

## 2: Chef Resources



## Slide 2

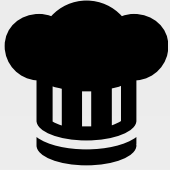
## Objectives

After completing this module, you should be able to:

- Use Chef to install packages on your virtual workstation
- Use the chef-client 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-client' command, create a basic Chef recipe file and define Chef Resources.

## Slide 3




## GL: Time for Some Fun!

*The workstation needs a little personal touch;  
something that makes it a little more fun.*

**Objective:**

- ☐ Write a recipe that installs the 'cowsay' package
- ☐ Apply the recipe to the workstation
- ☐ Use 'cowsay' to say something

---

©2016 Chef Software Inc. 2-3 

I have given you a workstation with a number of tools installed but it is missing something delightful to make the system fun. Together let's walk through using Chef to install the 'cowsay' package.

## Slide 4

## 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.

For those comfortable with Linux distributions it seems rather straight forward to installing packages through the distribution's specific package manager. This is a perfect opportunity to experiment with how to solve configuration problems with Chef. For those not familiar with Linux distributions do not worry, Chef will take care of figuring out those details for us when it comes time to do the installation of the package.

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 5

## Choose an Editor

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

*Tips for using these editors can be found below in your participant guide.*

**emacs**

**nano**

**vi / vim**

During this course we are going to use the text-based editors installed on these virtual workstations. There are at least three command-line editors that we can choose from on the Linux workstation: Emacs, Nano, or Vim.

**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 6



The slide features a light gray background. In the top left, the word "DOCS" is written in large, white, outlined letters, with "Resources" in orange below it. In the top right, there is a black icon of three books. The main text is centered and reads: "A resource is a statement of configuration policy." followed by "It describes the desired state of an element of your infrastructure and the steps needed to bring that item to the desired state." Below this is a blue hyperlink: <https://docs.chef.io/resources.html>. At the bottom, there is a thin orange line, and below that, the copyright notice "©2016 Chef Software Inc.", the page number "2-6", and the Chef logo.

# DOCS

## Resources

A resource is a statement of configuration policy.

It describes the desired state of an element of your infrastructure and the steps needed to bring that item to the desired state.

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

©2016 Chef Software Inc. 2-6 

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 7

## Example: Package

```
package 'httpd' do  
  action :install  
end
```

The package named 'httpd' is installed.

[https://docs.chef.io/resource\\_package.html](https://docs.chef.io/resource_package.html)

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

## Slide 8

## 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](https://docs.chef.io/resource_service.html)

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



# Slide 9

## Example: File

```
file '/etc/motd' do
  content 'This computer is the property ...'
end
```

The file name '/etc/motd' is created with content 'This computer is the property ...'

[https://docs.chef.io/resource\\_file.html](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 10

## 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](https://docs.chef.io/resource_file.html)

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

## Slide 11

## GL: Use Your Editor to Open the Recipe



```
$ nano setup.rb
```

©2016 Chef Software Inc. 2-11 

Now that we have seen a few examples of resources let's get to work installing that package. Using your editor of choice open up a file named 'setup.rb'.

## Slide 12

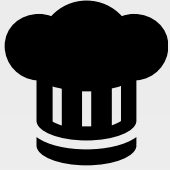
## GL: Update the Setup Recipe

 ~/setup.rb

```
package 'cowsay' do  
  action :install  
end
```

In the file add the following resource to install the 'cowsay' package.

## Slide 13




## GL: Time for Some Fun!

*The workstation needs a little personal touch;  
something that makes it a little more fun.*

**Objective:**

- ✓ Write a recipe that installs the 'cowsay' package
- ❑ Apply the recipe to the workstation
- ❑ Use 'cowsay' to say something

---

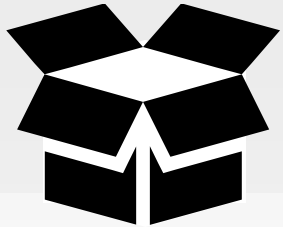
©2016 Chef Software Inc. 2-13 

Save the file and return back to the shell. It is now time to apply the recipe to the workstation.

## Slide 14

# 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](https://docs.chef.io/chef_client.html)

---

©2015 Chef Software Inc.2-14

In the Chef Development Kit (ChefDK), we package a tool that is called 'chef-client'.

'chef-client' is a command-line application that can be used to apply a recipe file. 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 15

# CONCEPT

## --local-mode (or -z)


chef-client's default mode attempts to contact a Chef Server and ask it for the recipes to run for the given node.

We are overriding that behavior to have it work in a local mode.



©2015 Chef Software Inc.


4-15




'chef-client' has the default default behavior to communicate with a Chef server. So we use the '--local-mode' flag to ask 'chef-client' to look for the recipe file locally.

## Slide 16

## GL: Apply the Setup Recipe

 `$ sudo chef-client --local-mode setup.rb`

```
Starting Chef Client, version 12.5.1
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Compiling Cookbooks...
[2016-02-19T13:08:13+00:00] WARN: Node ip-172-31-12-176.ec2.internal has an empty run list.
Converging 1 resources
Recipe: @recipe_files::/home/chef/setup.rb
  * yum_package[nano] action install
    - install version 3.03-8.el6 of package cowsay
Running handlers:
Running handlers complete
Chef Client finished, 1/1 resources updated in 38 seconds
```

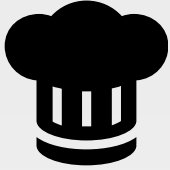
©2015 Chef Software Inc. 2-16 

Execute the following command to have chef-client apply the recipe file. Because we are installing a package the prefix 'sudo' is necessary. This ensures that we have elevated our permissions to the appropriate level to install the package.

In the output you should see Chef installing the appropriate package.



## Slide 17




## GL: Time for Some Fun!

*The workstation needs a little personal touch;  
something that makes it a little more fun.*

**Objective:**

- ✓ Write a recipe that installs the 'cowsay' package
- ✓ Apply the recipe to the workstation
- ❑ Use 'cowsay' to say something

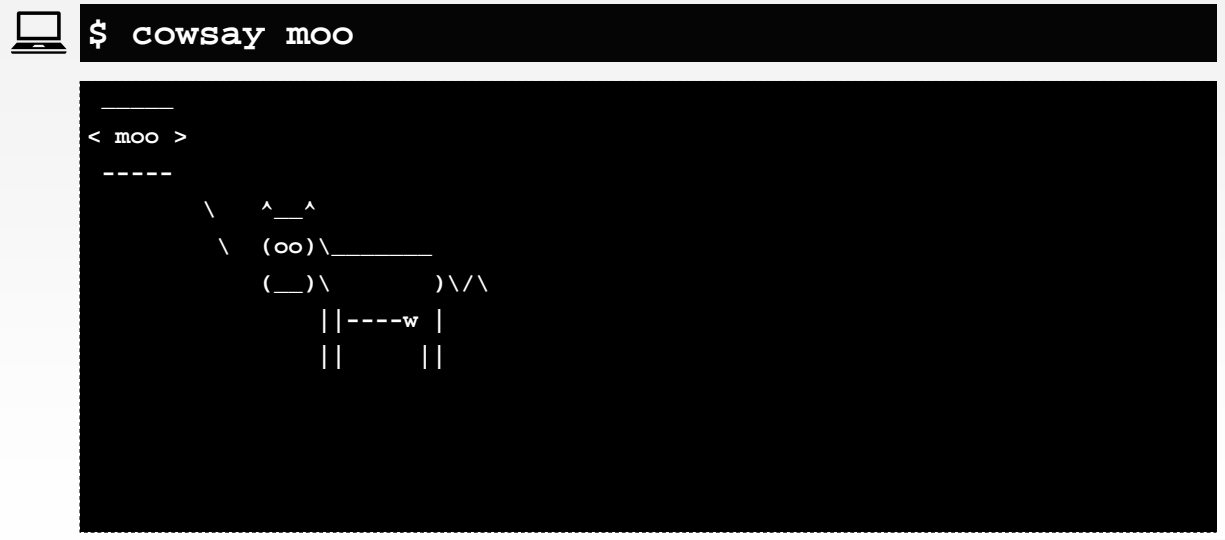
---

©2016 Chef Software Inc. 2-17 

With the package installed it is time to use it.


## Slide 18

## GL: Run cowsay with a Message



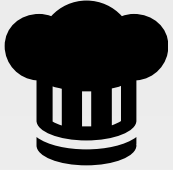
```
$ cowsay moo
```

```
< moo >
-----
      \   ^__^
      \  (oo)\_______
         (__)\\       )\/\
              ||----w |
              ||     ||
```

©2016 Chef Software Inc. 2-18 

Run cowsay and give it a parameter or a few parameters. Enjoy your new bovine friend that will parrot back what you type into the shell.

## Slide 19




## GL: Time for Some Fun!

*The workstation needs a little personal touch;  
something that makes it a little more fun.*

**Objective:**


- ✓ Write a recipe that installs the 'cowsay' package
- ✓ Apply the recipe to the workstation
- ✓ Use 'cowsay' to say something

---

©2016 Chef Software Inc. 2-19 

In this exercise we wrote a resource in a recipe file and applied that recipe file to the workstation. More importantly we brought a little more fun to our workstation.


## Slide 20



## Discussion

1. What would happen if you applied the recipe again?
2. What would happen if the package were to become uninstalled?

---

©2016 Chef Software Inc. 2-20 

What would happen if you applied the recipe again? Before you execute the command to apply the recipe 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 applied the recipe again? Was there a situation where the package was already uninstalled and we applied this recipe?

## Slide 21

# CONCEPT

## Test and Repair



`chef-client` 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.

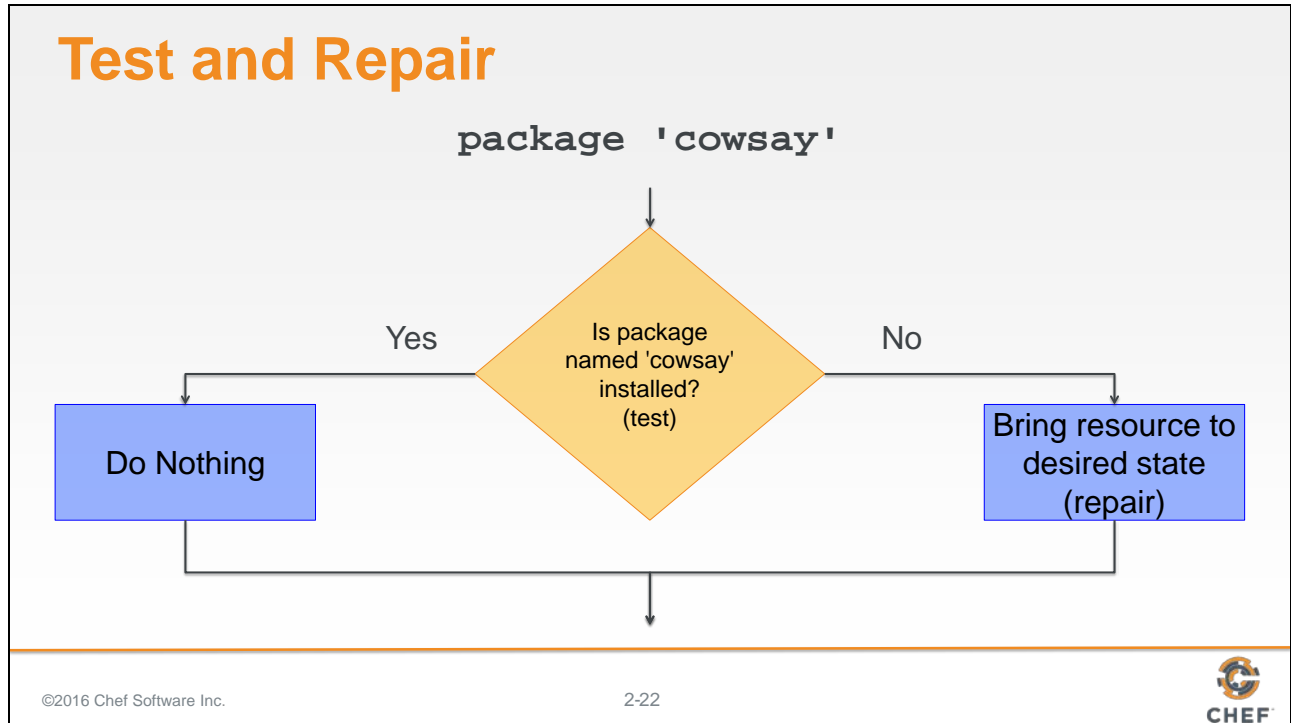
---

©2016 Chef Software Inc. 2-21 

Hopefully it is clear from running the ``chef-client`` 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.

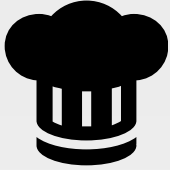
## Slide 22



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 23




## GL: Hello, World?

*I heard Chef is written in Ruby. If that's the case its required that we write a quick "Hello, world!" application.*

**Objective:**

- ☐ Create a recipe that writes out a file with the contents "Hello, world!"
- ☐ Apply that recipe to the workstation
- ☐ Verify the contents of the file

---

©2016 Chef Software Inc. 2-23 

Great! You installed a package with ``chef-client`` 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 24

## GL: Create and Open a Recipe File



A terminal window with a black background and white text. The prompt is '\$' followed by the command 'nano hello.rb'. The terminal is open, showing a large black rectangular area representing the editor's workspace.

©2016 Chef Software Inc. 2-24 

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 25

## GL: Create a Recipe File Named hello.rb

 ~/hello.rb

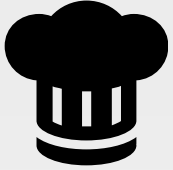
```
file 'hello.txt' do
  content 'Hello, world!'
end
```

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!'.

## Slide 26




## GL: Hello, World?

*I heard Chef is written in Ruby. If that's the case its required that we write a quick "Hello, world!" application.*

**Objective:**

- ✓ Create a recipe that writes out a file with the contents "Hello, world!"
- ❑ Apply that recipe to the workstation
- ❑ Verify the contents of the file


---

©2016 Chef Software Inc. 2-26 


Now that we have created the recipe file it is time to apply it.

## Slide 27

## GL: Apply the Recipe File

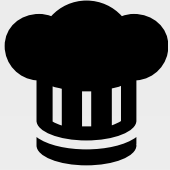
 `$ sudo chef-client --local-mode hello.rb`

```
Starting Chef Client, version 12.5.1
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Compiling Cookbooks...
[2016-02-19T13:08:13+00:00] WARN: Node ip-172-31-12-176.ec2.internal has an empty run list.
Converging 1 resources
Recipe: @recipe_files::/home/chef/hello.rb
  * file[hello.txt] action create
    - create new file hello.txt
    - update content in file hello.txt from non to 315f5b
    +++ ./hello.txt20160224-8559-19kqial
        2016-02-24 16:51:04.400844959 +0000
    @@ -1 +1,2 @@
    +Hello, world!
```

©2015 Chef Software Inc. 2-27 

Using `chef-client` with the local mode flag we specify the new recipe file and apply it to the system. In this instance we are creating a file locally within the current directory and do not actually need to use the 'sudo' prefix.

## Slide 28




## GL: Hello, World?

*I heard Chef is written in Ruby. If that's the case its required that we write a quick "Hello, world!" application.*

**Objective:**

- ✓ Create a recipe that writes out a file with the contents "Hello, world!"
- ✓ Apply that recipe to the workstation
- ❑ Verify the contents of the file

---

©2016 Chef Software Inc. 2-28 

In the output it looks like the recipe we applied to the system created a hello.txt file. Now it is time to examine the contents.

## Slide 29

## GL: What Does hello.txt Say?



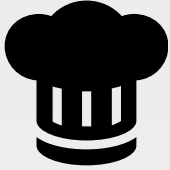
A terminal window icon is shown next to the command prompt. The command `$ cat hello.txt` is entered. The output `Hello, world!` is displayed on the next line. The terminal window has a black background with white text. The output line is highlighted with a brown background.

©2016 Chef Software Inc. 2-29



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 30




## GL: Hello, World?

*I heard Chef is written in Ruby. If that's the case its required that we write a quick "Hello, world!" application.*

**Objective:**

- ✓ Create a recipe that writes out a file with the contents "Hello, world!"
- ✓ Apply that recipe to the workstation
- ✓ Verify the contents of the file

---

©2016 Chef Software Inc. 2-30 

Great. Again we created a recipe file with a resource and applied it to the system. This time it was a file and not a package but we can start to see that with Chef there are many different resources that we can use to express the desired state of the system.

## Slide 31



## Discussion

What would happen if the 'hello.txt' file contents were modified?


---

©2016 Chef Software Inc. 2-31 

Similar to the discussion we had before it is important to reflect on what would happen in this case with a file. What would happen if the contents of the target file were to change? How would 'chef-client' handle that situation? What would the output look like compared to when it created a file?

I encourage you to take a guess, make the change, and then apply the recipe again.

Slide 32



## Test and Repair


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

Have we defined a policy for these attributes?

---

©2016 Chef Software Inc.

2-32



What would happen if the file permissions, owner or group of the file changed? Did we define our desired policy for those attributes?



## Slide 33

# CONCEPT

## Resource Definition

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

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



---

©2016 Chef Software Inc.

2-33



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.

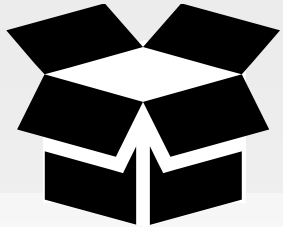
## Slide 34

# CONCEPT

## Resource Definition

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


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



---

©2016 Chef Software Inc.

2-34



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'.

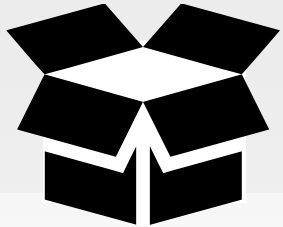
## Slide 35

# CONCEPT

## Resource Definition

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


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



---

©2016 Chef Software Inc.

2-35



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.

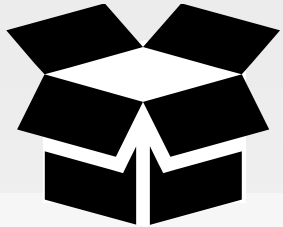
## Slide 36

# CONCEPT

## Resource Definition


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

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



©2016 Chef Software Inc.

2-36



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 37

# CONCEPT

## Resource Definition

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

?

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



---

©2016 Chef Software Inc.


2-37



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 38




## Lab: The `file` Resource

- ☐ Read <https://docs.chef.io/resources.html>
- ☐ Discover the `file` resource's:
  - default action.
  - default values for `mode`, `owner`, and `group`.
- ☐ Update the `file` policy in "`hello.rb`" to:

The file named '`hello.txt`' should be created with the content '`Hello, world!`', mode '`0644`', owner is '`root`', and group is '`root`'.

---


©2016 Chef Software Inc. 2-38 

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`.

## 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).

©2016 Chef Software Inc. 2-39 

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 40




## Lab: The `file` Resource

- ✓ Read <https://docs.chef.io/resources.html>
- ✓ Discover the `file` resource's:
  - default action.
  - default values for `mode`, `owner`, and `group`.
- ✓ Update the `file` policy in "`hello.rb`" to:

The file named '`hello.txt`' should be created with the content '`Hello, world!`', mode '`0644`', owner is '`root`', and group is '`root`'.

---

©2016 Chef Software Inc. 2-40 

You successfully updated the file resource to include the attributes and being explicit with the action. You have demonstrated the important part of reading the documentation and taking action to meet the defined requirements.




Slide 41

## Questions

What questions can we answer for you?



## Slide 42



## Lab: Workstation Setup

- ❑ Create a recipe file named "setup.rb" that defines the policy:
  - The package named 'cowsay' is installed.
  - The package named 'tree' is installed.
  - The file named '/etc/motd' is created with the content 'Property of ...'.
- ❑ Use chef-client to apply the recipe file named "setup.rb"

©2016 Chef Software Inc. 2-42 

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 43

## Lab: Workstation Setup Recipe File



```
~/setup.rb
```

```
package 'cowsay' do
  action :install
end
```

The package named 'cowsay' is installed.

```
package 'tree' do
  action :install
end
```

The package named 'tree' is installed.


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

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

Here is a version of the recipe file that installs cowsay, tree, and creates the message-of-the-day file.


## Slide 44

## GL: Apply the Recipe File

 **\$ sudo chef-client --local-mode setup.rb**


```
Converging 3 resources
Recipe: @recipe_files::/home/chef/setup.rb
* yum_package[cowsay] action install (up to date)
* yum_package[tree] action install
  - install version 1.5.3-3.el6 of package tree
* file[/etc/motd] action create
  - update content in file /etc/motd from e3b0c4 to d100eb
--- /etc/motd      2010-01-12 13:28:22.000000000 +0000
+++ /etc/.motd20160224-8754-1xczeyn 2016-02-24 16:57:57.203844958 +0000
@@ -1,2 @@
+Property of ...

Running handlers:
Running handlers complete
Chef Client finished, 2/3 resources updated in 17 seconds
```

©2015 Chef Software Inc. 2-44 

Applying it is the same as we did before with chef-client.


## Slide 45



## Lab: Workstation Setup


- ✓ Create a recipe file named "setup.rb" that defines the policy:
  - The package named 'cowsay' is installed.
  - The package named 'tree' is installed.
  - The file named '/etc/motd' is created with the content 'Property of ...'.
- ✓ Use chef-client to apply the recipe file named "setup.rb"

©2016 Chef Software Inc. 2-45



Wonderful. The setup recipe now installs something fun, useful, and configures something important on our system. We can use cowsay throughout the rest of the course to give ourselves a little chuckle. We will use tree in the upcoming sections to help us understand all the folder structure of things we will develop. And the message of the day we configured will greet us with the important property line the next time we or someone else logs into the system.

## Slide 46



## Discussion


What is a resource?

What are some other possible examples of resources?

How did the example 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?

---

©2016 Chef Software Inc. 2-46 

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

## Slide 47




## Q&A

What questions can we answer for you?

- chef-client
- Resources
- Resource - default actions and default attributes
- Test and Repair

---

©2016 Chef Software Inc. 2-47 

What questions can we answer for you?

About anything or specifically about:

- chef-client
- resources
- a resources default action and default attributes
- Test and Repair

Slide 48

