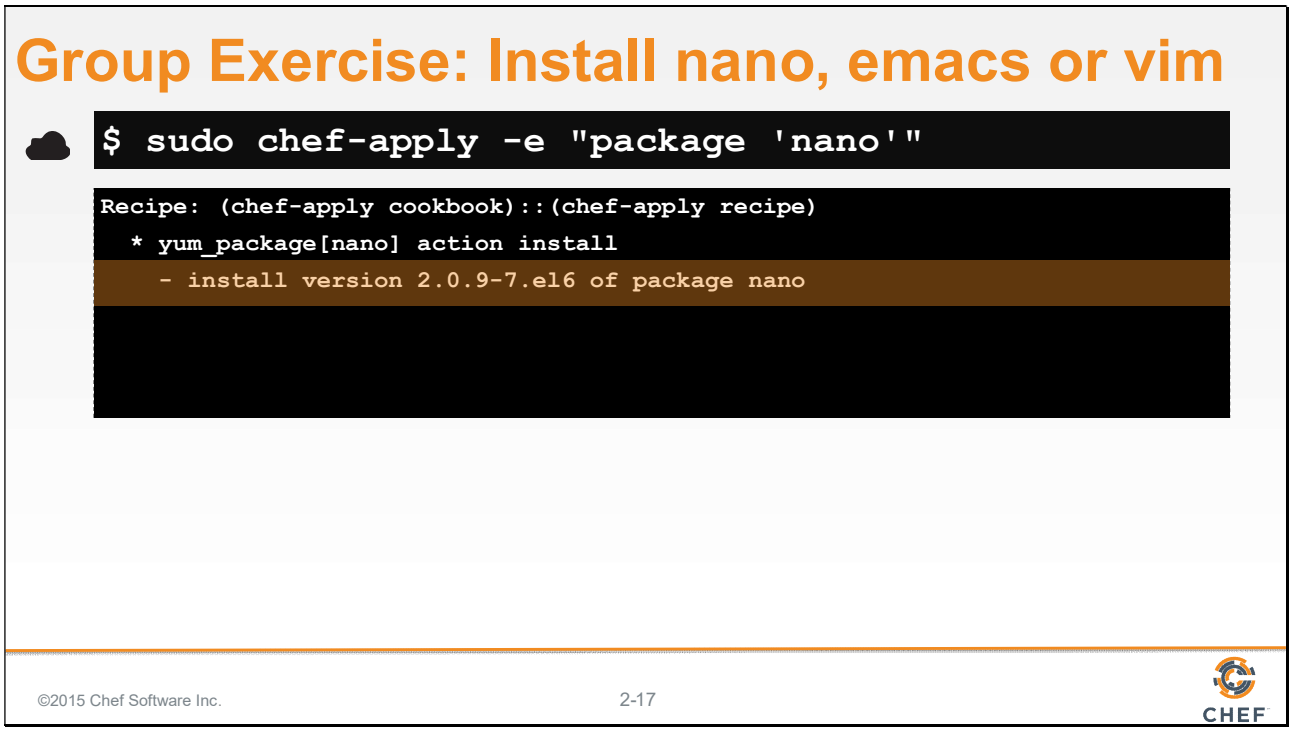
Slide 16



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.

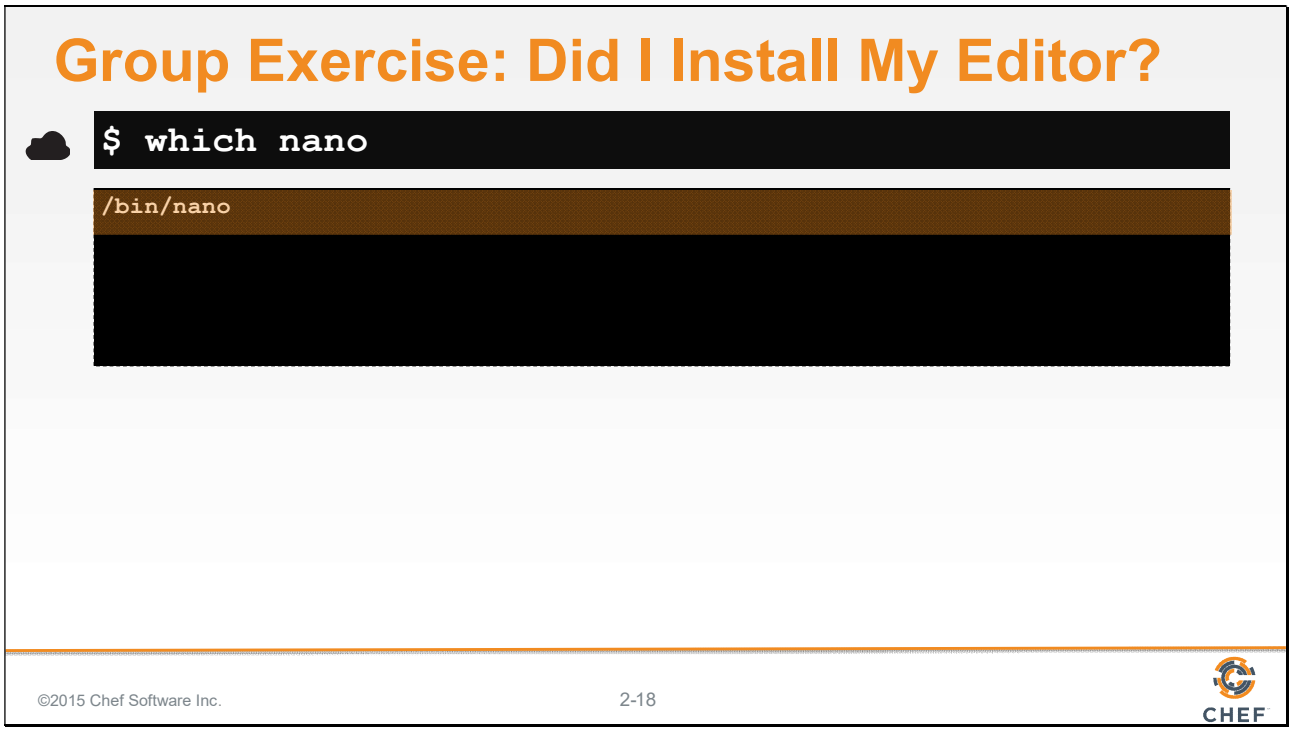
Slide 17



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.

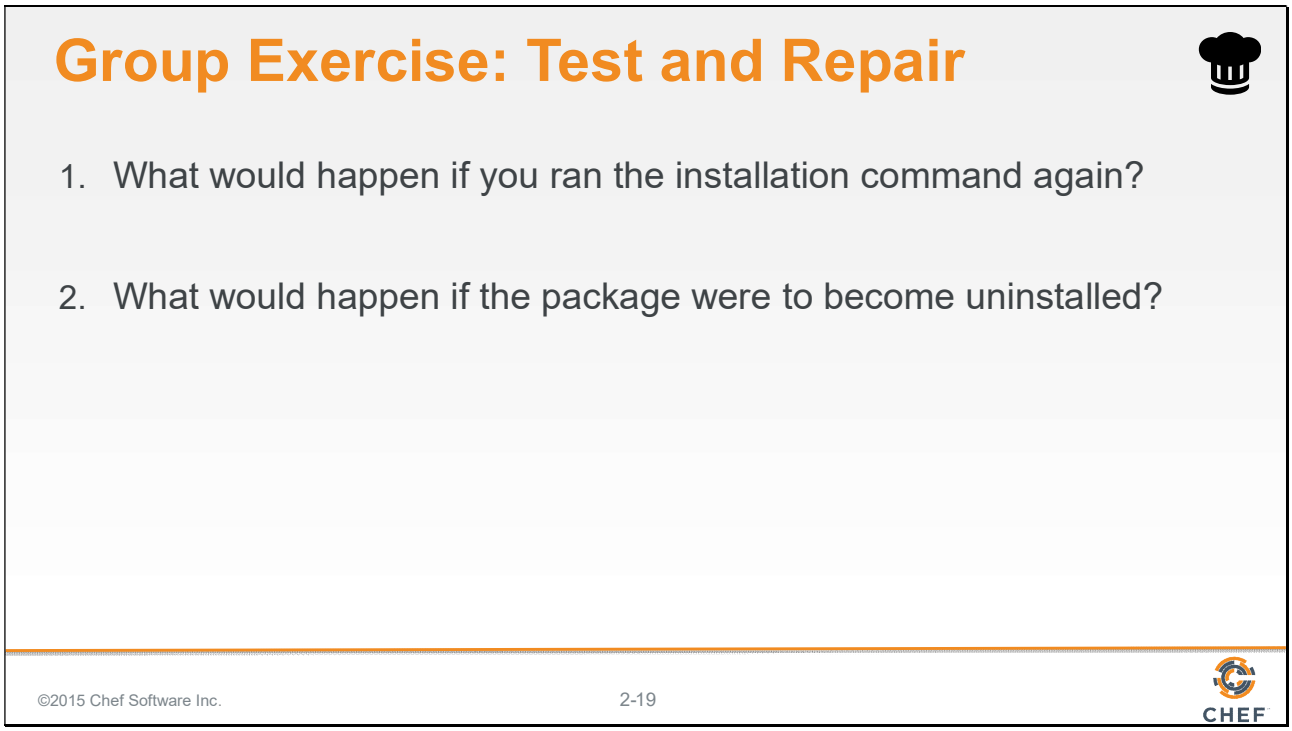
Slide 18



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



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

Test and Repair

`chef-apply` takes action only when it needs to. Think of it as test and repair.

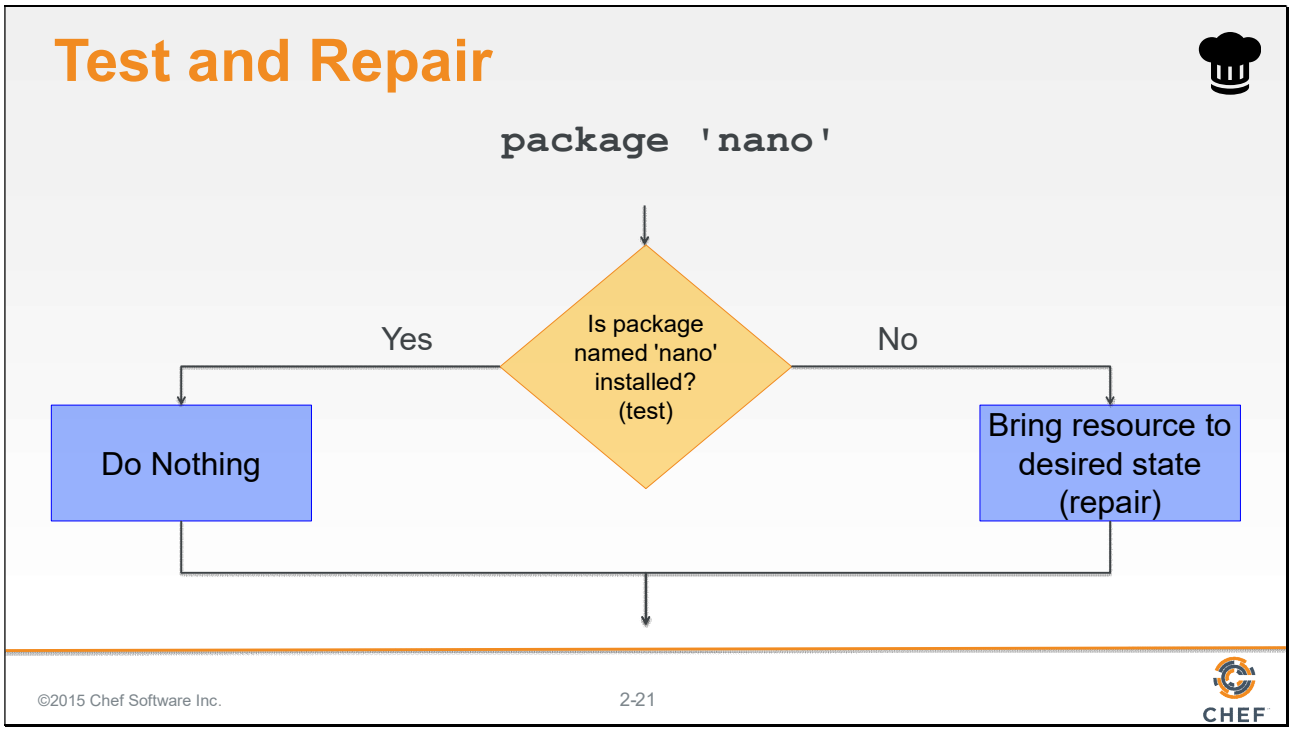
Chef looks at the current state of each resource and takes action only when that resource is out of policy.

©2015 Chef Software Inc.

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.

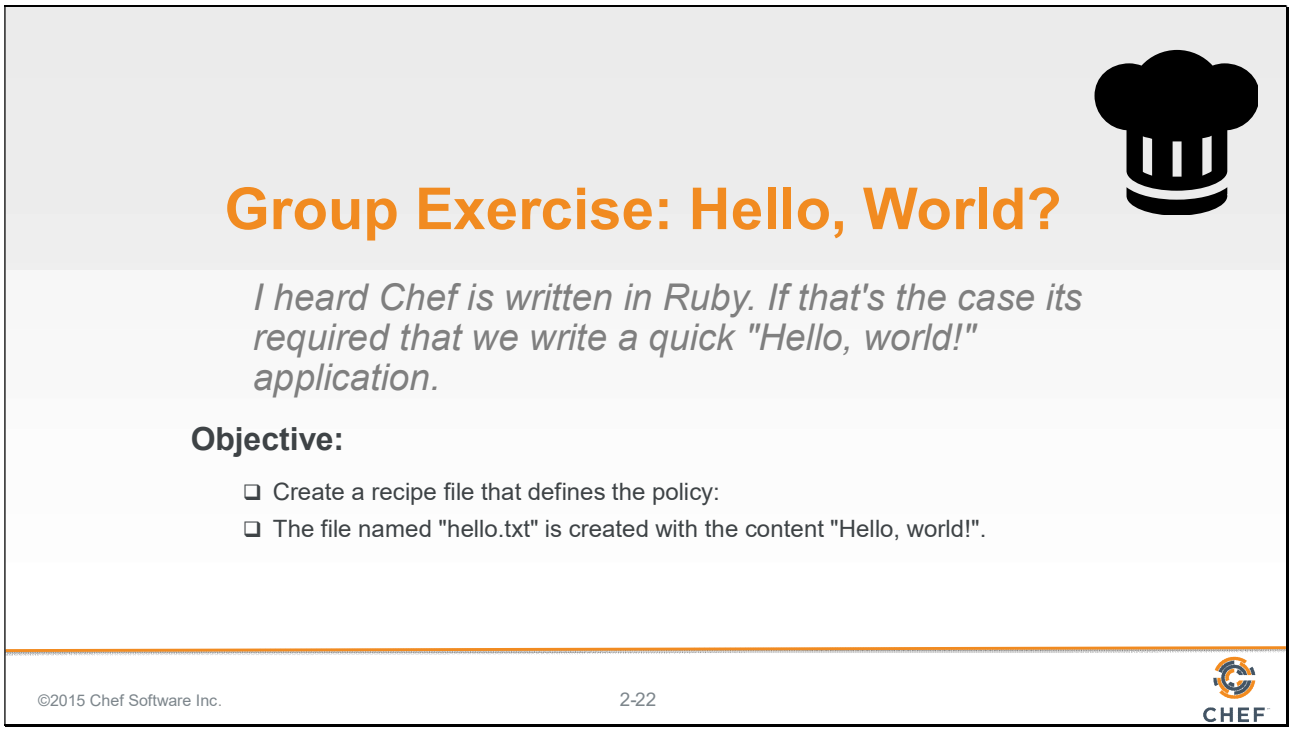
Slide 21



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

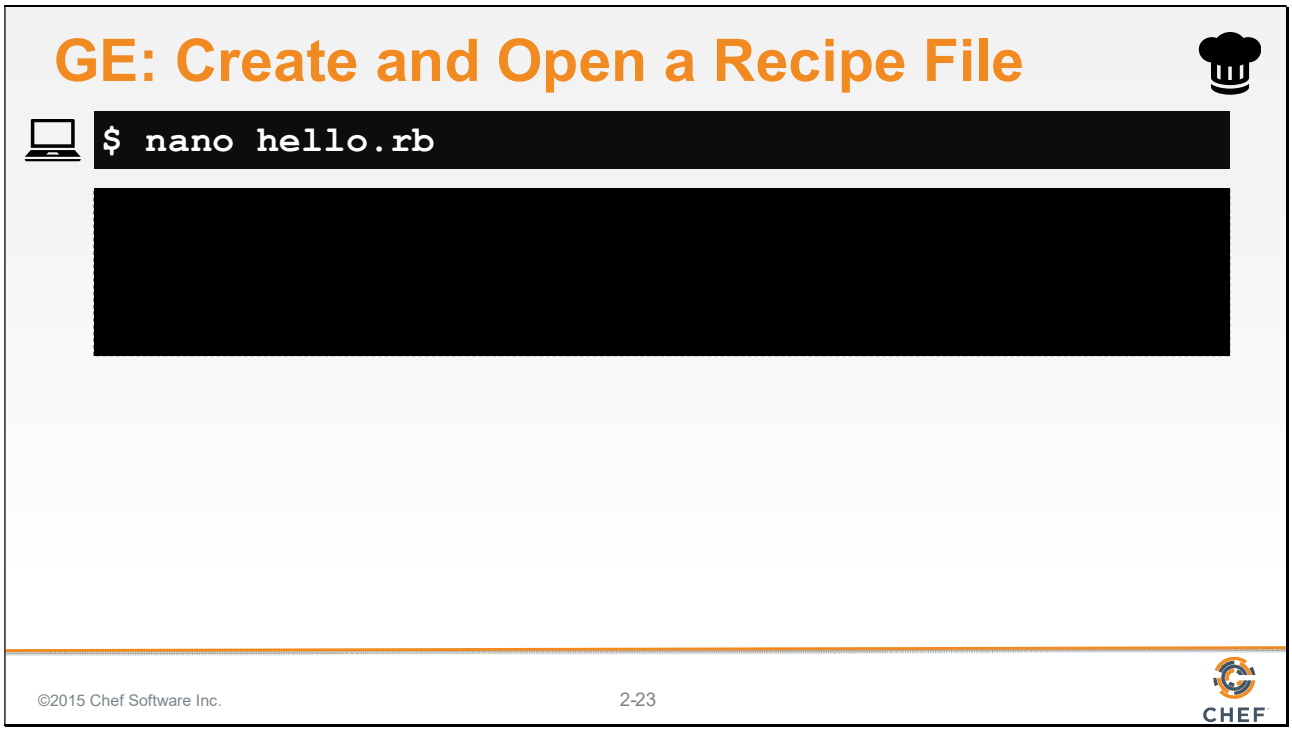


Great! You installed an editor using `chef-apply` but we missed a very important step.

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



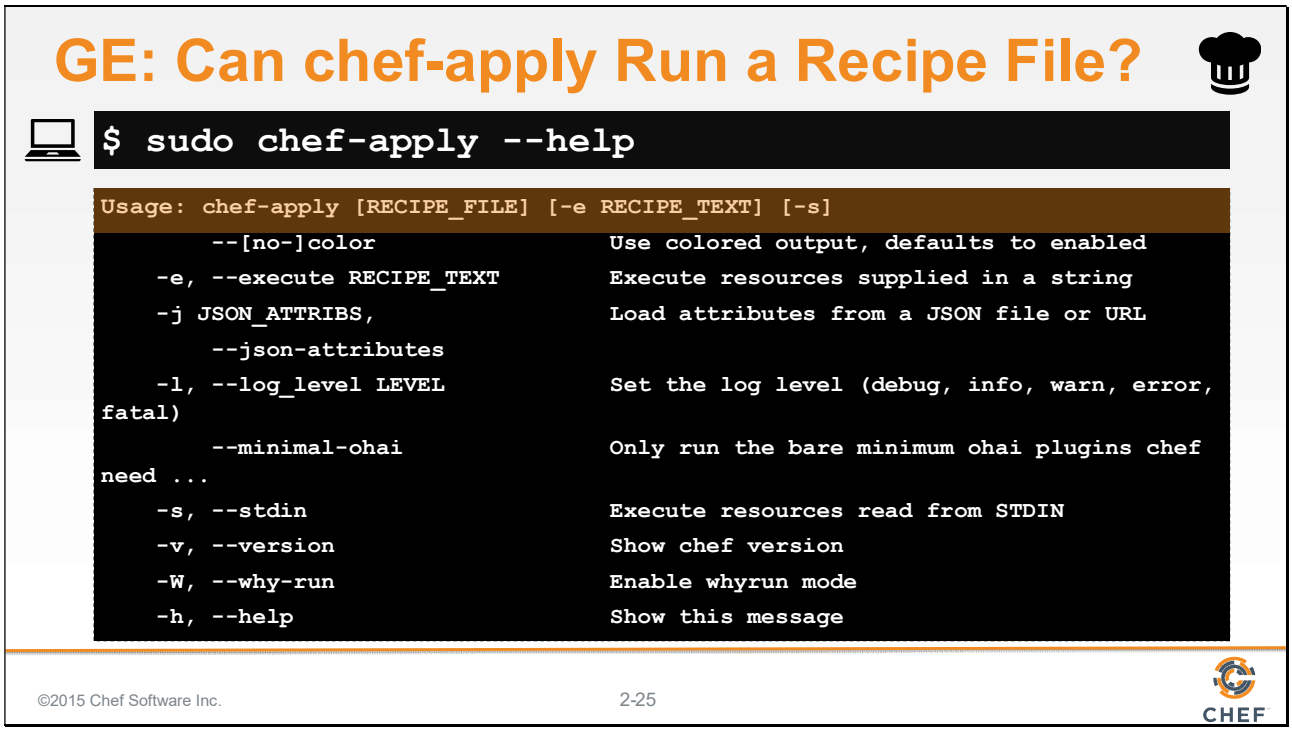
Using your editor open the file named 'hello.rb'. 'hello.rb' is a recipe file. It has the extension '.rb' because it is a ruby file.

Slide 24

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!'.

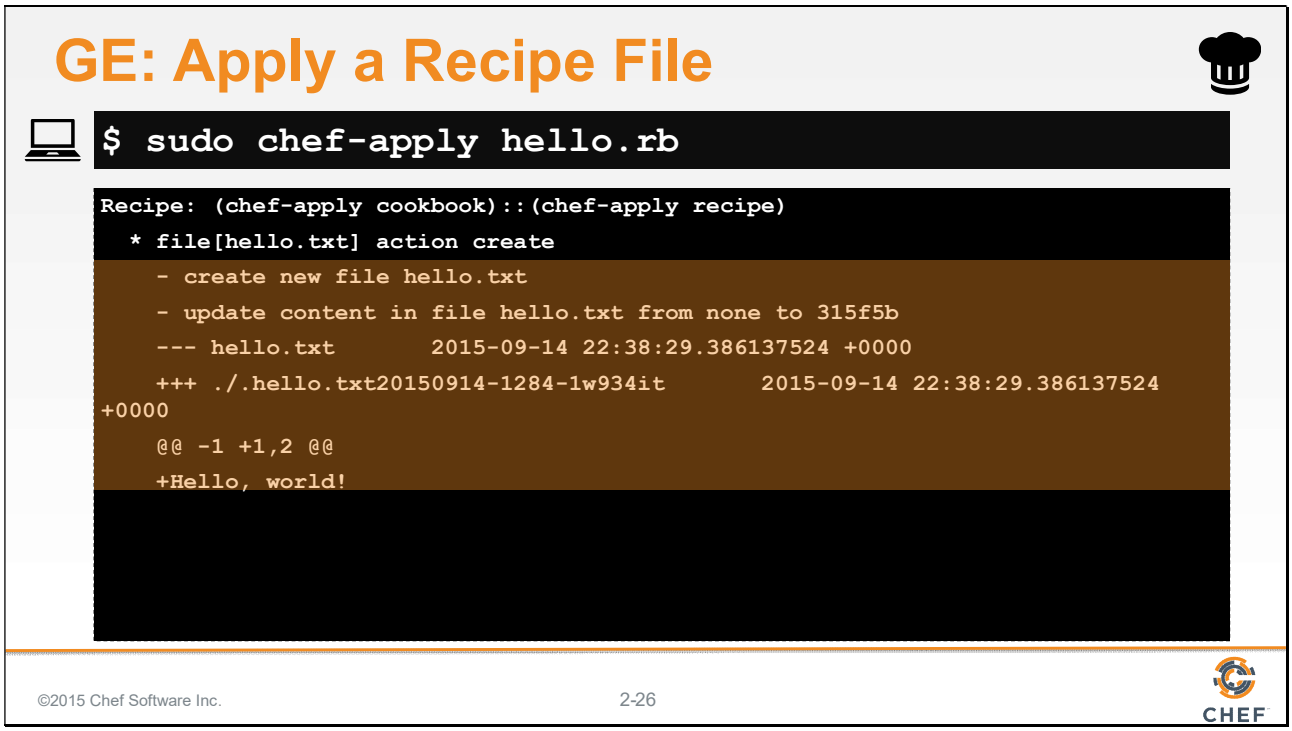
Save the file and return to the terminal and the `chef-apply` command.

Slide 25



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

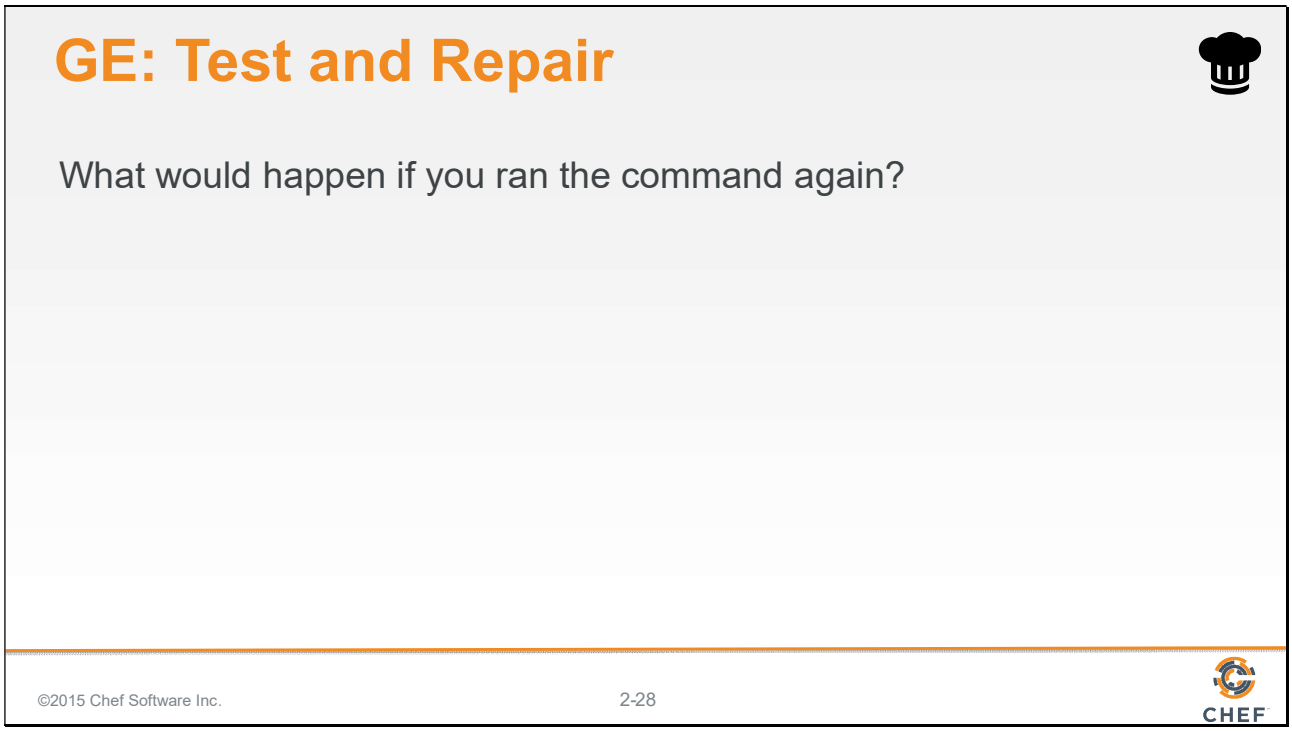


Type the specified command to apply the recipe file. You should see that a file named 'hello.txt' was created and the contents updated to include your 'Hello, world!' text.

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



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

GE: Test and Repair

What would happen if the file contents were modified?

Go ahead and modify the contents of 'hello.txt' with your text editor.
Write the file and then think about what you expect to see in the output.
Then run the chef-apply command again.

©2015 Chef Software Inc.

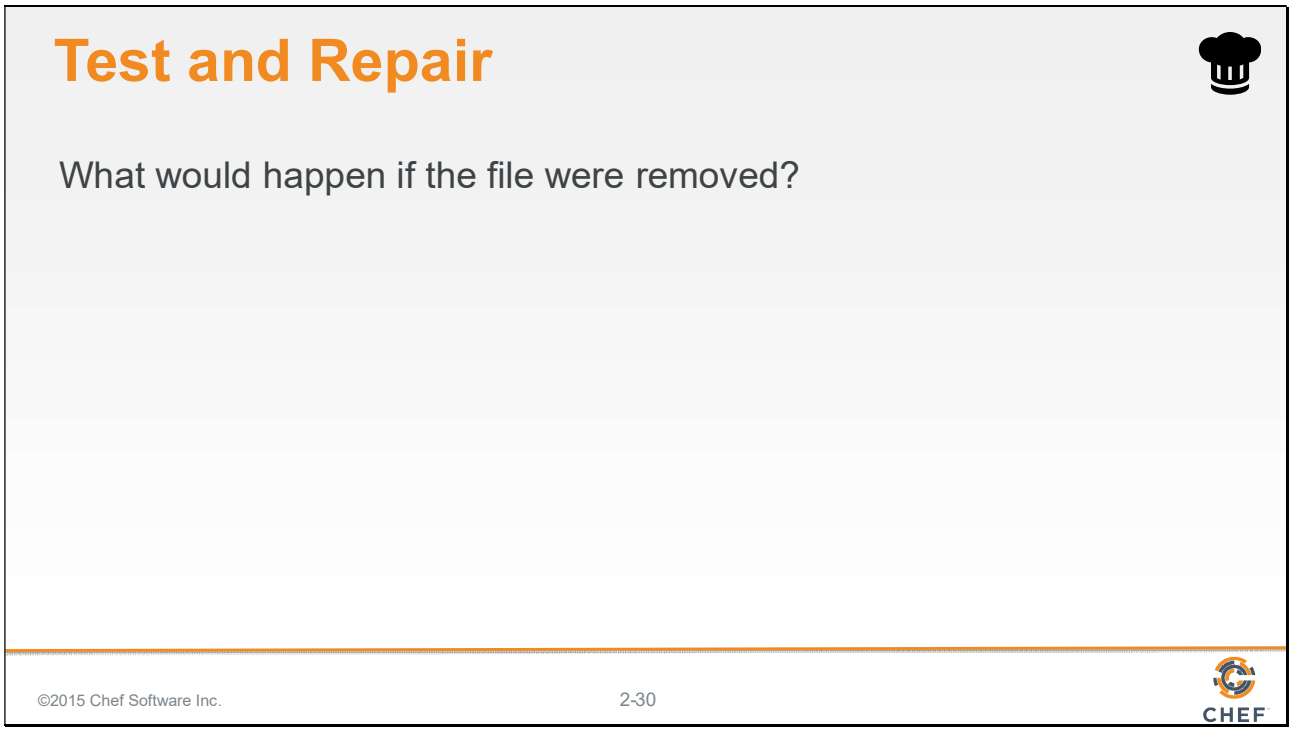
CHEF

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



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

Test and Repair

What would happen if the file permissions (mode), owner, or group changed?

Have we defined a policy for these attributes?

©2015 Chef Software Inc.

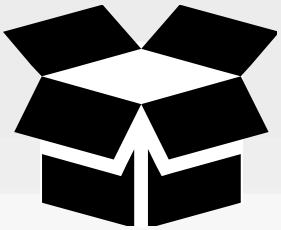
CHEF

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

CONCEPT

Resource Definition



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

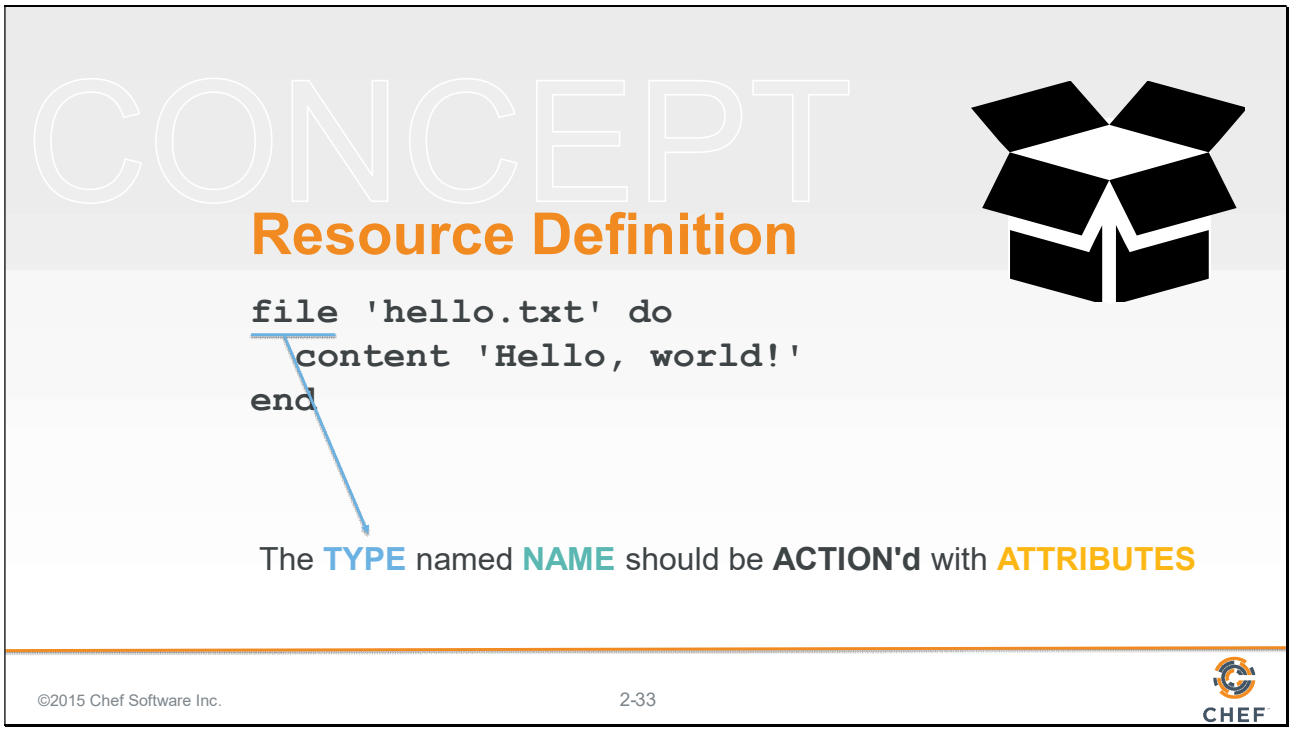
The TYPE named NAME should be **ACTION'd** with ATTRIBUTES

©2015 Chef Software Inc.

CHEF

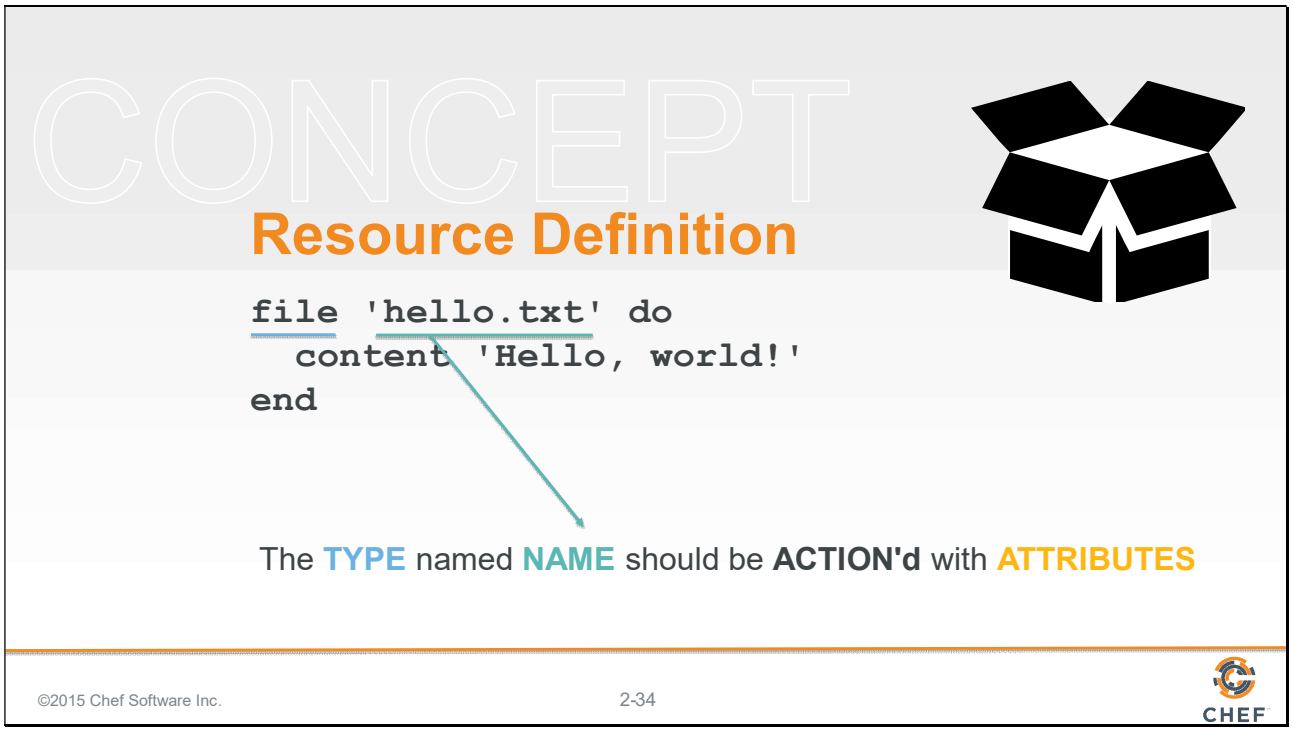
Let's take a moment and talk about the structure of a resource definition. We'll break down the resource that we defined in our recipe file.

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'.

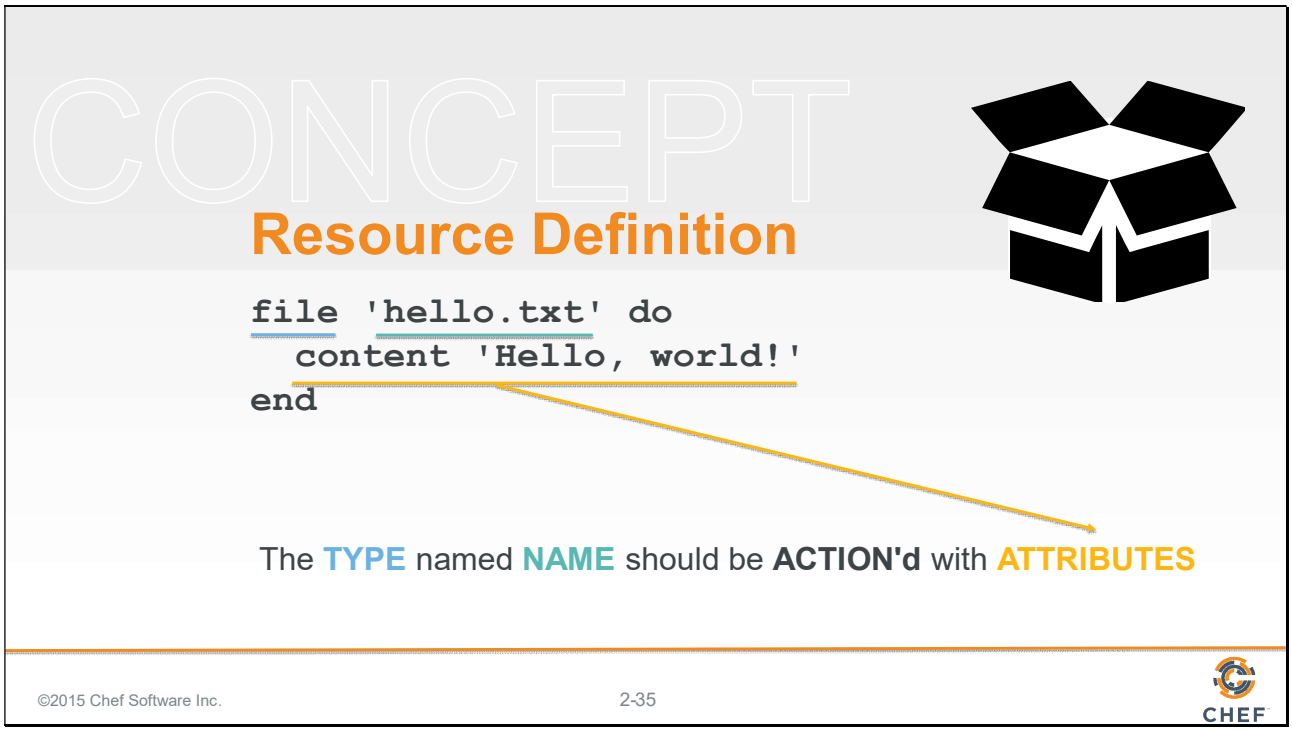
Slide 34



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.

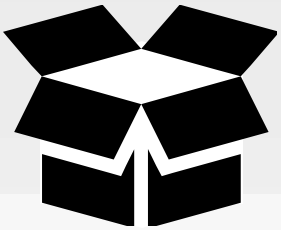
The contents of this block contains attributes (and other things) that help describe the state of the resource. In this instance, the content attribute here specifies the contents of the file.

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

Slide 36

CONCEPT

Resource Definition



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

?

The TYPE named NAME should be **ACTION'd** with ATTRIBUTES

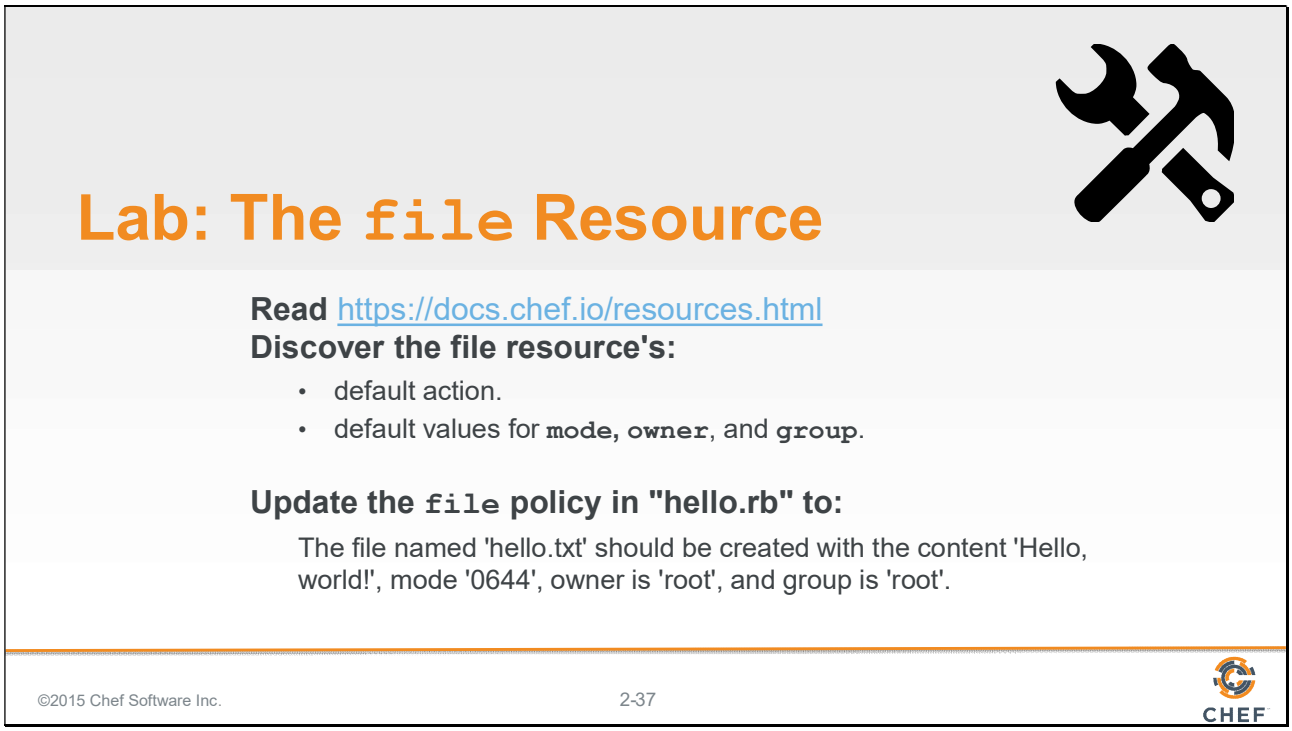
©2015 Chef Software Inc.

CHEF

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

Lab: The Updated file Resource

~/hello.rb

```
file 'hello.txt' do
  content 'Hello, world!'
  mode '0644'
  owner 'root'
  group 'root'
  action :create
end
```

The default mode is set by the POSIX Access Control Lists.

The default owner is the current user (could change).

The default group is the POSIX group (if available).

The default action is to create (not necessary to define it).

©2015 Chef Software Inc.

CHEF

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 for this Operating System 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.

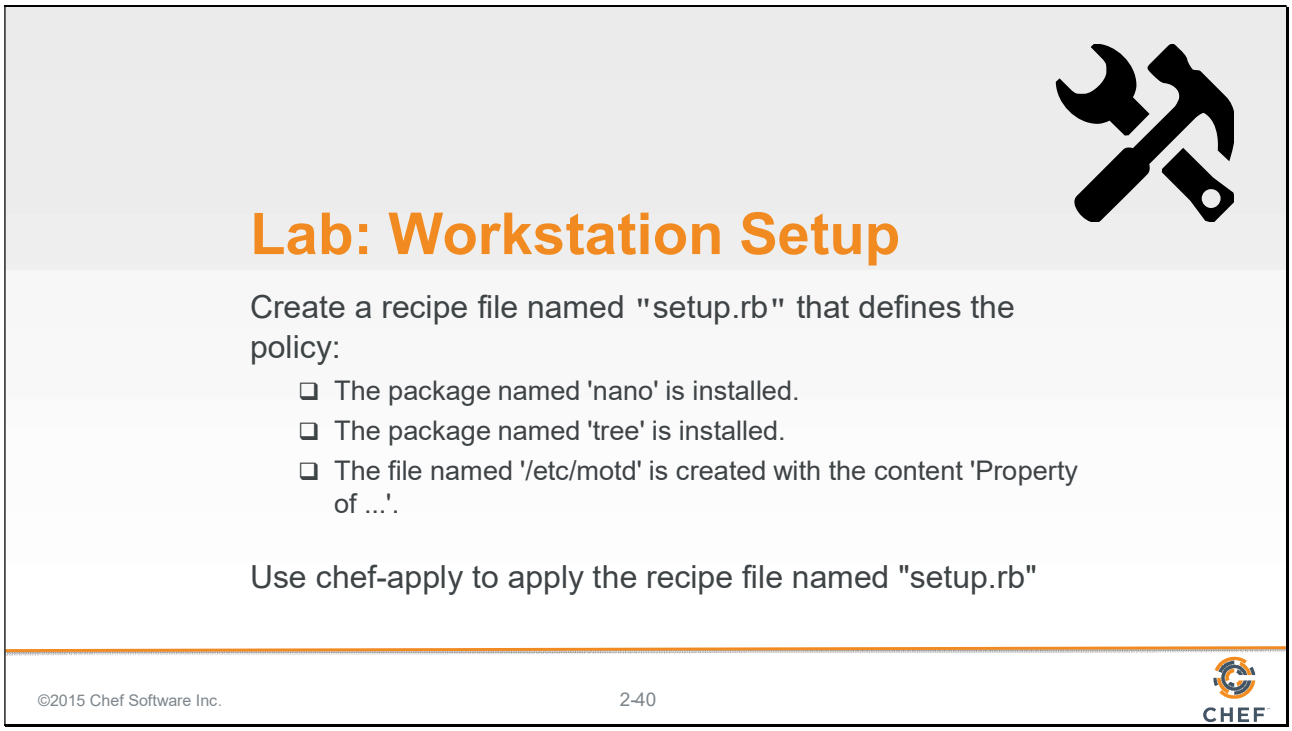
Slide 39

Questions

What questions can we answer for you?

©2015 Chef Software Inc.

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 ...'.

©2015 Chef Software Inc.

CHEF

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)
  * yum_package[nano] action install (up to date)
  * yum_package[vim] action install
    - install version 7.4.629-5.e16 of package vim-enhanced
  * yum_package[emacs] action install
    - install version 23.1-28.e16 of package emacs
  * yum_package[tree] action install
    - install version 1.5.3-3.e16 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   2015-12-14 18:29:54.049748179 +0000
```

©2015 Chef Software Inc.

This is how you apply the created recipe.

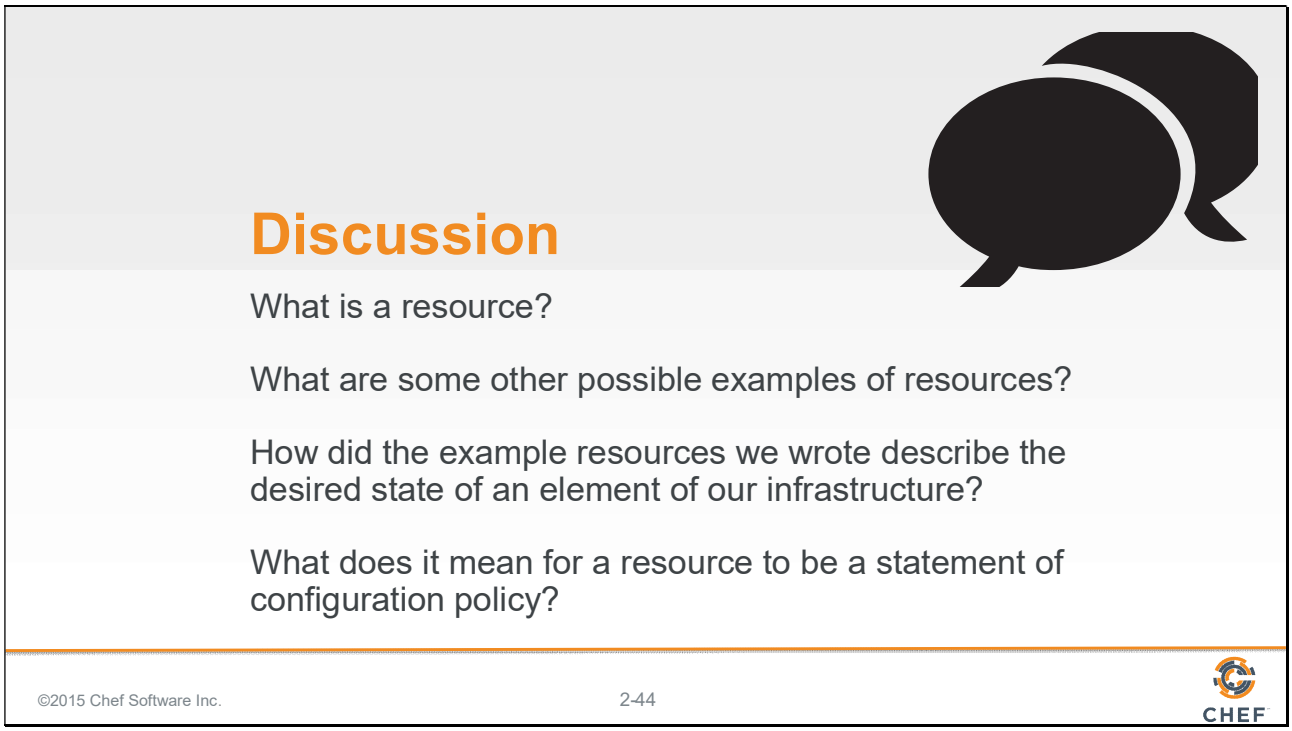
Slide 43

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?

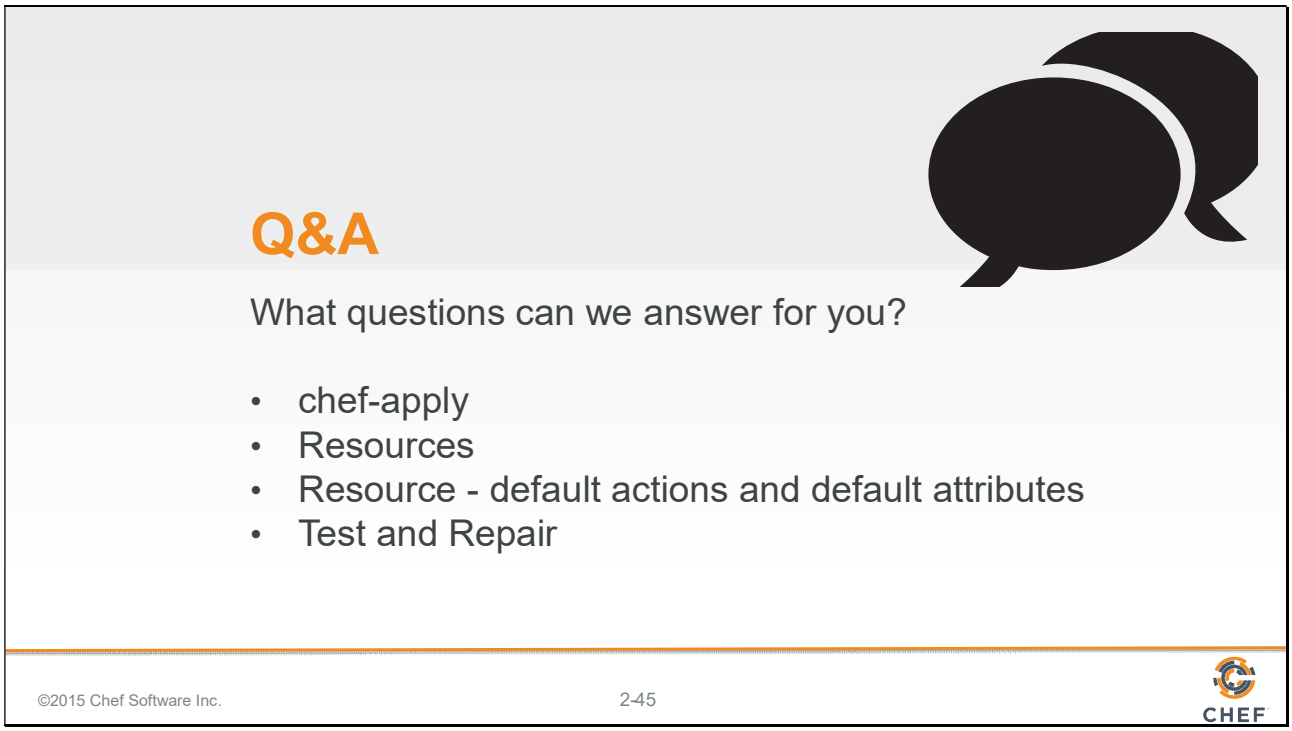
What are some other possible examples of resources?

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