2: Chef Resources

Slide 2

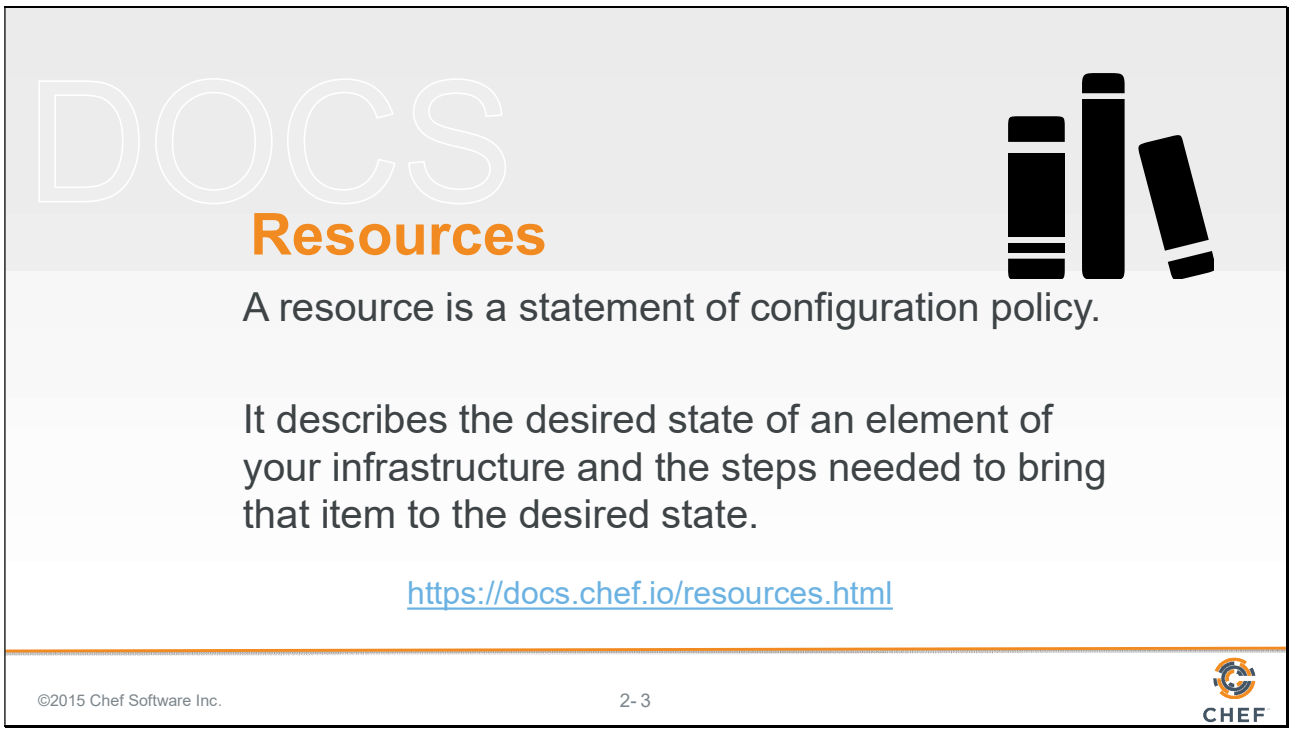
Objectives

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



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.

Instructor Note: This may sound unusual to ask people to read the documentation site but it is important that they learn to refer to the documentation. This page in an important reference page.

Slide 4

Example: powershell_script

```
powershell_script 'Install IIS' do
  code 'add-windowsfeature Web-Server'
  action :run
end
```

The powershell_script named 'Install IIS' is run with the code 'add-windowsfeature Web-Server'.

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

CHEF

Here is an example of the powershell_script resource. The powershell_script named 'Install IIS' is run with the code 'add-windowsfeature Web-Server'

Slide 5

Example: service

```
service 'w3svc' do
  action [ :enable, :start ]
end
```

The service named 'w3svc' is enabled (start on reboot) and started.

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

In this example, the service named w3svc' is enabled and started.

Service resources are often defined with two actions. The action method can only take one parameter so to provide two actions you need to specify the two actions within an Array.

Slide 6

Example: file

```
file 'c:\inetpub\wwwroot\Default.htm' do
  content 'Hello, world!'
  rights :read, 'Everyone'
end
```

The file 'c:\inetpub\wwwroot\Default.htm' with the content 'Hello, world!' and grants 'read' rights for 'Everyone'.

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

In this example, the file named 'c:\inetpub\wwwroot\Default.htm' with the content 'Hello, world!' and has allowed Everyone rights to read the file.

The default action for the file resource is to create the file.

Slide 7

Example: file

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

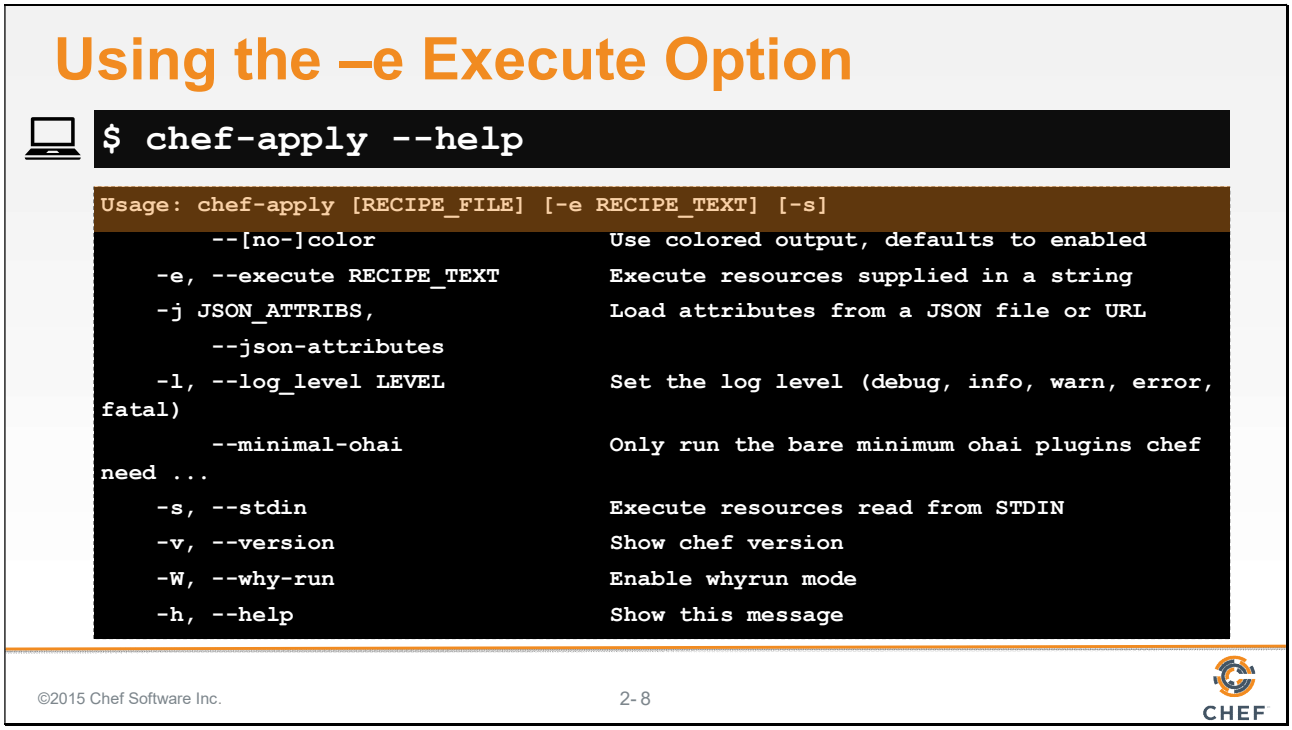
The file name '/etc/php.ini.default' is deleted.

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

CHEF

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

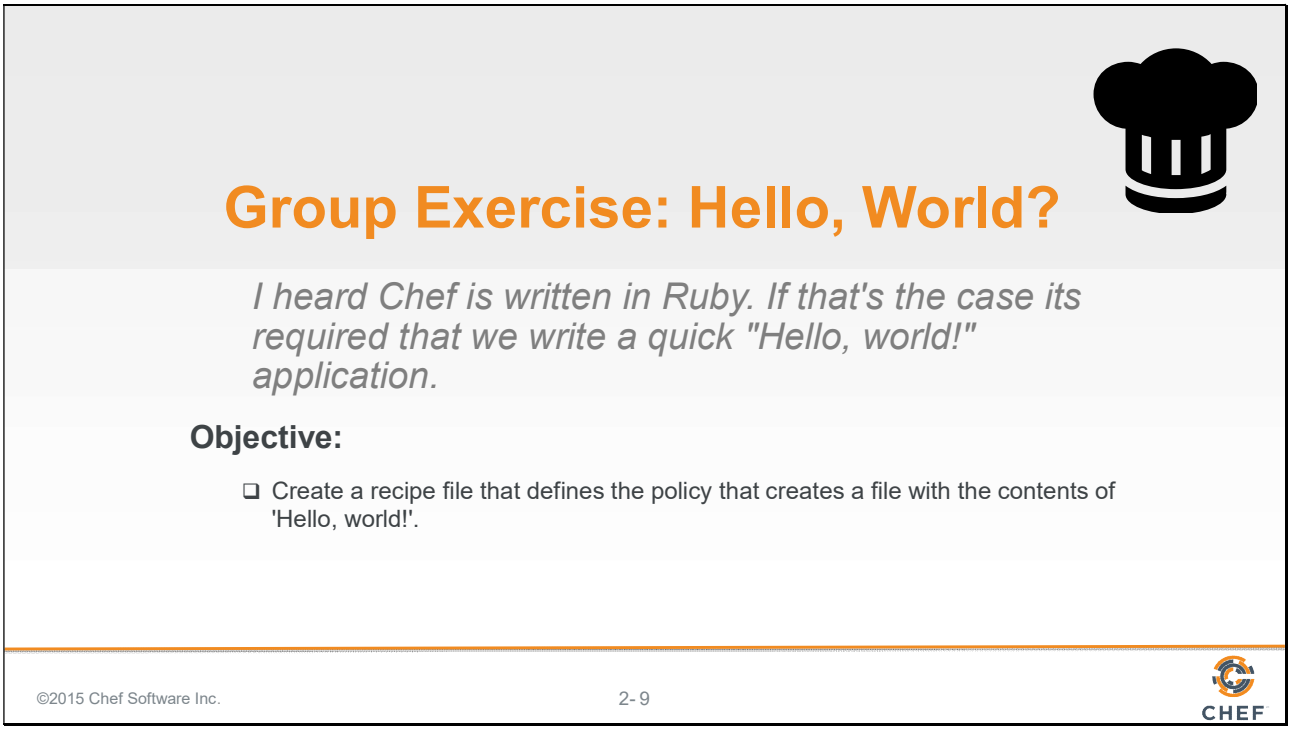
Slide 8



Let's take a look at the `chef-apply` command. The `chef-apply` command is installed in the Chef Development Kit. It is a command that allows you to apply recipe files, recipe text as a string on the command line (-e flag), or even accept input from the STDIN (-s 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.

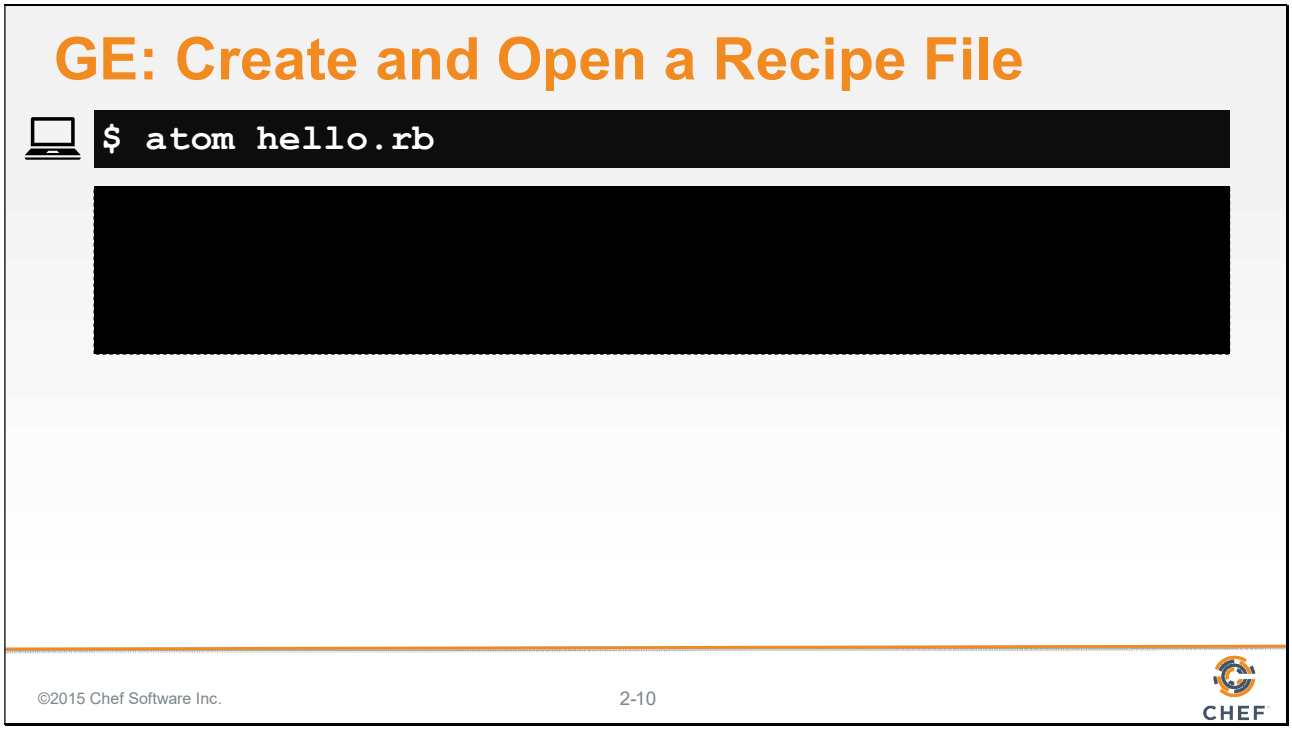
Slide 9



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 10



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 11

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!'

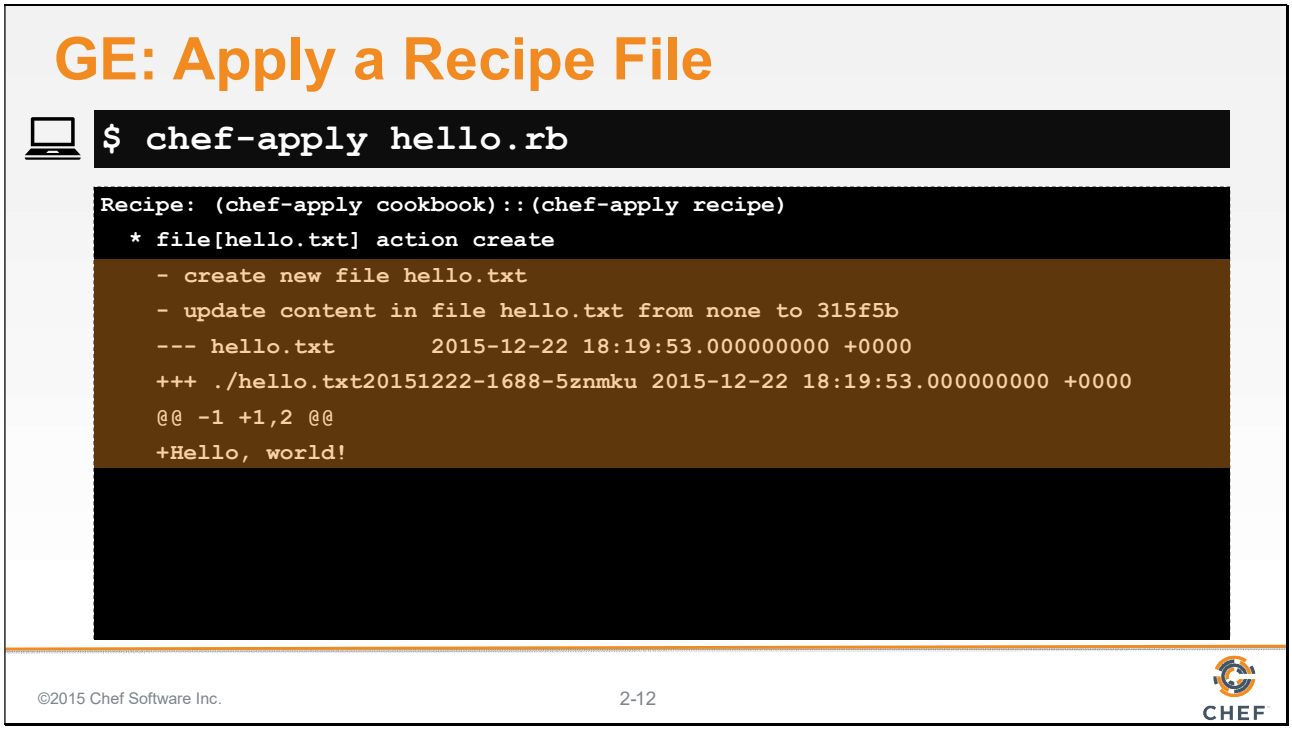
https://docs.chef.io/resources.html

CHEF

• 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 command prompt.

Slide 12



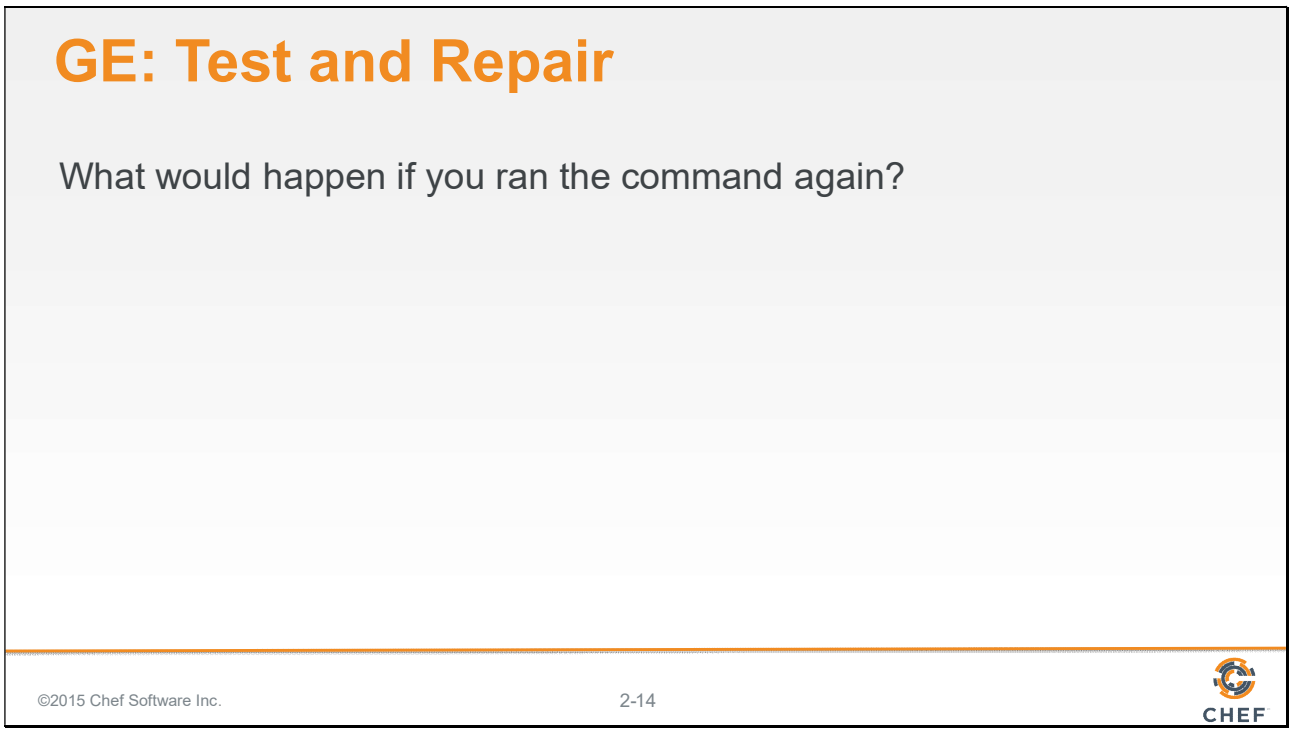
Type the specified command to apply the recipe file. You should see that a file named 'hello.txt' was created and the contents updated to include your 'Hello, world!' text.

The output that shows the contents of the file have been modified is being displayed in a format similar to a git diff (http://stackoverflow.com/questions/2529441/how-to-read-the-output-from-git-diff).

Slide 13

Let's 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!'.

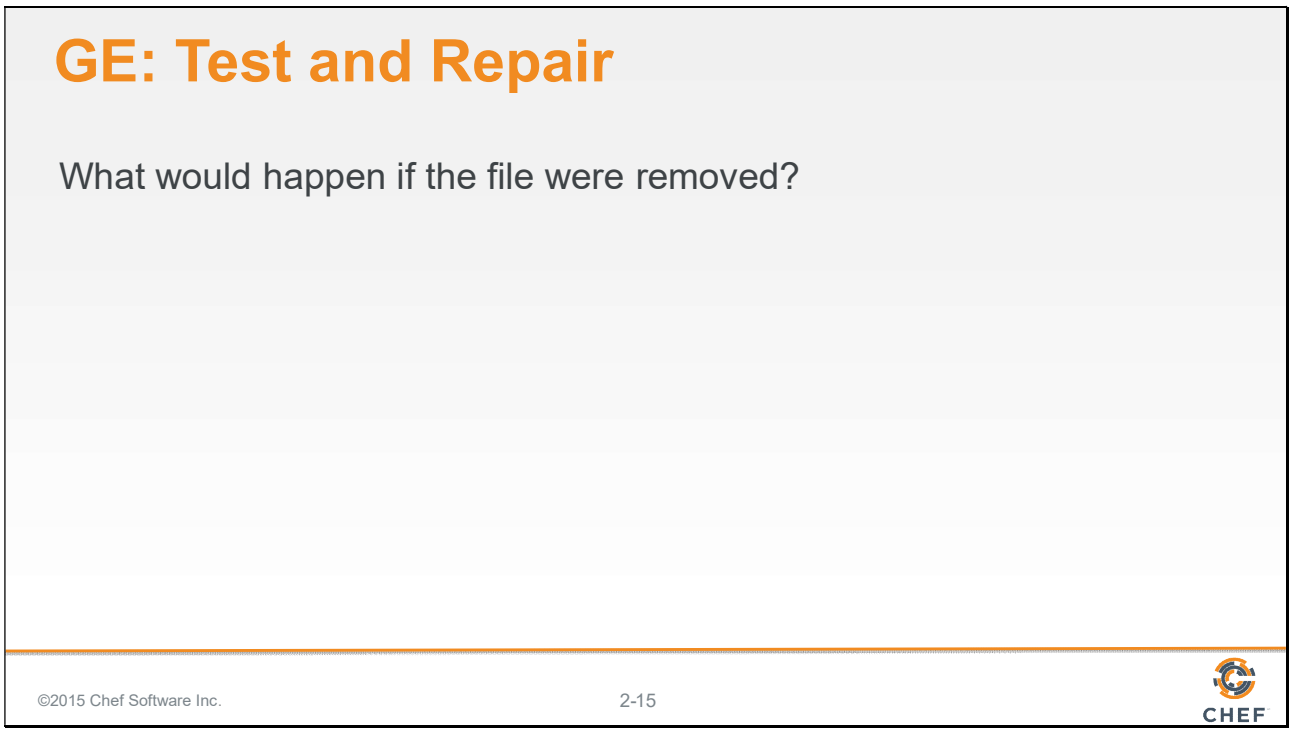
Slide 14



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 15



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 16

• 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.

Instructor Note: Allow 5 minutes to complete this exercise.

Slide 17

Test and Repair

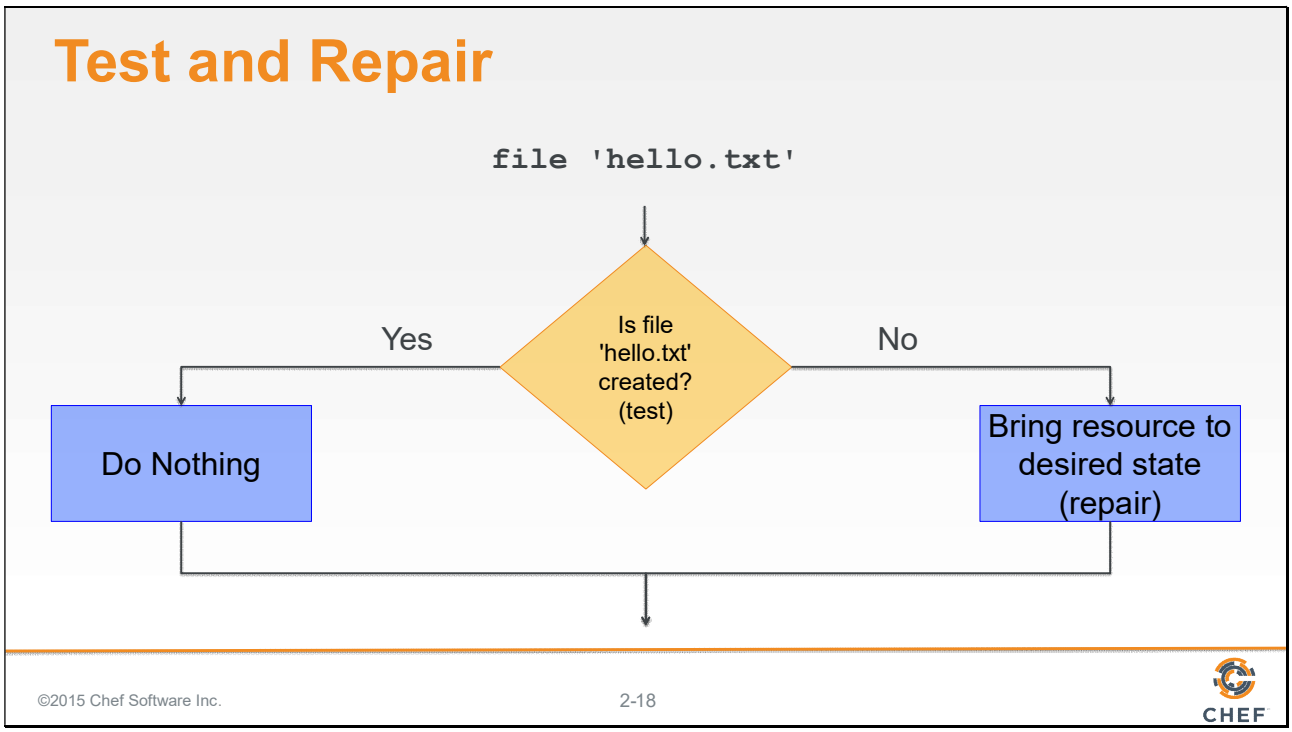
`chef-apply` takes action only when it needs to. Think of it as test and repair.

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

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 18



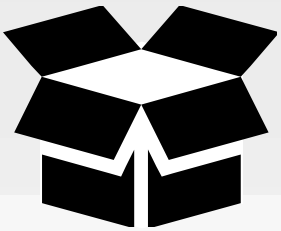
If the file is already created and not modified, then the resource does not need to take action.

If the file is not created, then the resource NEEDS to take action to create the file.
If the file is not in the desire state, then the resource NEEDS to take action to modify the file.

Slide 19

CONCEPT

Resource Definition



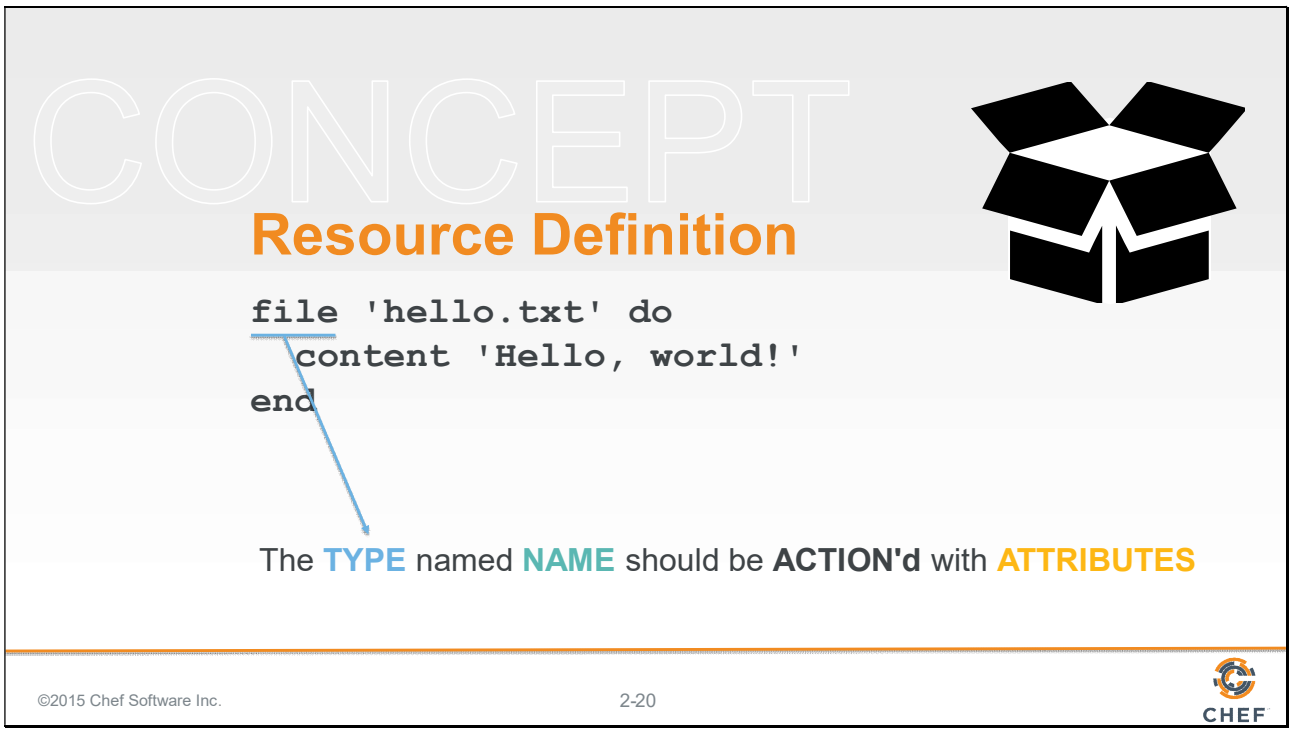
```
file 'hello.txt' do
  content 'Hello, world!'
end
```

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

CHEF

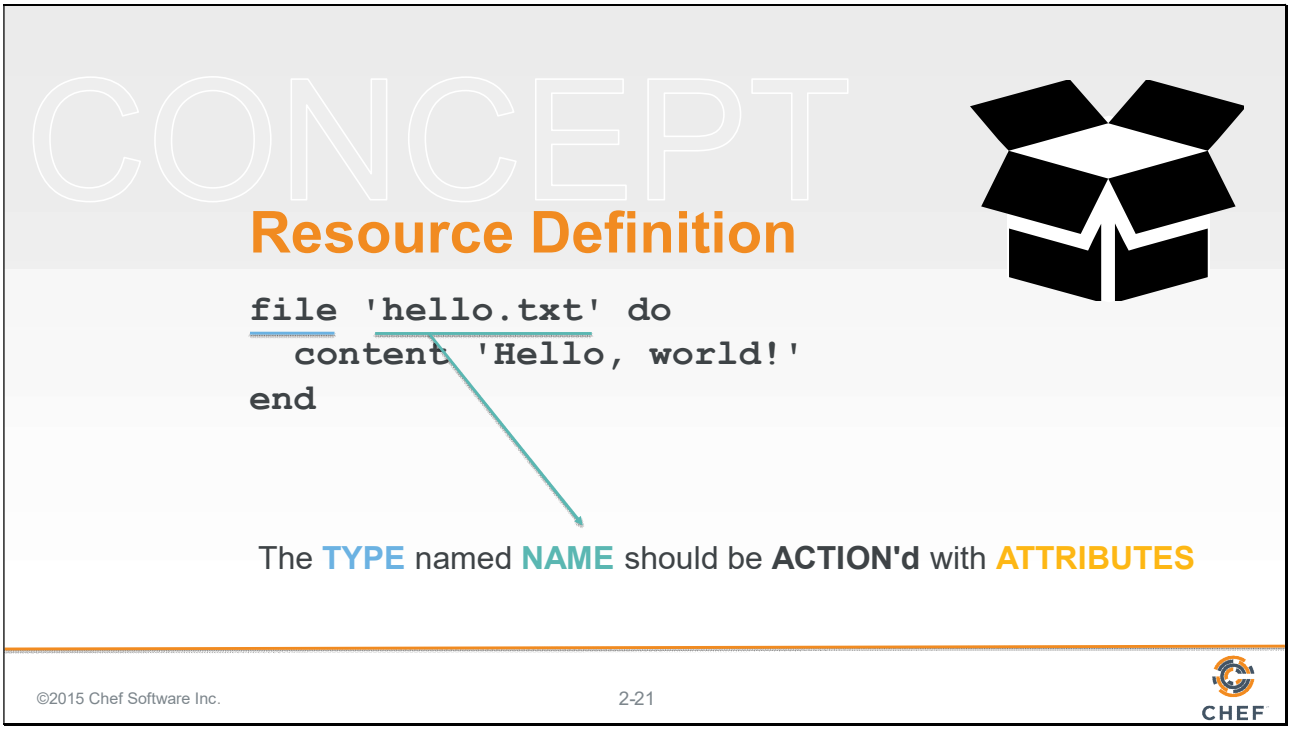
Let's take a moment and talk about the structure of a resource definition. We'll break down the resource that we defined in our recipe file.

Slide 20



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'.

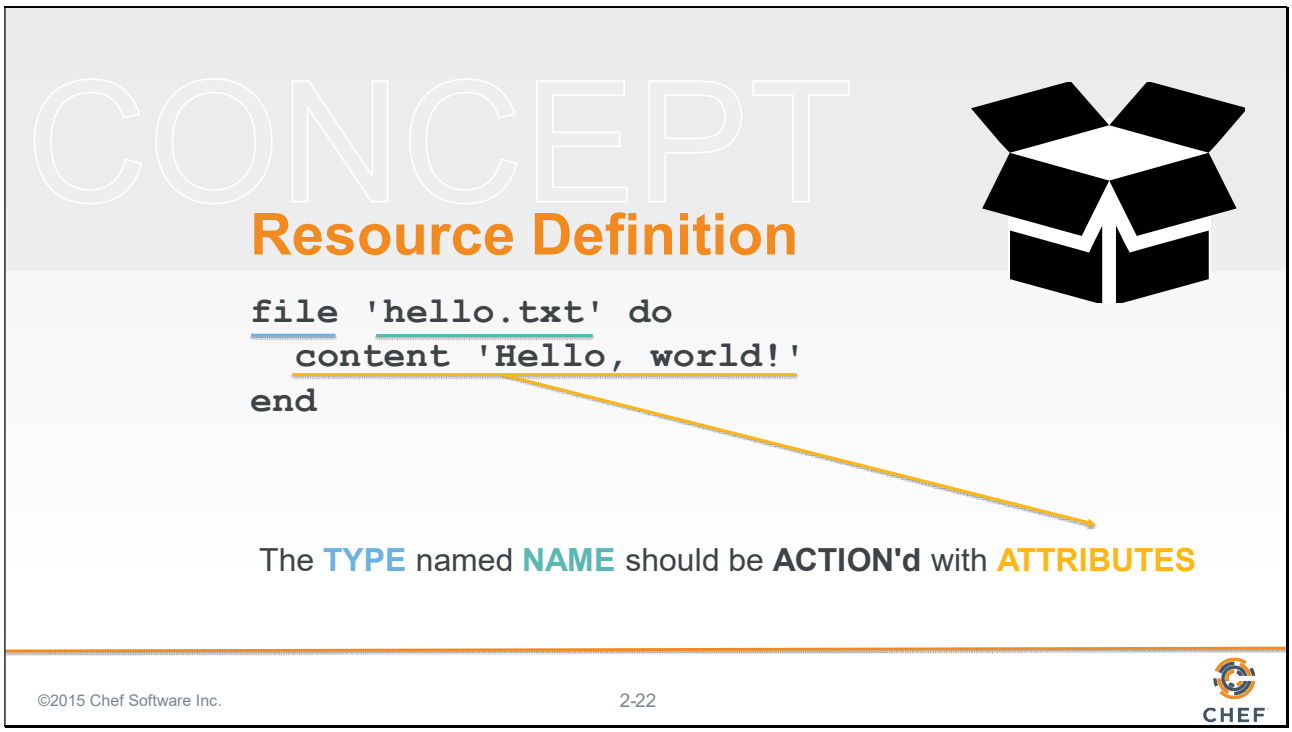
Slide 21



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 22



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.

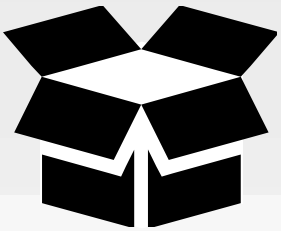
The contents of this block contains attributes (and other things) that help describe the state of the resource. In this instance, the content attribute here specifies the contents of the file.

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

Slide 23

CONCEPT

Resource Definition



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

?

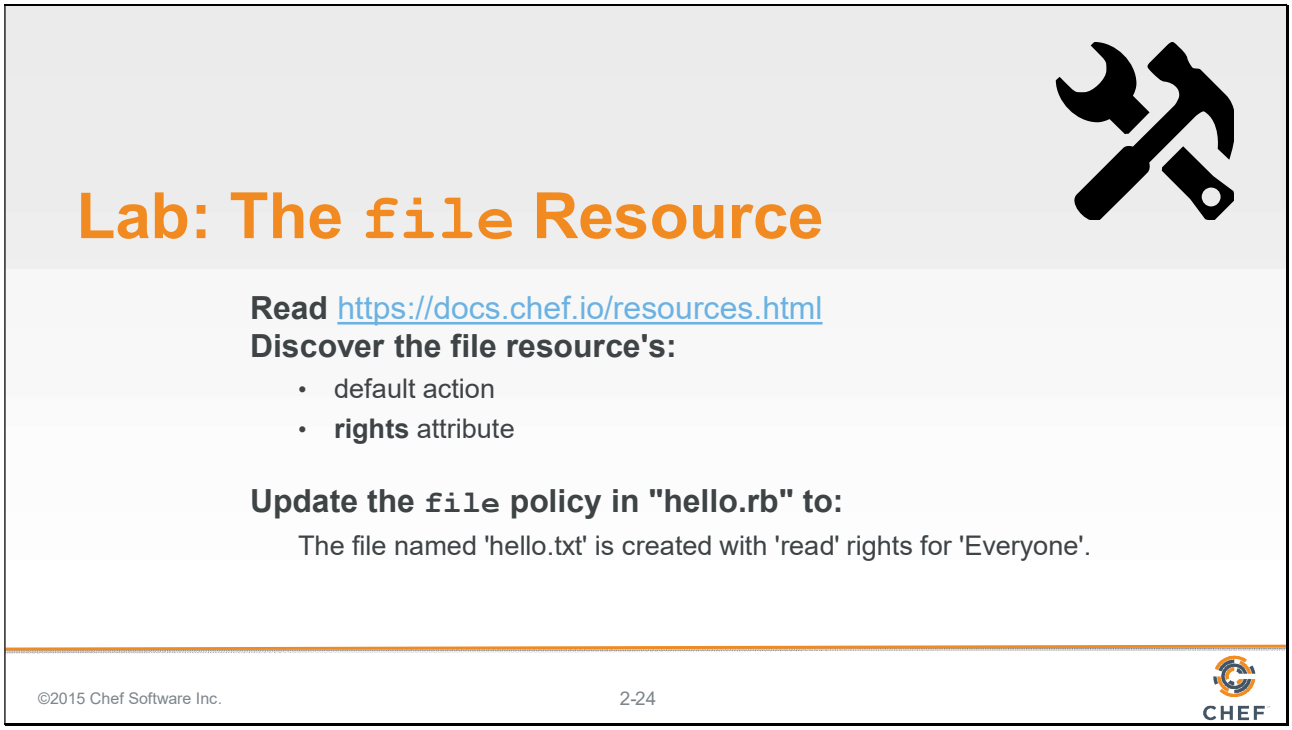
The TYPE named NAME should be **ACTION'd** with ATTRIBUTES

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 24



Find that information in the documentation for the file resource?

- Read through the file Resource documentation.
- Find the list of actions and find the default one.
- Find the list of attributes and find the 'rights' attribute and read a little about it.

The reason for doing this is that we want you to update the file resource in the the recipe file and add the action and add 'read' rights for Everyone.

Instructor Note: Allow 10 minutes to complete this exercise.

Slide 25

Lab: The Updated file Resource

~\hello.rb

```
file 'hello.txt' do
  content 'Hello, world!'
  rights :read, 'Everyone'
  action :create
end
```

This sets the file's rights to allow 'Everyone' access.

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

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.

A file resource's rights attribute supports many different rights values. We want to grant Everyone read access. This will allow users that belong to the 'Everyone' group to read the contents of this file.

Slide 26

Questions

What questions can we answer for you?

Slide 27

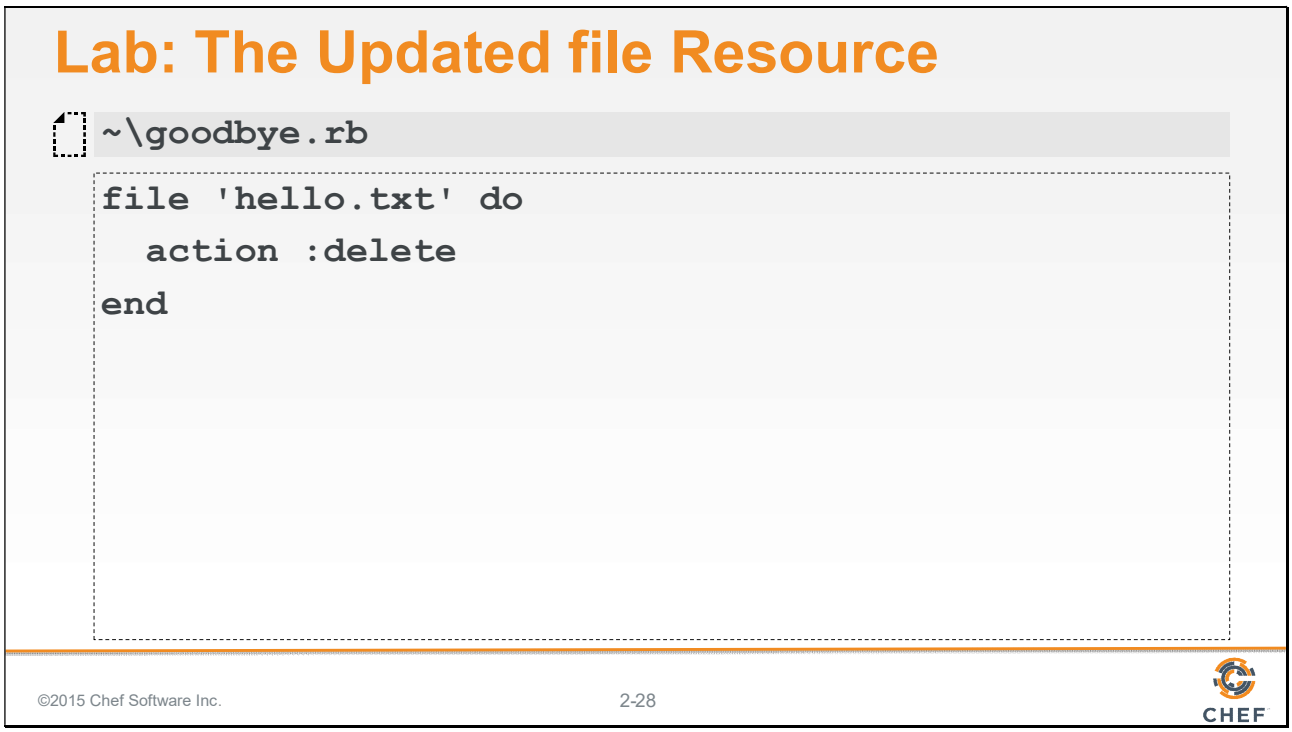
Now that you've practiced:

- 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.

Instructor Note: Allow 10 minutes to complete this exercise.

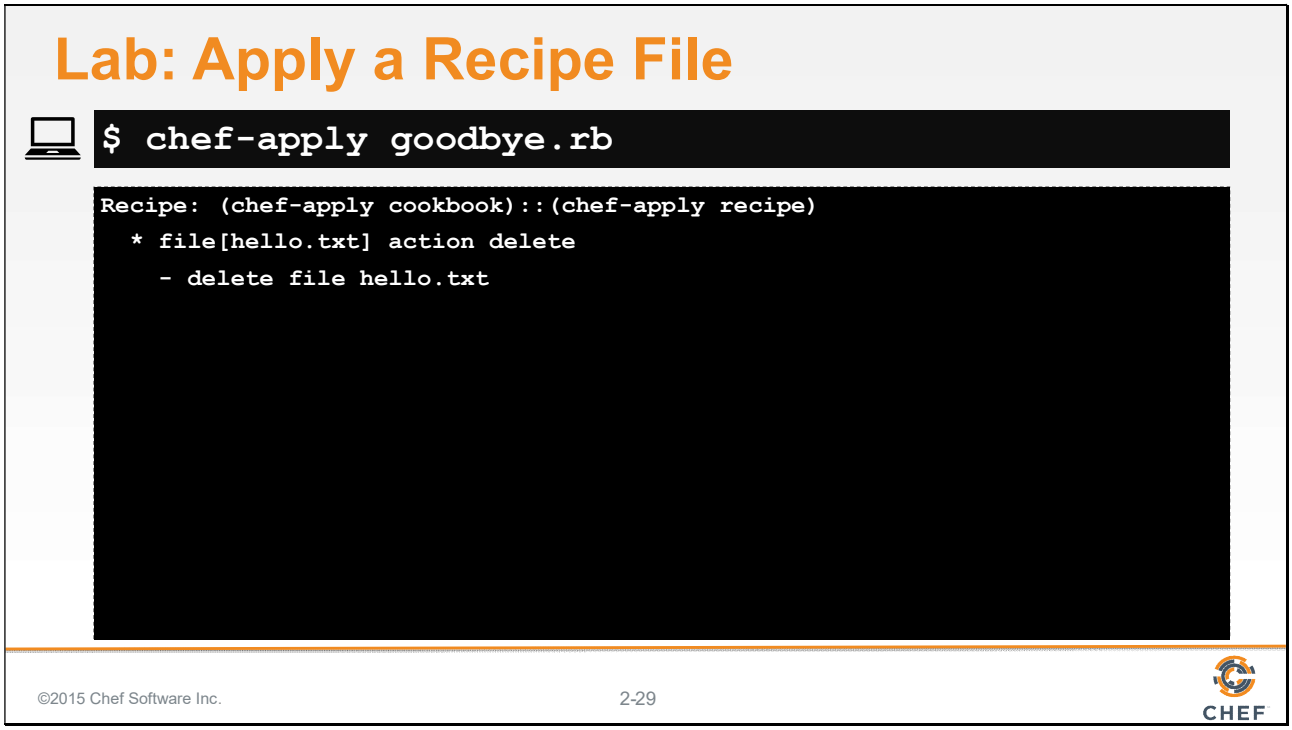
Slide 28



The file resources default action is to create the file. So if we want to remove a file we need to explicitly define the action.

The following policy will delete the 'hello.txt' file when applied.

Slide 29

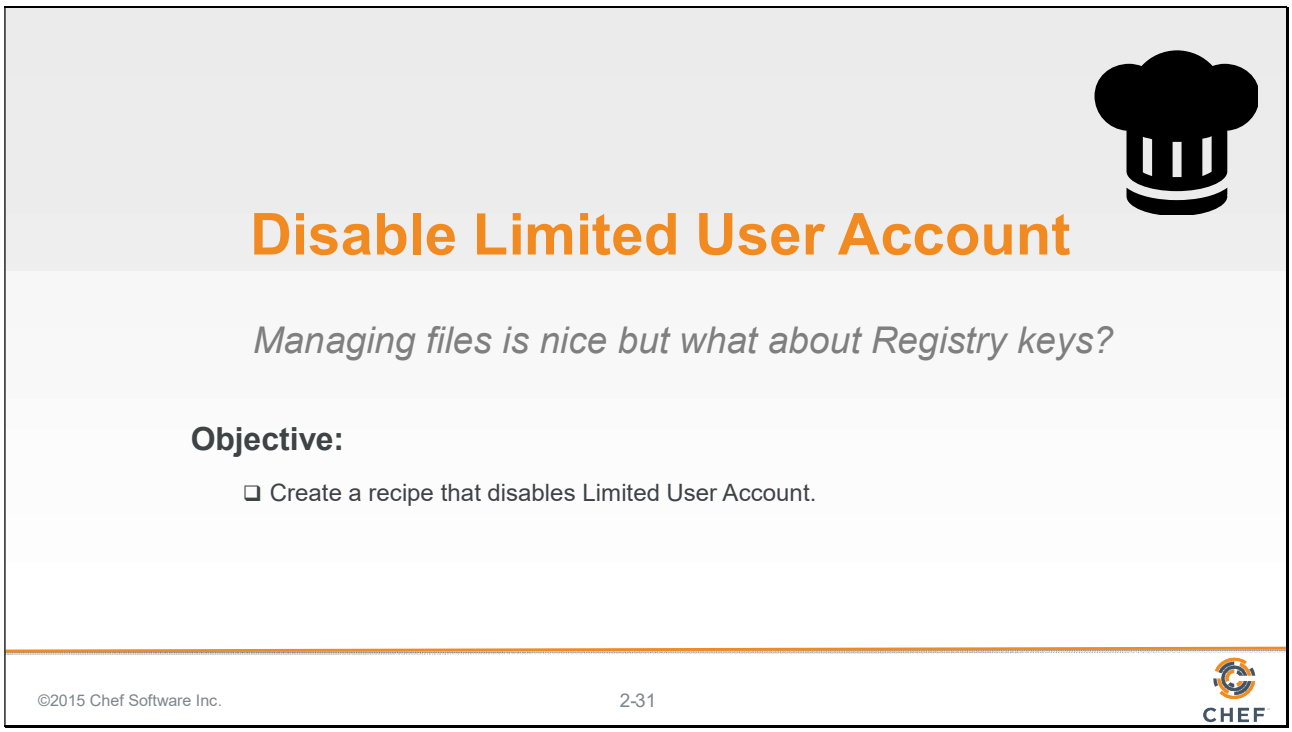


Type the specified command to apply the recipe file. You should see that a file named 'hello.txt' was deleted.

Slide 30

To test that file was removed from the file system successfully we can run the following command.

Slide 31



Managing files is useful but when managing Windows systems we are often more concerned with managing the keys within the registry.

To help setup our system to be more user 'friendly' we want to disable some of the User Access Control (UAC) features that are initially enabled on a Windows system.

Slide 32

GE: Disable the Limited User Account

~\disable-uac.rb

```
system_policies = 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System'

registry_key system_policies do
  values [{
    :name => 'EnableLUA',
    :type => :dword,
    :data => 0
  }]
end
```


Here we are using a new resource named 'registry_key' that takes the name of a registry key. We then provide to the values attribute the values we want to set/insert in the registry. Here we are setting the EnableLUA key to have a dword value of 0. This will make it so that Windows will no longer notify the user when programs try to make changes to the computer. See the following documentation for more information: https://technet.microsoft.com/en-us/library/ff715520.aspx.

Slide 33

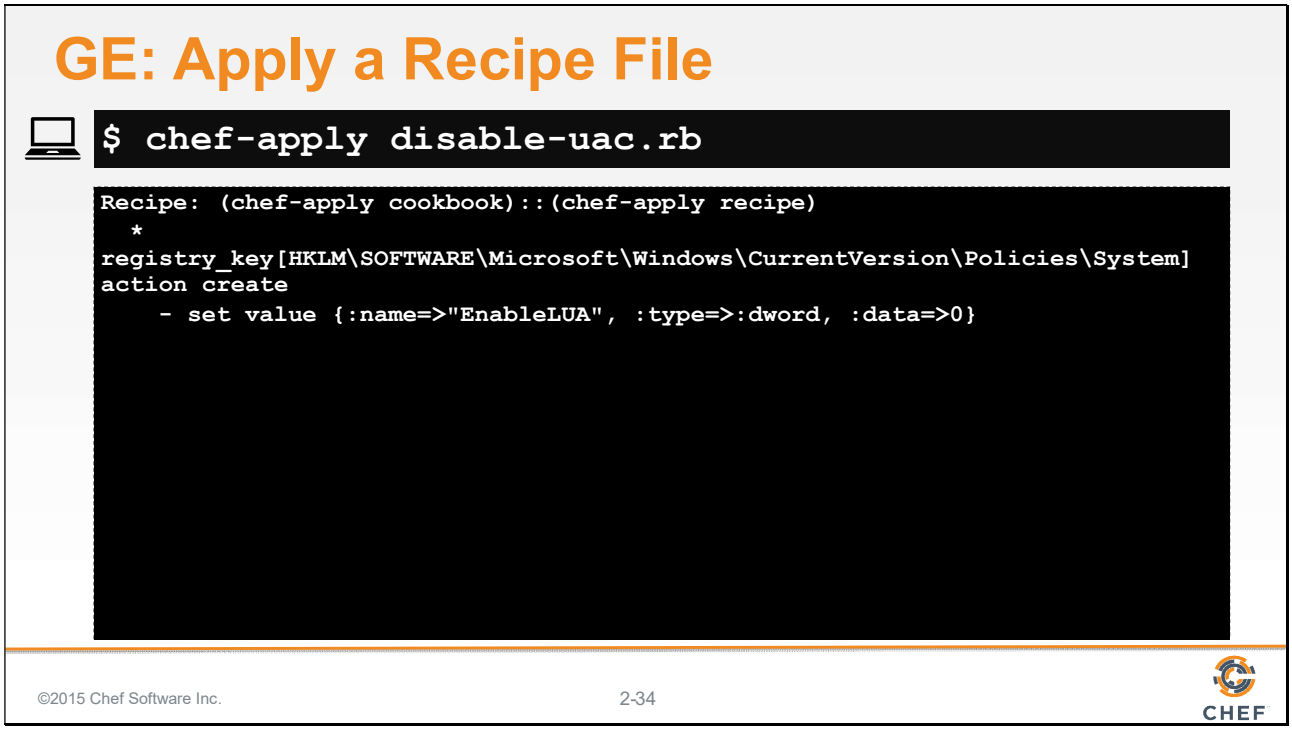
GE: Disable the Limited User Account

~\disable-uac.rb

```
system_policies = 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System'

registry_key system_policies do
  values [{
    :name => 'EnableLUA',
    :type => :dword,
    :data => 0
  }]
end
```


Here we are defining a variable named 'system_policies'. With Ruby you can define variables instantly whenever you need them. Here we define this variable to store our registry key in case we need to use the same registry key to set more values.

Slide 34



Type the specified command to apply the recipe file. This should make a change to the registry key and alert you that you need to restart Windows to disable UAC.

Slide 35

Changing the previous registry key only disables some of UAC. To finish the work return to the recipe file that you created and add another registry resource with the following values.

Slide 36

GE: Disable the Limited User Account

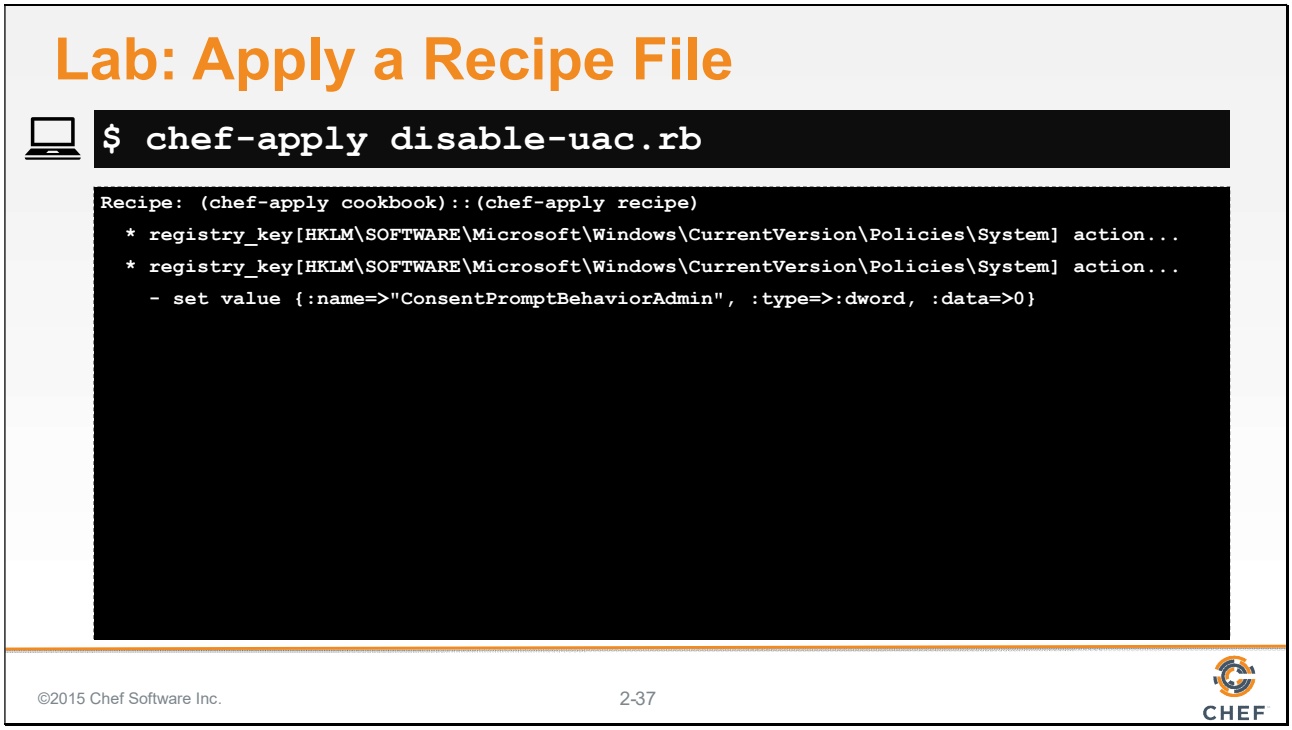
~\disable-uac.rb

```
system_policies = 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System'

registry_key system_policies do
  # ... ENABLE LUA VALUES (NOT SHOWN HERE TO CONSERVE SPACE)
end

registry_key system_policies do
  values [{
    :name => 'ConsentPromptBehaviorAdmin',
    :type => :dword,
    :data => 0
  }]
end
```


CHEF

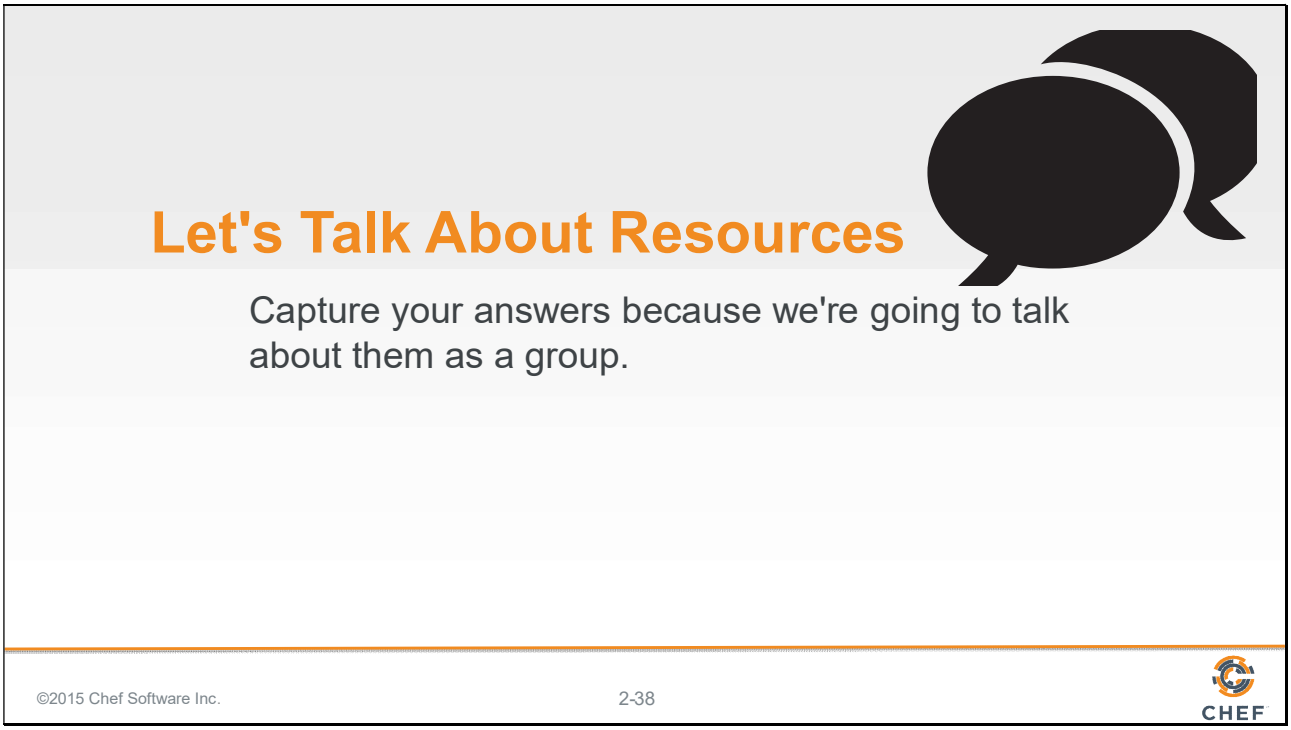
This is the final recipe that contains the two registry keys. This new registry key uses the same variable that we defined before and sets a different values to disable the consent prompt.

Slide 37



Type the specified command to apply the recipe file. The first registry key should report that it is up-to-date. The second registry key will be updated to disable the consent prompt.

Slide 38

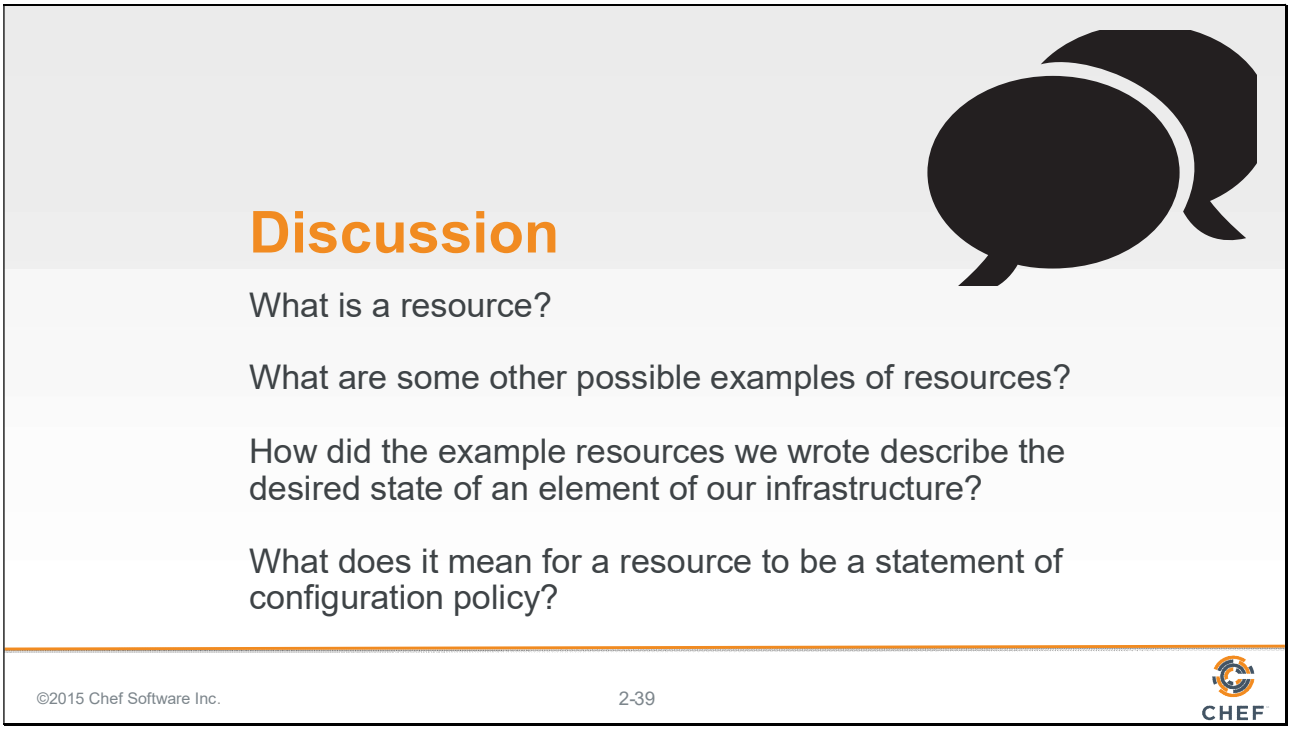


Let's finish this Resources module with a discussion.

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 39

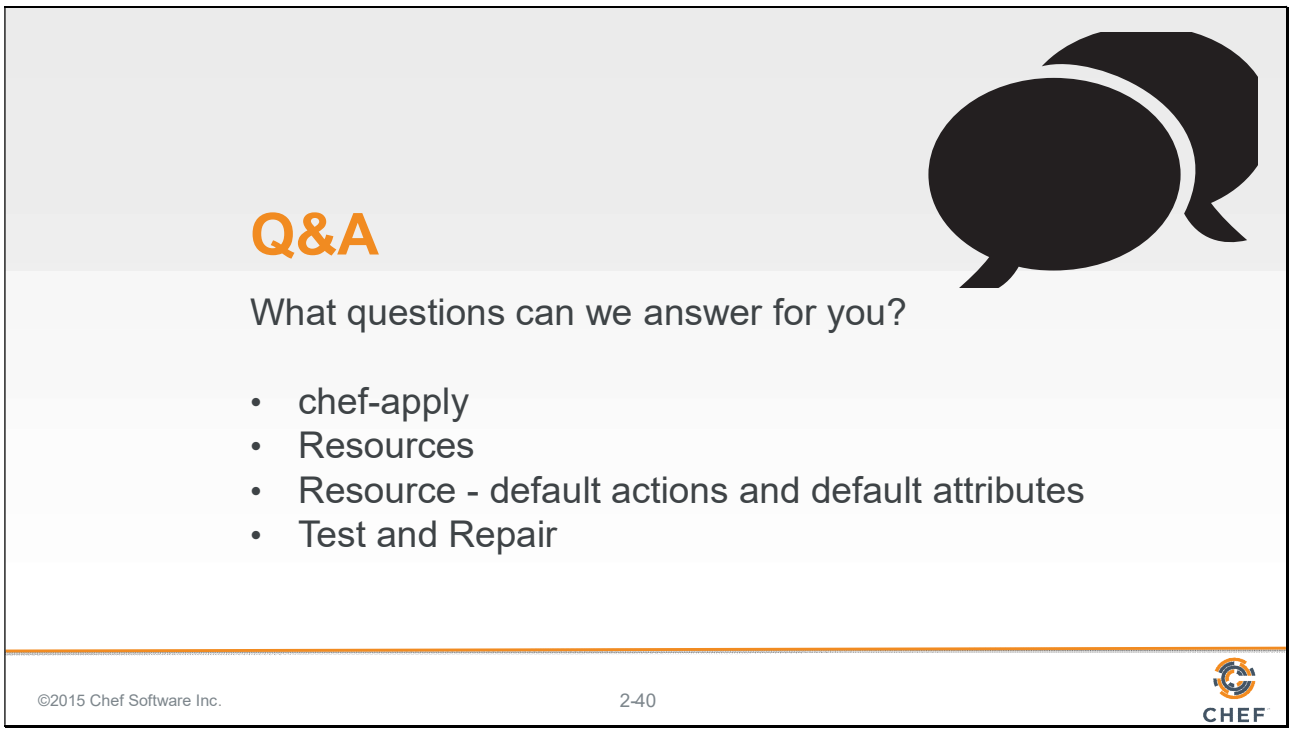


Answer these four questions:

- 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 40



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 41