



Chef Infra Foundations

Introduction



1L

Course v6.0.2

1-1

This course provides a basic understanding of Chef's Infra's core components, basic architecture, commonly used tools, and basic troubleshooting methods for both windows and Linux platforms.

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

Instructor Note: Be sure to read Appendix Z in the instructor kit for training lab set up notes and additional instructor notes. The "1L" on this slide means you will need 1 Linux WS for each student in this module.



The illustration features a woman with dark hair tied back, wearing a pink long-sleeved top and a dark blue knee-length skirt. She is holding a white laptop in her left hand and gesturing with her right hand towards a yellow lightbulb. The background is a blue gradient with abstract shapes like a pink circle, a blue cloud-like shape, and a yellow circle with a dotted pattern. The Progress Chef logo is visible at the bottom left of the illustration area.

Introductions

Let's get to know each other and the training.

- Name
- Current job role
- Previous job roles/background
- Experience with Chef and/or config management
- Favorite text editor
- Location
- Expectations

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

1- 2

Before we get started with this training let's take a moment to get acquainted with each other and with the content that we are going to be exploring.

Course Objectives

You will leave this class with a basic understanding of Chef's core components, architecture, and commonly used tools. After completing this course, you should be able to:

- Write Chef recipes with Chef Resources that model the desired state of a system
- Manage these recipes in cookbooks that you are able to apply to a system
- Test cookbooks with linting tools and Test Kitchen
- Add multiple nodes to be managed by a Chef Infra Server

We will be focusing on creating recipes that contain the necessary resources to define the desired state of our systems. We will be managing these systems via a Chef Server through the use of Policyfiles.

Course Objectives continued

After completing this course, you should be able to:

- Explain Chef Automate and Chef Infra Server usage
- Deploy a load balancer to distribute traffic to nodes
- Manage the deployment of cookbooks to nodes with Policyfiles

Agenda: Day 1

- Using Chef Resources
- Building Chef Cookbooks
- Ohai
- Using Test Kitchen
- Templates

Agenda: Day 2

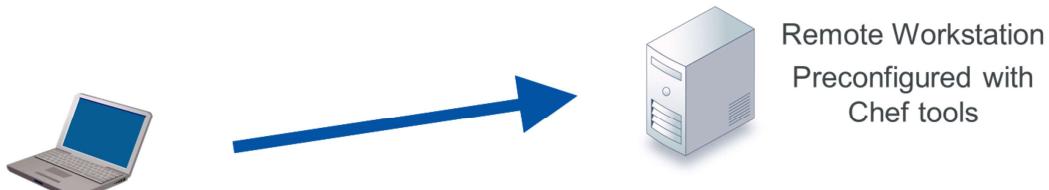
- Accessing Chef Infra & Chef Automate
- Bootstrap a node
- Policyfiles
- Attribute Files and Dependencies

Agenda: Day 3

- Using Policyfiles to define roles
- Use Search within a recipe
- Set up chef-client to run as a service/task
- Use policy_group to create environments
- Workstation Installation
- Further Resources

Pre-built Workstation

- We will provide you a workstation with all the tools installed.
- Login to the Remote Workstation

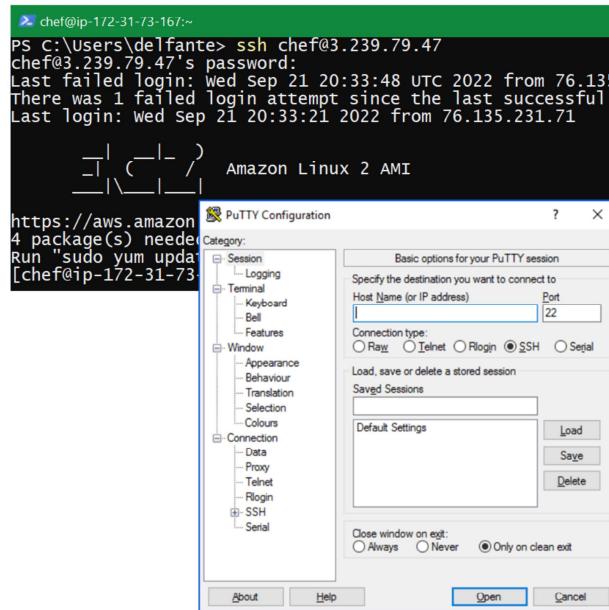


Hands-on Legend

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

GL: Log in to the workstation

You can use a MacOS/Linux terminal, Windows Powershell, or PuTTY to connect via ssh to your workstation.



 Progress Chef®

1-10

We will provide you with the address, username and password of the workstation. With that information you will need to use a Remote Desktop Connection tool that you have installed to connect that workstation.

Instructor Note: Assign the participants their Day 1 virtual workstations at this time. The login credentials and password for the virtual workstations are:

user: chef
password: Cod3Can!

GL: Log into your assigned workstation



```
> ssh chef@IPADDRESS
```

```
chef@3.239.79.47's password:
```

The instructor will provide you with your assigned IP address and password.

```
Last login: Wed Sep 21 20:07:14 2022
```



```
Amazon Li
```

In the next steps you will setup Visual Studio Code so you can more easily write code without needing to use Linux editors like vi/vim, emacs, or nano.

```
https://aws.amazon.com/amazon
```

```
4 package(s) needed for security, out of 13 available  
Run "sudo yum update" to apply all updates.
```

GL: Install Visual Studio Code Using Script



```
> ./vscode.sh
```

```
***** Clean up existing installations *****
***** Setting Up VS CODE *****
***** Starting VS CODE Server *****
You can now access vs code at http://3.95.226.21:8000
password is : yj0aBQHzhQ0by
```

Type `./vscode.sh` to install VS Code.

Then copy the password.

```
[chef@ip-172-31-77-66 ~]$ ./vscode.sh
```

```
***** Clean up existing installations *****
***** Setting Up VS CODE *****
***** Starting VS CODE Server *****
You can now access vs code at http://3.95.226.21:8000
password is : yj0aBQHzhQ0by
```

Note: If during this class you should lose connection to VS Code and pointing your browser to its IP address again does not resolve it, simply run this script again as follows:

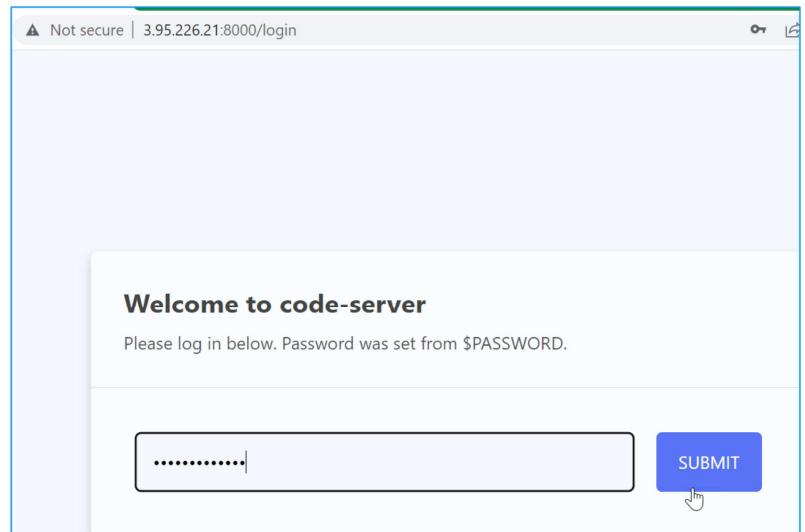
`./vscode.sh`

GL: Accessing Visual Studio Code

Point a web browser to the URL that was output by the script. For example:

<http://3.95.226.xx:8000>

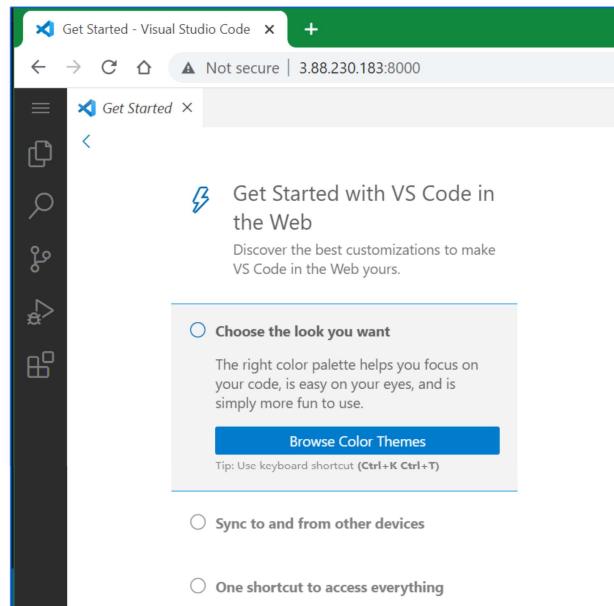
Paste or type the password when prompted and click Submit



GL: Accessing Visual Studio Code

Visual Studio Code will display.

Visual Studio Code makes it easy to write and edit code without the need to use Linux text editors.



 Progress Chef®

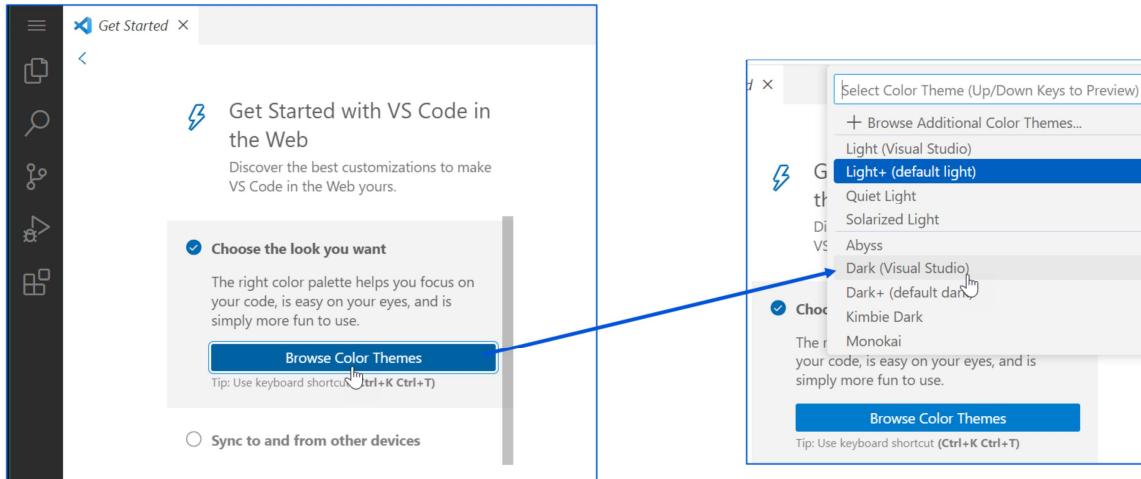
© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

1-14

You can now access vs code at <http://IPADDRESS:8000/?tkn=52e5c553-d02e-4cd7-934c-500aa57b40e9>

GL: Modifying Visual Studio Code

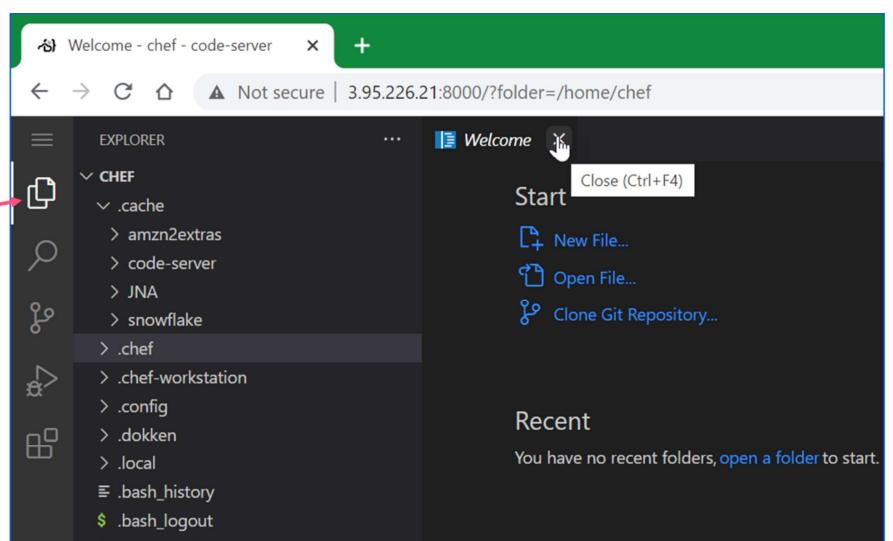
If you like you can set the look of VS Code. For example:



GL: Using Visual Studio Code

Close the Welcome tab.

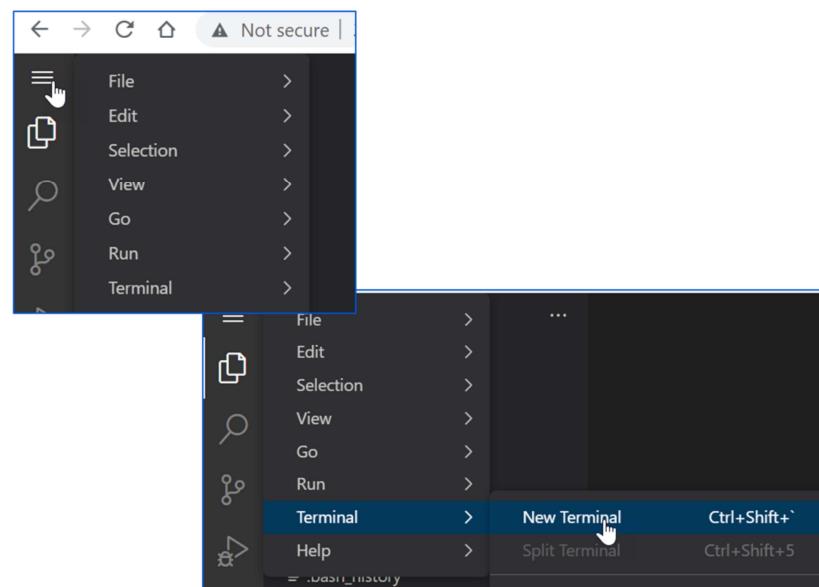
The folder should be open to /home/chef



GL: Using VS Code

From the main menu icon, click **Terminal > New Terminal** to open a terminal.

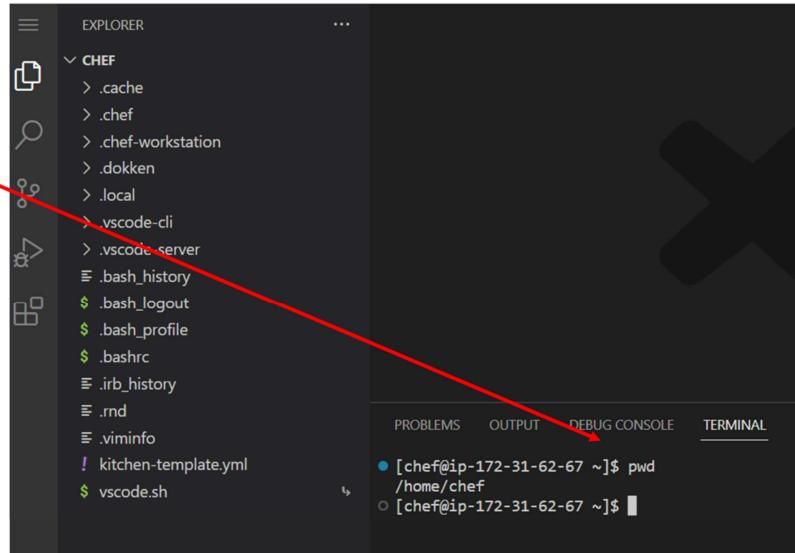
The terminal will open at the bottom of the VS Code UI.



GL: Using VS Code

From the VS Code terminal,
type `pwd`

...and ensure you are in
`/home/chef/`



The screenshot shows the VS Code interface. The left side features the Explorer pane, which is expanded to show a 'CHEF' folder containing various files and folders like '.cache', '.chef', '.chef-workstation', '.dokken', '.local', 'vscode-cli', 'vscode-server', '.bash_history', '.bash_logout', '.bash_profile', '.bashrc', '.irb_history', '.rnd', '.viminfo', 'kitchen-template.yml', and 'vscode.sh'. The right side features the Terminal pane, which displays the command output: `[chef@ip-172-31-62-67 ~]$ pwd /home/chef [chef@ip-172-31-62-67 ~]$`. A red arrow points from the text '...and ensure you are in /home/chef/' towards the terminal output.

GL: Using VS Code

In this class you can write and edit code in the top right portion of VS Code.

You can execute various commands in the bottom-right terminal section.

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

```
[chef@ip-172-31-73-167 ~]$ chef --help
The Chef command line tool for managing your infrastructure
  . . .
  Docs: https://docs.chef.io/workstation/
  Patents: https://www.chef.io/patents
```

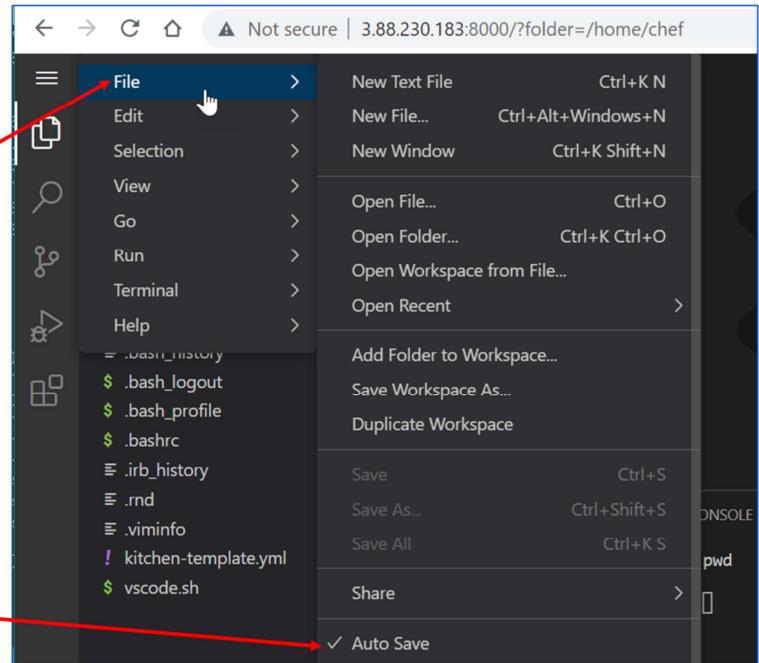
GL: Using VS Code

The VS Code app on your assigned workstation should have Auto Save enabled.

So you should not have to save changes manually using **Ctrl + S**

You can confirm the Auto Save setting as shown in this image.

Or you can uncheck Auto Save if you prefer to save changes manually. (e.g., using **Ctrl + S** to save.)



GL: Using VS Code

You can now exit from the terminal in which you ssh'd into the instance.

Just type exit at the prompt and close the terminal.

```
Administrator: Chef Workstation (delfante)
[17:02:53] [76.135.231.71][7c57faa8][ExtensionHostConnection] N
[17:02:53] [76.135.231.71][b4c6da7c][ManagementConnection] New
[17:02:53] [76.135.231.71][7c57faa8][ExtensionHostConnection] <
[root@ip-172-31-78-199 home]#
[root@ip-172-31-78-199 home]# exit
Logout
[chef@ip-172-31-78-199 home]$ exit
Logout
Connection to 3.221.150.29 closed.
PS C:\Users\delfante> -
```

From this point forward you will use the terminal you opened in VS Code.



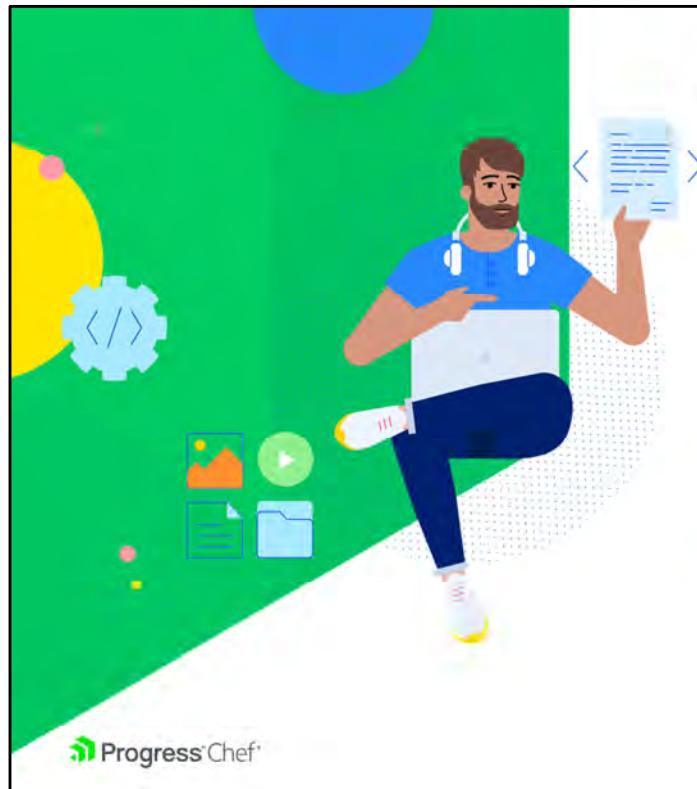


Chef Resources and Recipes

Chef's Fundamental Building Blocks



2- 1



Objectives

After completing this module, you should be able to:

- Define Chef Resources
- Create basic Chef recipe files
- Use the chef-client command

© 2020 Progress Software Corporation and/or its affiliates. All rights reserved.

2- 2

In this module you will learn how to use the 'chef-client' command, create a basic Chef recipe file and define Chef Resources.



Concept

Resources

A resource is a statement of configuration policy.

<https://docs.chef.io/resource/>

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

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 is an important reference page.

Resource Type Examples:

- file
- template
- remote_file
- package
- service

<https://docs.chef.io/resource/>

Concept



Resource Definition

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

The **TYPE** named **NAME** should be **ACTIONed** with **PROPERTIES**

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.



Concept

Resource Definition

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

The **TYPE** named **NAME** should be **ACTIONed** with **PROPERTIES**

The first element of the resource definition is the resource type. In this instance the type is 'file'.



Concept

Resource Definition

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

The **TYPE** named **NAME** should be **ACTIONed** with **PROPERTIES**

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 file path to the file we want created. We specified a fully-qualified file path to ensure the file was written to this exact location and not dependent on our current working directory, but we can use a relative path here also.

Concept



Resource Definition

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

The **TYPE** named **NAME** should be **ACTIONed** with **PROPERTIES**

The `do` and `end` keywords here define the beginning and end of a Ruby block. The Ruby block and all the contents within it are the properties to our resource.

The contents of this block contain properties (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 properties followed by a space and then the value for the attribute.



Concept

Resource Definition

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



The **TYPE** named **NAME** should be **ACTIONed** with **PROPERTIES**

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.

When an action is not specified a default action is chosen. The default action, often times, will not surprise you and in most cases perform an action that is creative or additive to the system. In this instance the default action for the file resource is to create the file if it does not exist.

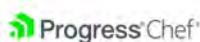
It's a best practice to always specify the action because you may not remember what every default action is.

Example: package

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

The package named 'httpd' is installed.

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



© 2019 Progress Software Corporation. All rights reserved.

2- 9

In this example, the package named 'httpd' is installed.

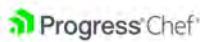
<https://docs.chef.io/resources/package/>

Example: service

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

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

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



© 2019 Progress Software Corporation. All rights reserved. All trademarks are property of their respective owners.

2- 10

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

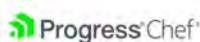
<https://docs.chef.io/resources/service/>

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/resources/file/>



In this example, the file named '/etc/motd' is created.

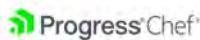
<https://docs.chef.io/resources/file/>

Example: file

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

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

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



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

Instructor Note: A resource's default action is based on the principle of least surprise. So they are often creative actions towards the system. This is why the file resource specified here has the action specified. It is not the default action.

Exercise



Group Lab: Hello, World?

- Create a recipe file writes out 'Hello, world!' to a text file
- Apply the recipe to the workstation

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

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!'.

Run Commands in VS Code in this course

The screenshot shows a Windows 10 desktop environment with the Visual Studio Code application open. In the top-left corner, there's a small icon of a laptop with a yellow arrow pointing from it towards the terminal pane. The terminal pane displays two commands: `> cd /home/chef` and `> pwd`. A yellow box with the text "Important: In this course when you see commands like this..." is overlaid on the terminal area, with a yellow arrow pointing from the text box to the first command. Below the terminal is the file explorer pane, showing files like test, .gitignore, CHANGELOG.md, .chefignore, kitchen.yml, LICENSE, metadata.rb, and .Policyfile.lock.json. The bottom-right corner of the screen shows the VS Code interface with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, and a status bar indicating the terminal tab is active. The terminal output shows the results of the commands entered: [chef@ip-172-31-73-167 home]\$ cd /home/chef, [chef@ip-172-31-73-167 ~]\$ pwd, /home/chef, [chef@ip-172-31-73-167 ~]\$.

Run Commands in VS Code in this course. The only exception is in the "Testing Cookbooks" module.

Execute on: **workstation**

GL: Move to your home directory and type pwd

```
> cd /home/chef
> pwd
```

test
.gitignore
CHANGELOG.md
chefignore
kitchen.yml
LICENSE
metadata.rb
Policyfile.lock.json

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[chef@ip-172-31-73-167 home]$ cd /home/chef
[chef@ip-172-31-73-167 ~]$ pwd
/home/chef
[chef@ip-172-31-73-167 ~]$
```

Progress Chef

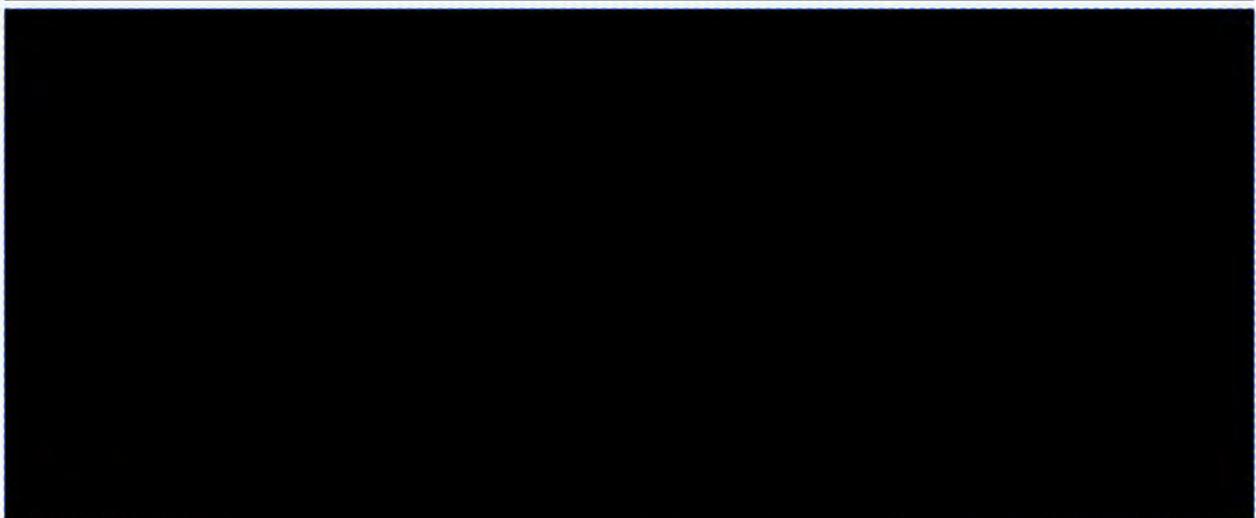
This is our working directory for today.

Execute on: **workstation**

GL: List the contents of your home directory



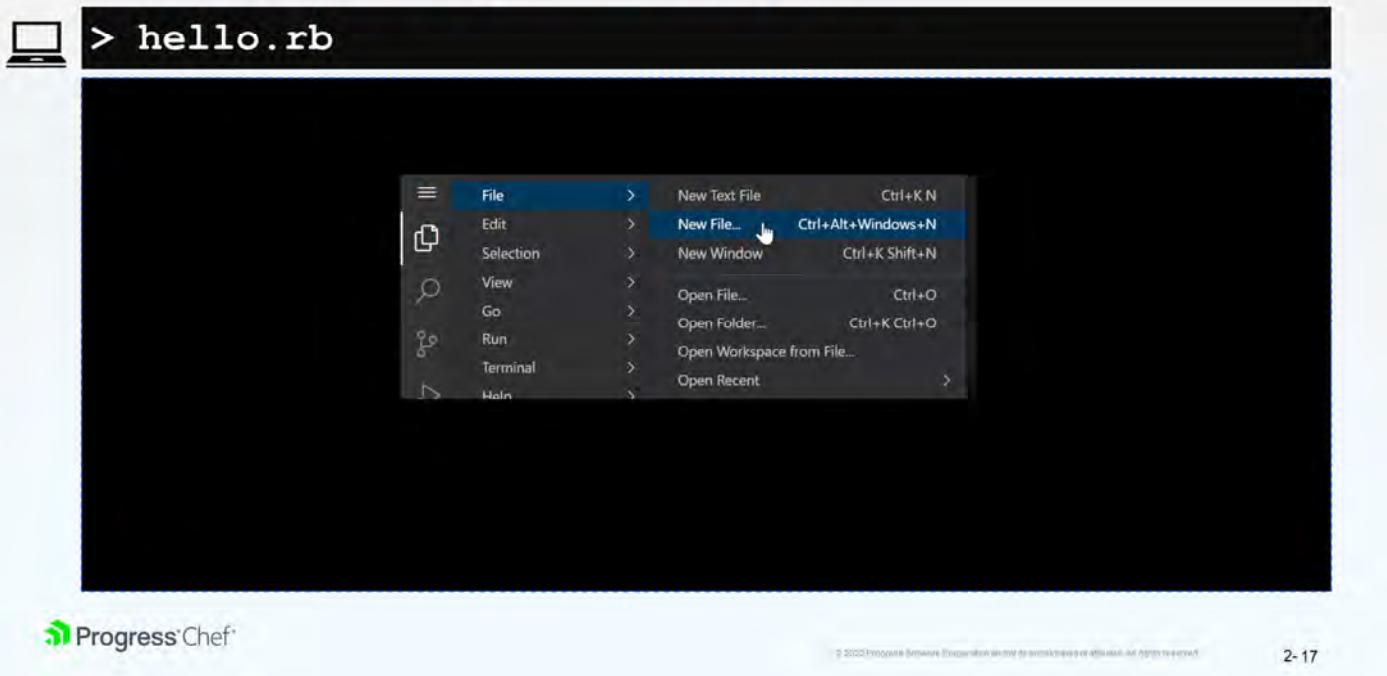
> ls



Execute on: **workstation**

Execute from: **/home/chef**

GL: Create and open a recipe file



Now that we have seen a few examples of resources let's get to work creating a text file. 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.

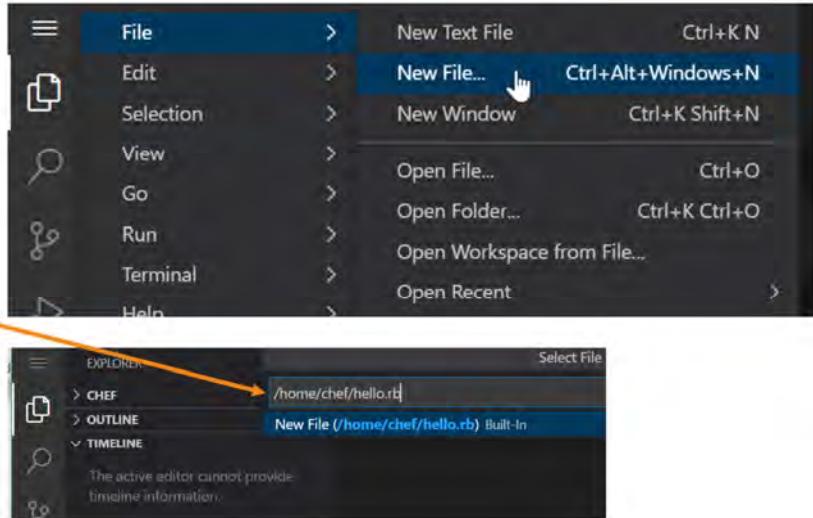
Execute on: **workstation**

Execute from: **/home/chef**

GL: Create a recipe file named hello.rb

Click **File > New File.**

Type **/home/chef/hello.rb**
and then Enter.



GL: Create a recipe file named hello.rb

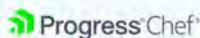
/home/chef/hello.rb

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

Type these contents into your hello.rb file.

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

<https://docs.chef.io/recipes/>



Add the resource definition displayed above. We are defining a resource with the type called 'file' and named '/home/chef/hello.txt'. We also are stating that the contents of that file should contain 'Hello, world!'.

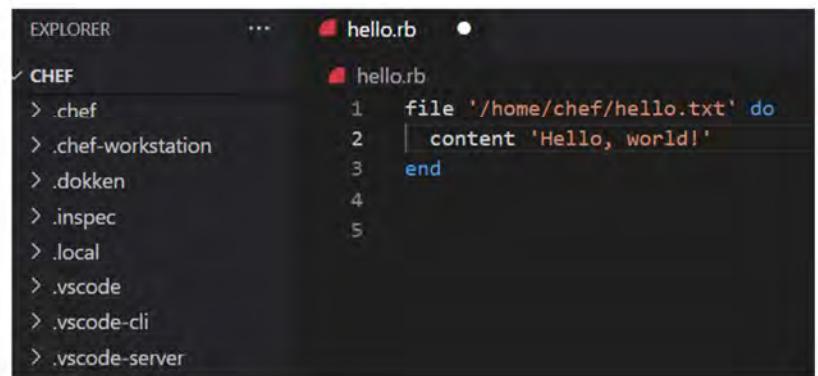
Save the file and return to the command prompt.

The default action is to create the file.

<https://docs.chef.io/recipes/>

GL: Create a recipe file named hello.rb

Your new file should look like this in VS Code.



A screenshot of the Visual Studio Code interface. On the left is the Explorer sidebar with a 'CHEF' folder expanded, containing subfolders like '.chef', '.chef-workstation', '.dokken', '.inspec', '.local', '.vscode', '.vscode-cli', and '.vscode-server'. On the right is the main code editor window titled 'hello.rb'. The code inside is:

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

Exercise



Group Lab: Hello, World?

- ✓ Create a recipe file writes out 'Hello, world!' to a text file
- Apply the recipe to the workstation

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

Now the file is created with the resource that will create the file with the content we want to see. It is time to apply that recipe to the system.



Concept

chef-client

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

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

https://docs.chef.io/ctl_chef_client/

In the Chef Workstation, we have packaged 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.

https://docs.chef.io/ctl_chef_client/

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 local mode.

'chef-client' has the 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.

https://docs.chef.io/ctl_chef_client/#run-in-local-mode

GL: Apply a recipe file



```
> sudo chef-client --chef-license accept --local-mode hello.rb
```

```
Compiling cookbooks...
Loading Chef InSpec profile files:
Loading Chef InSpec input files:
Loading Chef InSpec waiver files:
[2022-09-13T16:24:03+00:00] WARN: Node ip-172-31-73-167.ec2.internal has an empty run list.
Converging 1 resources
Recipe: @recipe_files:::/home/chef/hello.rb
* file[/home/chef/hello.txt] action create
  - create new file /home/chef/hello.txt
  - update content in file /home/chef/hello.txt from none to 315f5b
  --- /home/chef/hello.txt 2022-09-13 16:24:03.523305198 +0000
  +++ /home/chef/.chef-hello20220913-10624-cw4fut.txt 2022-09-13 16:24:03.523305198 +0000
  @@ -1 +1,2 @@
  +Hello, world!

Running handlers:
Running handlers complete
Infra Phase complete, 1/1 resources updated in 02 seconds
```

In addition to applying our recipe file, we are accepting the Chef licenses. This is because this is the first time you have run a Chef command on the workstation you're using. https://docs.chef.io/chef_license_accept.html#workstation-products

This license acceptance should persist as long as your workstation is running.

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>).

Execute on: **workstation**

Execute from: **/home/chef**

GL: What does hello.txt say?



```
> cat hello.txt
```

```
Hello, world!
```

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

Execute on: **workstation**

Execute from: **/home/chef**

Discussion

Discussion

What would happen if you ran the command again?

What would happen if the file were removed?

2-26

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?

And, of course, what would happen if the file was removed and the 'chef-client' command were to be run again?



Concept

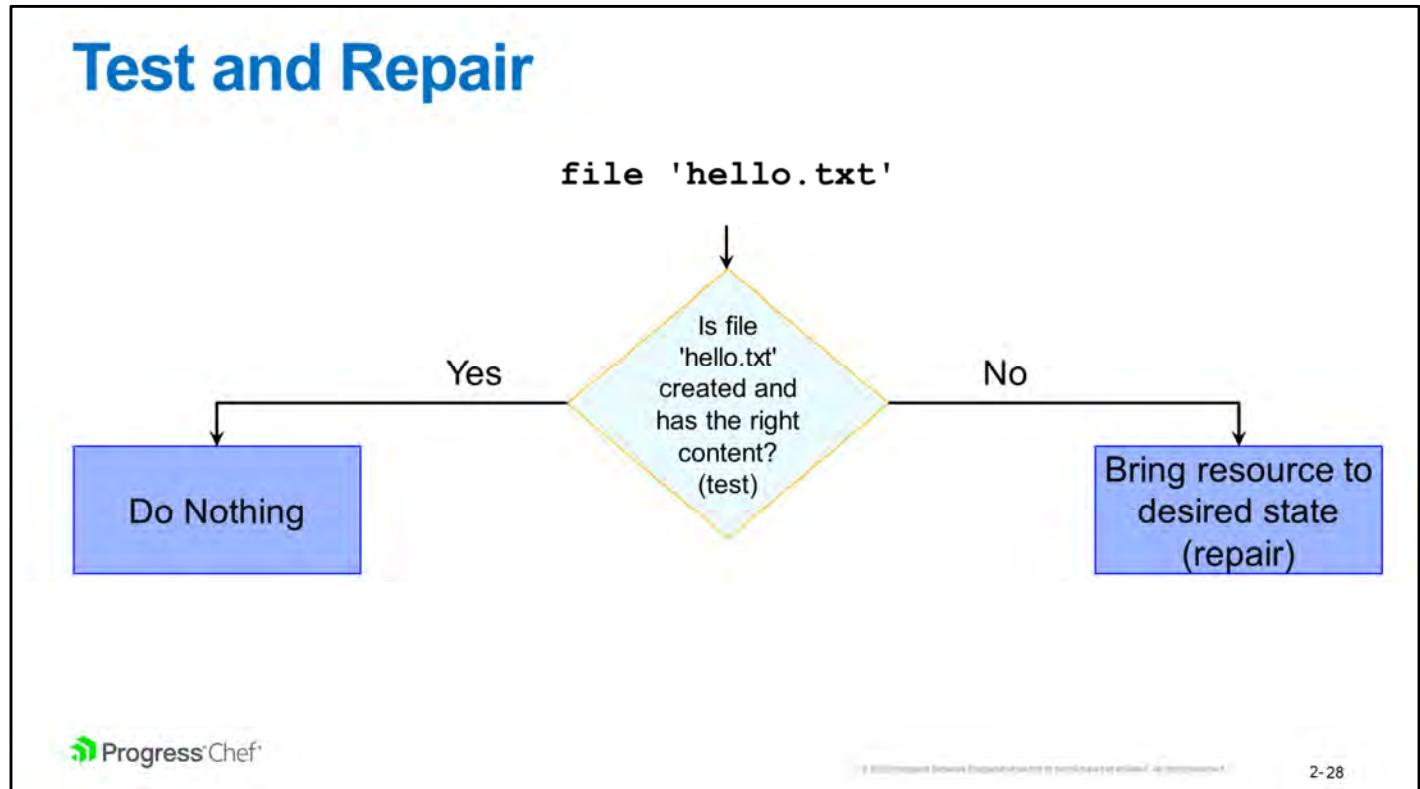
Test and Repair

`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.

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 is first tested on the system before it takes action.



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.

Exercise



Group Lab: Goodbye Recipe

Objective:

- Create a recipe named 'goodbye.rb' that removes 'hello.txt'
- Apply the recipe to the workstation

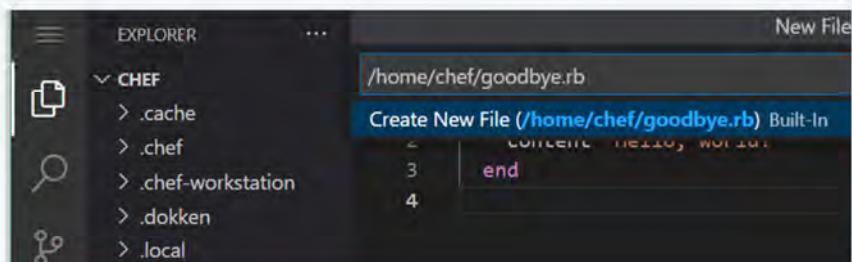
We know how to create a file with Chef, what about removing it?

We wrote and applied a recipe that creates a file on the workstation. Now, let's create a recipe that removes that same file. We will define a new recipe and then apply it to the workstation.

GL: Create a recipe named goodbye.rb

```
/home/chef/goodbye.rb
```

```
file '/home/chef/hello.txt' do
  action :delete
end
```

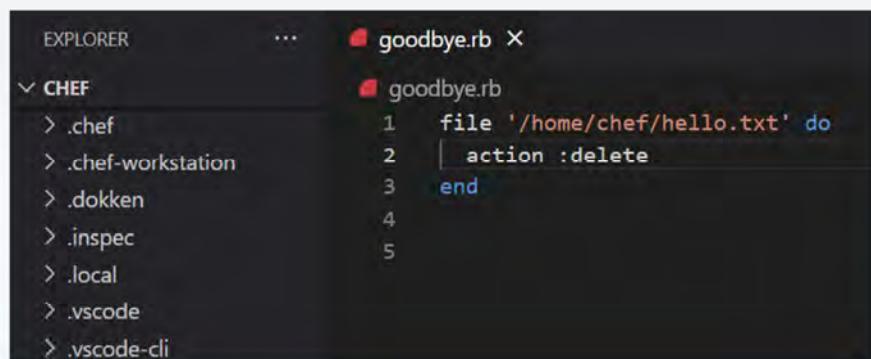


The file resource's default action is to create the file. So if we want to remove a file we need to explicitly define the action.

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

GL: Results in VS Code

□ /home/chef/goodbye.rb



The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a tree view with a expanded node labeled **CHEF**, which contains the following items:
 - .chef
 - .chef-workstation
 - .dokken
 - .inspec
 - .local
 - .vscode
 - .vscode-cli
- goodbye.rb**: The active file in the editor, showing the following code:

```
1 file '/home/chef/hello.txt' do
2   action :delete
3 end
4
5
```

Exercise



Group Lab: Goodbye Recipe

Objective:

- ✓ *Create a recipe that removes 'hello.txt'*
- *Apply the recipe to the workstation*

We know how to create a file with Chef, what about removing it?

The recipe has been defined and now it is time to apply it to the workstation.

GL: Verify that hello.txt still exists



> ls

goodbye.rb hello.rb hello.txt kitchen-template.yml nodes

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

Execute on: **workstation**

Execute from: **/home/chef**

GL: Apply a recipe file



```
> chef-client --local-mode goodbye.rb
```

```
Resolving cookbooks for run list: []
Synchronizing cookbooks:
Installing cookbook gem dependencies:
Compiling cookbooks...
Loading Chef InSpec profile files:
Loading Chef InSpec input files:
Loading Chef InSpec waiver files:
[2022-09-13T16:29:29+00:00] WARN: Node ip-172-31-73-167.ec2.internal has an
empty run list.
Converging 1 resources
Recipe: @recipe_files:::/home/chef/goodbye.rb
* file[/home/chef/hello.txt] action delete
  - delete file /home/chef/hello.txt
```

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

Execute on: **workstation**

Execute from: **/home/chef**

GL: Test that the hello.txt was deleted



> ls

```
goodbye.rb  hello.rb  kitchen-template.yml  nodes
```

Exercise



GL: Workstation Setup

Objective:

- *Create a recipe file named "setup.rb" that defines the policy:*
 - 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"*

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.

GL: Workstation Setup Recipe File

home/chef/setup.rb

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

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

The package named 'tree' is installed.

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



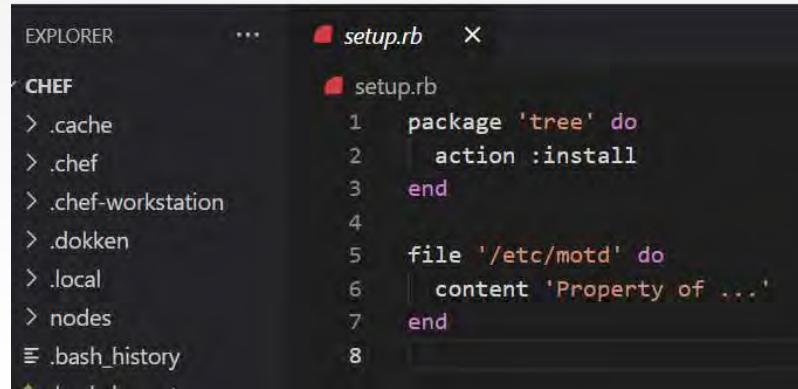
© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

2-37

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

GL: Results in VS Code

home/chef/setup.rb



The screenshot shows a dark-themed VS Code interface. In the top left, there's an 'EXPLORER' icon and three dots. To its right, a red circular icon with a white 'F' and the text 'setup.rb' next to it. On the far right is a close button ('X'). Below this, the word 'CHEF' is followed by a list of files: '.cache', '.chef', '.chef-workstation', '.dokken', '.local', 'nodes', and '.bash_history'. The main code editor area contains the following Ruby code:

```
1 package 'tree' do
2   action :install
3 end
4
5 file '/etc/motd' do
6   content 'Property of ...'
7 end
8
```

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

GL: Apply the Recipe File



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

```
Converging 2 resources
Recipe: @recipe_files:::/home/chef/setup.rb
  * yum_package[tree] action install (up to date)
    * file[/etc/motd] action create[2022-09-14T17:35:44+00:00] WARN: File /etc/motd managed by
      file[/etc/motd] is really a symlink (to /var/lib/update-motd/motd). Managing the source file
      instead.
[2022-09-14T17:35:44+00:00] WARN: Disable this warning by setting `manage_symlink_source true` on
the resource
[2022-09-14T17:35:44+00:00] WARN: In a future release, `manage_symlink_source` will not be enabled
by default

  - update content in file /etc/motd from a85ff3 to d100eb
  --- /etc/motd      2022-09-14 14:14:10.040291217 +0000
```

```
Running handlers:
```

```
Running handlers complete
```

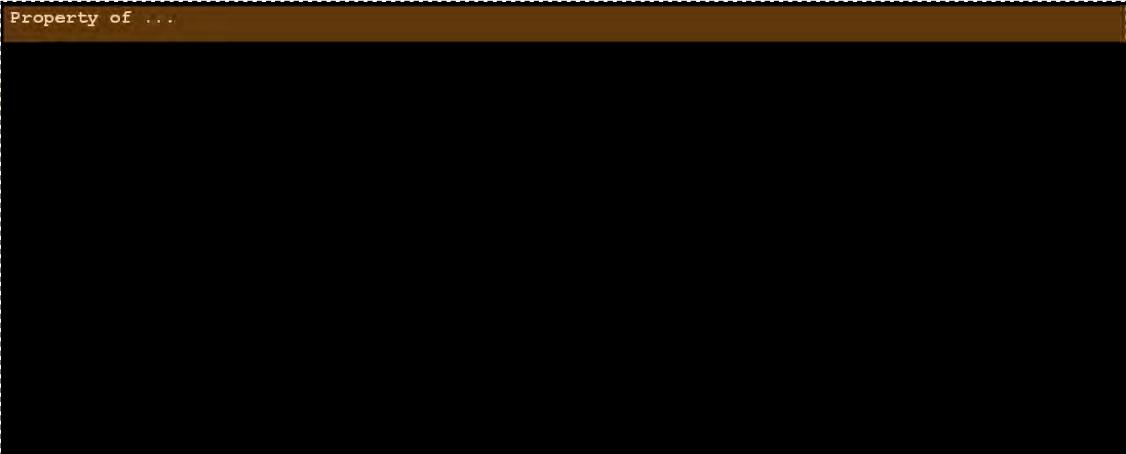
Applying it is the same as we did before with chef-client.

GL: Test the motd



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

Property of ...



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

Exercise



GL: Workstation Setup

Objective:

- ✓ *Create a recipe file named "setup.rb" that defines the policy:*
 - 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"*

Wonderful. The setup recipe now installs something fun, useful, and configures something important on our system. 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.

Review



Review Questions

What is a resource?

What is a Chef recipe?

What is chef-client and what does it do?

Answer these questions:

Q: What is a resource?

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.

Q: What is a Chef recipe?

A: A collection of resources that defines a desired state.

Q: What is chef-client and what does it do?

A: 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.



Review Questions

Q&A

What questions can we answer for you?

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

What questions can we answer for you?

About anything or specifically about: chef-client; resources; a resources default action and default properties; and Test and Repair



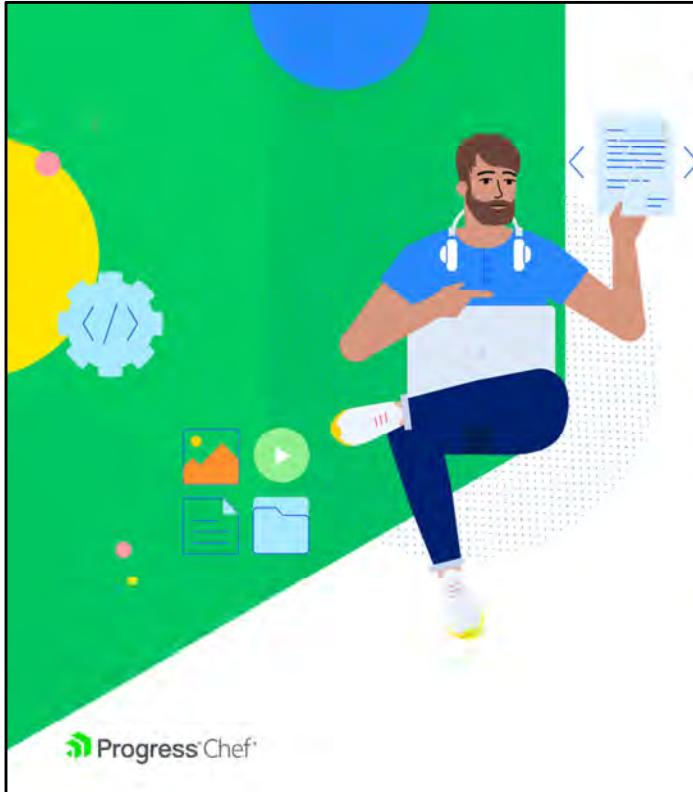


Cookbooks

Organizing Recipes



3- 1



Objectives

After completing this module you should be able to:

- Generate a Chef cookbook
- Use version control
- Implement the include_recipe method
- Apply a run-list of recipes to a system
- Define a Chef cookbook that sets up a web server.
- Use `cookstyle` for linting cookbooks

© 2020 Progress Software Corporation and/or its subsidiaries. All rights reserved.

3- 2

In this module you will learn how to generate a cookbook with the Chef command-line application and apply multiple recipes to a system through a run-list.

Collaboration and Version Control

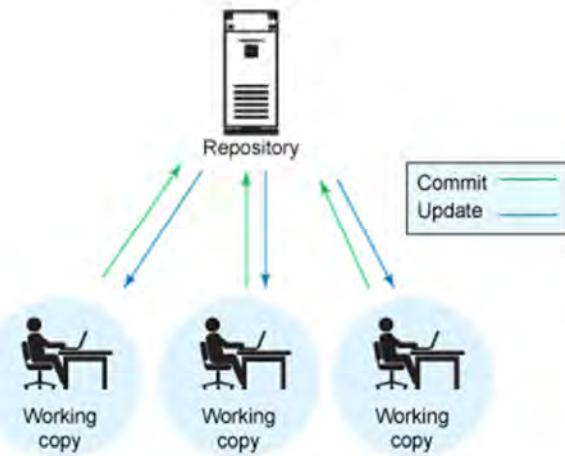
A versioning system should include:

A Central Repository into which all the developers publish their work.

Each revision should be stored as a new version.

For each change, a commit message should be added so that everyone knows what has or has not been changed.

Centralized version control system



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 commit message should be added so that everyone knows what has or has not been changed.

Versioning pros and cons

```
> cp hello.rb hello.rb.bak  
or  
> cp hello.rb hello-20170401.rb  
or  
> cp setup.rb setup-20170401-username.rb
```

Saving a copy of the original file as another filename.

Let's 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 let's talk about the pros and cons of using this strategy.

So 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 let's 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?

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 this module so you can get an overview of how **git** works.



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.

Instructor Note: It is not important that the learners understand and learn all of git during this course. It is more important that the learners understand when and where to use version control to save their work. This is about training them on making changes, testing, and then committing their work. Version control is an instrumental piece of the workflow when you adopt Infrastructure as code. There are some benefits of learning and using git because Chef uses git and GitHub to do almost all development of Chef. The majority of the Chef community uses git and GitHub.

Git version control

Learning everything about **git** is beyond the scope of this course. So you should try to learn more git or a version control system of your organization's choice.

There are many free resources that you can use to learn more about **git**. For example:

<https://git-scm.com/doc>



<https://git-scm.com/doc>

GL: Setup your global git configuration



```
> git config --global user.email "you@example.com"
```

First, let's set up your email address and name in the global git configuration.

Set up the email you want to associate with your git commits.

Execute on: **workstation**

Execute from: **/home/chef**

GL: Setup your global git configuration



```
> git config --global user.name "Your Name"
```

Set-up the user name you want to associate with your git commits.

We'll come back to git later in the module.

Execute on: **workstation**

Execute from **/home/chef**

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: <https://docs.chef.io/cookbooks/>



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.

<https://docs.chef.io/cookbooks/>

Concept



Cookbook

A cookbook usually maps 1:1 to an application or to a scenario. When we define a cookbook we usually have a goal in mind that this cookbook will accomplish.

In our case we are interested in a cookbook that configures a workstation. So within that cookbook we will define all the recipes to accomplish this goal.

Exercise



GL: Setting up a Workstation

- *Create a chef-repo*
- *Create a cookbook*
- *Use git*
- *Copy the setup recipe within the cookbook*
- *Use the include_recipe method to insert setup recipe*
- *Apply the default recipe to the workstation*

The setup.rb recipe is one of many recipes that we could define to setup our workstations. But before we put 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'.

Instructor Note:

There are GitHub repositories available for all of the cookbooks created in this course. If a student is having difficulties following along or needs to catch up at any point, have them visit the url associated with that cookbook and download the repo to their workstation. Be aware that there might be multiple versions of a cookbook and they are all included in the repo. Make sure the student is using the correct version number of the cookbook at that point in the class.

The GitHub repo for the 'workstation' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-workstation.git>

Concept



chef-repo

chef-repo is a directory on your workstation that stores everything you need to define your infrastructure with Chef Infra:

- Cookbooks (including recipes, attributes, custom resources, libraries, and templates)
- Policyfiles

The chef-repo directory should be synchronized with a version control system, such as git. All of the data in the chef-repo should be treated like source code.

https://docs.chef.io/chef_repo/

https://docs.chef.io/chef_repo/



Concept

chef generate repo

In a moment we will use the 'chef generate' command to create a chef-repo.

You can generate a repo manually but we'll use chef to do this.

We'll discuss other 'chef generate' commands later in this module.

GL: Ensure you are in the home directory



```
> cd ~  
> pwd
```

```
/home/chef
```

We will want to have a location for our cookbooks to reside within the '/home/chef' directory of our lab system.

Execute on: **workstation**

GL: Generate a repo for your cookbooks



```
> chef generate repo chef-repo
```

```
Generating Chef Infra repo chef-repo
```

```
- Ensuring correct Chef Infra repo file content
```

```
Your new Chef Infra repo is ready! Type `cd chef-repo` to  
enter it.
```

Execute on: **workstation**

Execute from: **/home/chef**

GL: Move into chef-repo and list contents



```
> cd chef-repo  
> ls -l
```

```
total 12  
-rw-rw-r-- 1 chef chef 1252 Sep 14 18:50 cheffignore  
drwxrwxr-x 3 chef chef 38 Sep 14 18:50 cookbooks  
drwxrwxr-x 3 chef chef 38 Sep 14 18:50 data_bags  
-rw-rw-r-- 1 chef chef 70 Sep 14 18:50 LICENSE  
drwxrwxr-x 2 chef chef 23 Sep 14 18:50 policyfiles  
-rw-rw-r-- 1 chef chef 1349 Sep 14 18:50 README.md
```

In a moment you will generate a cookbook
within the new cookbooks directory.

Execute on: **workstation**

Execute from: **/home/chef**

Concept



What is 'chef'?

An executable program that allows you to generate cookbooks and cookbook components.

Cookbooks are nothing more than directories with specific files. We could create a cookbook by hand by finding an example and copying that pattern or we could use a tool to help us generate a cookbook.

The Chef Workstation comes with a tool named 'chef'. This command-line tool has a number of features.

GL: 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 Chef Workstation 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.

Execute on: **workstation**

Execute from: **Anywhere**

GL: What can 'chef generate' do?



```
> chef generate --help
```

```
Usage: chef generate GENERATOR [options]
```

```
Available generators:
```

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
helpers	Generate a cookbook helper file in libraries
resource	Generate a custom resource
repo	Generate a Chef policy repository
policyfile	Generate a Policyfile for use with the install/push commands

Let's 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'.

Execute on: **workstation**

Execute from: **Anywhere**

GL: What can 'chef generate cookbook' do?



```
> chef generate cookbook --help
```

Usage: chef generate cookbook NAME [options]

-C, --copyright COPYRIGHT	Name of the copyright holder - default...
-m, --email EMAIL	Email address of the author - defaults...
-a, --generator-arg KEY=VALUE	Use to set arbitrary attribute KEY to ...
-l, --license LICENSE	all_rights, httpd, mit, gplv2, gplv3 -
-g GENERATOR_COOKBOOK_PATH,	Use GENERATOR_COOKBOOK_PATH for the
--generator-cookbook	

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

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

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

Execute on: **workstation**

Execute from: **Anywhere**

GL: Let's create a cookbook



```
> chef generate cookbook cookbooks/workstation
```

```
Generating cookbook cookbooksworkstation
- Ensuring correct cookbook content
```

```
Your cookbook is ready. Type `cd cookbooksworkstation` to enter it.
```

```
There are several commands you can run to get started locally developing and testing
your cookbook.
```

```
Why not start by writing an InSpec test? Tests for the default recipe are stored at:
```

```
test/integration/default/default_test.rb
```

```
If you'd prefer to dive right in, the default recipe can be found at:
```

```
recipes/default.rb
```

```
- Ensuring correct cookbook content
```

Be sure to accept the license by
typing **yes** if prompted.

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.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

GL: The cookbook has a README



```
> ls -al cookbooks/workstation
```

```
drwxrwxr-x 6 chef chef 161 Sep 14 19:18 .
drwxr-xr-x 9 chef chef 292 Sep 14 18:50 ..
-rw-rw-r-- 1 chef chef 1252 Sep 14 18:50 chefignore
-rw-rw-r-- 1 chef chef 275 Sep 14 18:50 .chef-repo.txt
drwxrwxr-x 4 chef chef 57 Sep 14 19:14 cookbooks
drwxrwxr-x 3 chef chef 38 Sep 14 18:50 data_bags
drwxrwxr-x 7 chef chef 119 Sep 14 18:50 .git
-rw-rw-r-- 1 chef chef 2057 Sep 14 18:50 .gitignore
-rw-rw-r-- 1 chef chef 70 Sep 14 18:50 LICENSE
drwxrwxr-x 2 chef chef 23 Sep 14 18:50 policyfiles
-rw-rw-r-- 1 chef chef 1349 Sep 14 18:50 README.md
```

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.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

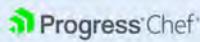
Concept



README.md

The description of the cookbook's features written in Markdown.

<http://daringfireball.net/projects/markdown/syntax>



© 2020 Progress Software Corporation and/or its affiliated entities. All rights reserved.

3-23

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 rendered as HTML or other formats by computers.

GL: Let's take a look at the README



```
> cat cookbooks/workstation/README.md
```

```
# workstation
```

```
TODO: Enter the cookbook description here.
```

The cookbook's metadata is where you can define the name of the cookbook's maintainer, a way to reach them, how the cookbook is licensed, descriptions, and the cookbook's version number.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

GL: The cookbook has some metadata



```
> ls cookbooks/workstation
```

```
CHANGELOG.md  compliance  LICENSE      Policyfile.lock.json  README.md  test  
chefignore    kitchen.yml  metadata.rb  Policyfile.rb        recipes
```

The cookbook also has a metadata file.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**



Concept

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.

https://docs.chef.io/config_rb_metadata/

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

https://docs.chef.io/config_rb_metadata/

GL: Let's take a look at the metadata



```
> cat cookbooks/workstation/metadata.rb
```

```
name 'workstation'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures workstation'
version '0.1.0'
chef_version '>= 16.0'
```

```
# The `issues_url` points to the location where issues are tracked. A `View Issues` link will be displayed when uploaded to a Supermarket.
#
# issues_url 'https://github.com/<insert_org_here>/workstation/issues'
```

```
...
```

Whenever you make a change to a cookbook, you should bump its version in that cookbook's metadata.rb file.

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.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

GL: The cookbook has a folder for recipes



```
> ls -al cookbooks/workstation
```

```
drwxrwxr-x 5 chef chef 198 Sep 14 19:14 .
drwxrwxr-x 4 chef chef 57 Sep 14 19:14 ..
-rw-rw-r-- 1 chef chef 160 Sep 14 19:14 CHANGELOG.md
-rw-rw-r-- 1 chef chef 1252 Sep 14 19:14 cheffignore
drwxrwxr-x 5 chef chef 68 Sep 14 19:14 compliance
-rw-rw-r-- 1 chef chef 211 Sep 14 19:14 .gitignore
-rw-rw-r-- 1 chef chef 728 Sep 14 19:14 kitchen.yml
-rw-rw-r-- 1 chef chef 70 Sep 14 19:14 LICENSE
-rw-rw-r-- 1 chef chef 696 Sep 14 19:14 metadata.rb
-rw-rw-r-- 1 chef chef 518 Sep 14 19:14 Policyfile.rb
-rw-rw-r-- 1 chef chef 59 Sep 14 19:14 README.md
drwxrwxr-x 2 chef chef 24 Sep 14 19:14 recipes
drwxrwxr-x 3 chef chef 25 Sep 14 19:14 test
```

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?

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

GL: The cookbook has a 'default' recipe



```
> cat cookbooks/workstation/recipes/default.rb
```

```
#  
# Cookbook:: workstation  
# Recipe:: default  
#  
# Copyright:: 2022, 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 need 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.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

Exercise



GL: Setting up a Workstation

- ✓ *Create a chef-repo*
- ✓ *Create a cookbook*
- *Use git*
- *Copy the setup recipe within the cookbook*
- *Use the include_recipe method to insert setup recipe*
- *Apply the default recipe to the workstation*

GL: Move your setup.rb recipe into the cookbook



```
> mv /home/chef/setup.rb cookbooks/workstation/recipes/setup.rb
```

```
> ls cookbooks/workstation/recipes/
```

```
total 8
drwxrwxr-x 2 chef chef 40 Sep 14 19:45 .
drwxrwxr-x 5 chef chef 198 Sep 14 19:14 ..
-rw-rw-r-- 1 chef chef 102 Sep 14 19:14 default.rb
-rw-rw-r-- 1 chef chef 98 Sep 14 17:31 setup.rb
```

Execute on: **workstation**

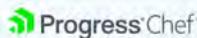
Execute from: **/home/chef/chef-repo**

GL: Initialize cookbook directory as a git repo



```
> cd cookbooks/workstation  
> git init  
  
...  
Initialized empty Git repository in /home/chef/chef-  
repo/cookbooks/workstation/.git/
```

REMOTE

© 2020 Progress Software Corporation. All rights reserved.3- 32

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.

You will notice that git will say that the repository has been 'Reinitialized'. This is because the chef cookbook generator detected that we have git installed and automatically initialized the cookbook as a git repository and added the new files to the local git repo.

'cd' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

'git' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation**

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>

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, 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.

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

GL: Use 'git add' to Stage Files to be Committed

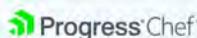


```
> git add .
```

```
warning: LF will be replaced by CRLF in .gitignore.
```

```
The file will have its original line endings in your working directory
```

REMOTE

© 2020 Progress Software Corporation. All rights reserved.3- 34

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 from the current directory or below into a staging area.

Note: You could also execute `git add FILENAME` if you want to be specific on what you are adding to git.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation**

GL: Use 'git status' to View the Staged Files



```
> git status
```

```
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:  .gitignore
    new file:  CHANGELOG.md
    new file:  LICENSE
    new file:  Policyfile.rb
    new file:  README.md
    new file:  cheffignore
    new file:  compliance/README.md
    new file:  kitchen.yml
    new file:  metadata.rb
    new file:  recipes/default.rb
    new file:  recipes/setup.rb
    new file:  test/integration/default/default_test.rb
```

Let's 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 accidentally threw in there.

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

Instructor Note: Git helpfully tries to show you the command you can use to remove an item from that box. This is useful if you want to include all items except for one or simply manage everything before you commit.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation**

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



```
> git commit -m "Adds two recipes and auto-generated content"
```

```
[chef@ip-172-31-73-167 workstation]$ git commit -m "Initial Commit"
[main (root-commit) 88a3113] Initial Commit
12 files changed, 277 insertions(+)
create mode 100644 .gitignore
create mode 100644 CHANGELOG.md
create mode 100644 LICENSE
create mode 100644 Policyfile.rb
create mode 100644 README.md
create mode 100644 chefignore
create mode 100644 compliance/README.md
create mode 100644 kitchen.yml
create mode 100644 metadata.rb
create mode 100644 recipes/default.rb
create mode 100644 recipes/setup.rb
create mode 100644 test/integration/default/default_test.rb
```

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, which is done with the **-m** flag and a string of text that describes that change.

The string of text we use here conforms to the accepted git style guide. In short, it means that we use a short message (less than 50 characters) that describes the change we are making in present imperative tense (change this vs changed that)

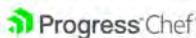
Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation**

Git Version Control

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

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

A screenshot of a GitHub repository page for "chef-training / chef-foundations-using-automate". The repository is private. The main branch is "main". There are 2 branches and 0 tags. The commit history shows four commits from "SteveDelFante" made 20 hours ago. Each commit is labeled "initial commit post formatting" and was made 6 months ago.

Commit	File	Message	Time Ago
1	01-Introduction.pptx	initial commit post formatting	6 months ago
2	02-Resources.pptx	initial commit post formatting	6 months ago
3	03-Cookbooks.pptx	initial commit post formatting	6 months ago
4	04-Ohai.pptx	initial commit post formatting	6 months ago

3-37

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. GitHub is an example of a central git repository.



Concept

chef-client

```
$ chef-client --local-mode RECIPE_FILE
```

How would we apply the workstation's setup recipe?

We have used 'chef-client' to apply recipes but we now face a new problem. How do we use this tool to apply multiple recipes to configure the state of our infrastructure? Combing the recipes seems like it goes against the concept that cookbooks map one-to-one to a piece of software. Running the command twice seems like it would make managing the system difficult to remember.

Concept



--runlist "recipe[COOKBOOK::RECIPE]"

In local mode, we need to provide a list of recipes to apply to the system. This is called a **run list**. A run list is an ordered collection of recipes to execute.

Each recipe in the run list must be addressed with the format **recipe[COOKBOOK::RECIPE]**.

Another flag we can use is '--runlist' or '-r' to specify a list of recipes we want to apply to the system. We call this list of recipes a run list.

This ordered list specifies the recipes in a different way. We are no longer interested in the filepath to the particular recipe file. We instead specify that we want a recipe and then within the square brackets we specify the name of the cookbook and then finally the name of the recipe.

"recipe[COOKBOOK::RECIPE]"

COOKBOOK means the name of the Cookbook.

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

GL: Applying the setup.rb recipe

```
> cd ..  
> sudo chef-client --local-mode --runlist "recipe[workstation::setup]"  
  
...  
Converging 2 resources  
Recipe: workstation::setup  
  * yum_package[tree] action install (up to date)  
  * file[/etc/motd] action create[2022-09-14T20:15:27+00:00] WARN: File  
/etc/motd managed by file[/etc/motd] is really a symlink (to /var/lib/update-  
motd/motd). Managing the source file instead.  
[2022-09-14T20:15:27+00:00] WARN: Disable this warning by setting  
'manage_symlink_source true' on the resource  
[2022-09-14T20:15:27+00:00] WARN: In a future release, 'manage_symlink_source'  
will not be enabled by default  
  (up to date)  
...  
Infra Phase complete, 0/2 resources updated in 03 seconds
```

Let's apply the workstation cookbook's recipe named 'setup'.

'cd' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation**

'chef-client' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks**

Exercise



GL: Setting up a Workstation

- ✓ Create a chef-repo
- ✓ Create a cookbook
- ✓ Use git
- ✓ Copy the setup recipe within the cookbook
 - Use the include_recipe method to insert setup recipe
 - Apply the default recipe to the workstation

Concept



include_recipe method

A recipe can include one (or more) recipes located in cookbooks by using the `include_recipe` method. When a recipe is included, the resources found in that recipe will be inserted (in the same exact order) at the point where the `include_recipe` keyword is located.

<https://docs.chef.io/recipes.html#include-recipes>

What if we want access to what's found in a recipe from another recipe? The `include_recipe` method will allow us to do that and essentially copy and paste everything found in one recipe into another.

<https://docs.chef.io/recipes.html#include-recipes>

GL: Include setup recipe in the default recipe

```
~/chef-repo/cookbooks/workstation/recipes/default.rb

#
# Cookbook:: workstation
# Recipe:: default
#
# Copyright:: 2022, 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`.

Concept



-r "recipe[COOKBOOK(:default)]"

When you are referencing the default recipe within a cookbook you may optionally specify only the name of the cookbook.

chef-client understands that you mean to apply the default recipe from within that cookbook.

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 properties, Chef uses the concept of providing sane defaults. 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.

GL: Apply the cookbook's default recipe



```
> sudo chef-client --local-mode -r "recipe[workstation]"  
  
Resolving cookbooks for run list: ["workstation"]  
Synchronizing cookbooks:  
  - workstation (0.1.0)  
Installing cookbook gem dependencies:  
...  
Converging 2 resources  
Recipe: workstation::setup  
  * yum_package[tree] action install (up to date)  
  * file[/etc/motd] action create[2022-09-14T20:24:22+00:00] WARN: File /etc/motd  
managed by file[/etc/motd] is really a symlink (to /var/lib/update-motd/motd).  
...  
[2022-09-14T20:24:22+00:00] WARN: In a future release, 'manage_symlink_source'  
will not be enabled by default  
(up to date)  
Running handlers:  
Running handlers complete  
Infra Phase complete, 0/2 resources updated in 03 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.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks**

Exercise



GL: Setting up a Workstation

- ✓ Create a chef-repo
- ✓ Create a cookbook
- ✓ Use git
- ✓ Copy the setup recipe within the cookbook
- ✓ Use the include_recipe method to insert setup recipe
- ✓ Apply the default recipe to the workstation

Instructor Note:

There are GitHub repositories available for all of the cookbooks created in this course. If a student is having difficulties following along or needs to catch up at any point, have them visit the url associated with that cookbook and download the repo to their workstation. Be aware that there might be multiple versions of a cookbook and they are all included in the repo. Make sure the student is using the correct version number of the cookbook at that point in the class.

Exercise



Lab: Setting up a Web Server

- Use `chef generate` to create a cookbook named "apache".
- Write and apply a recipe named "server.rb" with the policy:
 - The package named 'httpd' is installed.
 - The file named '/var/www/html/index.html' is created with the content '`<h1>Hello, world!</h1>`'
 - The service named 'httpd' is started and enabled.
- Apply the recipe with `chef-client`
- Verify the site is available by running `curl localhost`

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!

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Return to the chef-repo directory



```
> cd /home/chef/chef-repo
```

```
/home/chef/chef-repo
```

Lab: Create a cookbook



```
> chef generate cookbook cookbooks/apache
```

```
Generating cookbook apache
```

```
- Ensuring correct cookbook content
```

```
Your cookbook is ready. Type `cd cookbooks/apache` to enter it.
```

```
There are several commands you can run to get started locally developing and testing your cookbook.
```

```
Why not start by writing an InSpec test? Tests for the default recipe are stored at:
```

```
test/integration/default/default_test.rb
```

```
If you'd prefer to dive right in, the default recipe can be found at:
```

```
recipes/default.rb
```

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

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

Exercise



Lab: Setting up a Web Server

- ✓ Use `chef generate` to create a cookbook named "apache".
- Create a recipe named "server.rb" with the following policy:
 - The package named 'httpd' is installed.
 - The file named '/var/www/html/index.html' is created with the content '`<h1>Hello, world!</h1>`'
 - The service named 'httpd' is started and enabled.
- Apply the recipe with `chef-client`
- Verify the site is available by running `curl localhost`

Instructor Note:

The GitHub repo for the `apache` cookbook can be found at: <https://github.com/chef-training/devops-cookbooks-apache.git>

Lab: Create a recipe



```
> chef generate recipe cookbooks/apache server
```

```
Recipe: code_generator::recipe
* directory[cookbooks/apache/spec/unit/recipes] action create
  - create new directory cookbooks/apache/spec/unit/recipes
* cookbook_file[cookbooks/apache/spec/spec_helper.rb] action create_if_missing
  - create new file cookbooks/apache/spec/spec_helper.rb
  - update content in file cookbooks/apache/spec/spec_helper.rb from none to 945e09
    (diff output suppressed by config)
* template[cookbooks/apache/spec/unit/recipes/server_spec.rb] action create_if_missing
  - create new file cookbooks/apache/spec/unit/recipes/server_spec.rb
  - update content in file cookbooks/apache/spec/unit/recipes/server_spec.rb from none to 71bc42
    (diff output suppressed by config)
* directory[cookbooks/apache/test/integration/default] action create (up to date)
* template[cookbooks/apache/test/integration/default/server_test.rb] action create_if_missing
  - create new file cookbooks/apache/test/integration/default/server_test.rb
  - update content in file cookbooks/apache/test/integration/default/server_test.rb from none to
768718
    (diff output suppressed by config)
* template[cookbooks/apache/recipes/server.rb] action create
  - create new file cookbooks/apache/recipes/server.rb
  - update content in file cookbooks/apache/recipes/server.rb from none to 817f67
    (diff output suppressed by config)
```

From the chef-repo directory, run the command 'chef generate recipe apache server'. This will create a recipe called server.rb in the apache cookbook.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

Lab: Add code to the server recipe

cookbooks/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.

Instructor Note: This is the first time we are using the package resource without the action specified. Remind the learners that they have been doing the same thing for the file resource. When you specify no actions or parameters within the block, the block is no longer necessary.

Instructor Note: The service action defines two actions within a Ruby array. Ruby arrays are ordered, integer-indexed collections of any object. Each element in an array is associated with and referred to by an index.

Execute from: /home/chef/chef-repo

Lab: Include server in the default recipe

cookbooks/apache/recipes/default.rb

```
#  
# Cookbook:: apache  
# Recipe:: default  
#  
# Copyright:: 2022, The Authors, All Rights  
Reserved.  
#  
include_recipe 'apache::server'
```

We are interested in having the default recipe for our workstation 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.

include_recipe 'apache'

Lab: Apply the Default Recipe



```
> sudo chef-client --local-mode --runlist "recipe[apache]"  
Synchronizing cookbooks:  
  - apache (0.1.0)  
Installing cookbook gem dependencies:  
Compiling cookbooks...  
...  
Converging 3 resources  
Recipe: apache::server  
* yum_package[httpd] action install  
  - install version 0:2.4.54-1.amzn2.x86_64 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 2022-09-14 20:53:18.512331478 +0000  
    +++ /var/www/html/.chef-index20220914-15764-1w2tdl.html 2022-09-14 20:53:18.512331478 +0000  
    @@ -1 +1,2 @@  
    +<h1>Hello, world!</h1>  
* service[httpd] action enable  
  - enable service service[httpd]  
* service[httpd] action start  
  - start service service[httpd]  
Infra Phase complete, 4/4 resources updated in 09 seconds
```

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

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

Lab: Verify that the Website is Available



```
> curl localhost
```

```
<h1>Hello, world!</h1>
```

Exercise

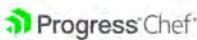


Lab: Setting up a Web Server

- ✓ Use `chef generate` to create a cookbook named "apache".
- ✓ Write and apply a recipe named "server.rb" with the policy:
 - The package named 'httpd' is installed.
 - The file named '/var/www/html/index.html' is created with the content '`<h1>Hello, world!</h1>`'
 - The service named 'httpd' is started and enabled.
- ✓ Apply the recipe with `chef-client`
- ✓ Verify the site is available by running `curl localhost`

Lab: Commit Your Work

```
> cd /home/chef/chef-repo/cookbooks/apache  
> git init  
> git add .  
> git status  
> git commit -m "Adds apache cookbook"
```

© 2018, Progress International Corporation. All rights reserved.3-57

Execute on: **workstation**

Execute from: **anywhere**

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

1. Move into the apache directory.
2. Initialize the cookbook as a git repository.
3. Add all the files within the cookbook.
4. Use `git status` to see what you're adding if you like.
5. Commit all the files in the staging area.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/**



Concept

Committing Your Work to a Version Control Repo

When you submit or "push" your changes to a central repo like github, you should always push your changes to a github **branch** other than the main branch.

In this way, your changes can later be merged into the main branch in a collaborative manner, allowing for reviews of your work before merging.

<https://learn.chef.io/modules/version-control#/>

<https://git-scm.com/doc>

Concept



Committing Your Work to Version Control

By now you should have an overview-level of understanding about version control and git.

In practice you should always commit your latest changes to version control, like git.

In the interest of time we won't do any more git tasks in this course. Version control links are below in your participant guide.

<https://learn.chef.io/modules/version-control#/>

<https://git-scm.com/doc>

Concept



Linting With cookstyle

'cookstyle' is an easy-to-use linting tool that is included with Chef Workstation.

Concept



Linting With cookstyle

You can simply run 'cookstyle' from the directory where your recipe resides and it will indicate any syntax offenses.

<https://docs.chef.io/workstation/cookstyle/>

<https://docs.chef.io/workstation/cookstyle/>

Example: Running cookstyle on recipes



```
$ cookstyle
```

```
Offenses:

default.rb:8:1: C: [Correctable] Layout/TrailingEmptyLines: 1 trailing blank lines detected.
(https://rubystyle.guide#newline-eof)
server.rb:10:1: C: [Correctable] Layout/IndentationWidth: Use 2 (not 4) spaces for indentation.
(https://rubystyle.guide#spaces-indentation)
  content '<h1>Hello, world!</h1>'
^^^^
server.rb:14:1: C: [Correctable] Layout/IndentationWidth: Use 2 (not 4) spaces for indentation.
(https://rubystyle.guide#spaces-indentation)
  action [:enable, :start]
^^^^
server.rb:16:1: C: [Correctable] Layout/TrailingEmptyLines: 1 trailing blank lines detected.
(https://rubystyle.guide#newline-eof)

2 files inspected, 4 offenses detected, 4 offenses auto-correctable
```

In this example, we ran `cookstyle` from within the recipes directory where we created the server.rb recipe. `cookstyle` has detected a trailing blank line, although it's not a critical error.

Notice that the line number where the offense exists is indicated. This command was run from : /home/chef/chef-repo/cookbooks/apache/recipes/apache/recipes

Example: Running cookstyle on recipes



```
$ cookstyle
```

```
Inspecting 6 files
...C...
Offenses:

recipes/default.rb:13:27: C: Trailing whitespace detected.
  action [:start, :enable]
                      ^
6 files inspected, 1 offense detected
```

In this example from a different recipe, 'cookstyle' even shows via the caret where the trailing whitespace exists.

REMOTE

Example: `cookstyle -a`



```
$ cookstyle -a
```

```
Inspecting 2 files
CC

Offenses:

default.rb:8:1: C: [Corrected] Layout/TrailingEmptyLines: 1 trailing blank lines
detected. (https://rubystyle.guide#newline-eof)
server.rb:10:1: C: [Corrected] Layout/IndentationWidth: Use 2 (not 4) spaces for
indentation. (https://rubystyle.guide#spaces-indentation)
  content '<h1>Hello, world!</h1>'
^^^^^trailing blank lines detected. (https://rubystyle.guide#newline-eof)

2 files inspected, 4 offenses detected, 4 offenses corrected
```

`cookstyle -a` will detect and auto-correct syntax errors within the directory you are located. You may want to save a copy of the original file (or utilize git) before running `cookstyle -a` just in case you don't like the correction(s) made.

Notice that the line number where the offense exists is indicated. This command was run from : /home/chef/chef-repo/cookbooks/apache/recipes/apache/recipes

Exercise



Group Lab: cookstyle

In this brief group lab exercise you will run `cookstyle`.

GL: Run cookstyle at the cookbook level



```
> cd /home/chef/chef-repo/cookbooks/workstation
```

```
> cookstyle
```

```
Inspecting 5 files  
.RCC.
```

```
Offenses:
```

```
metadata.rb:2:1: R: Chef/Sharing/DefaultMetadata information from the cookbook generator. Add actual maintainer 'The Authors'  
^^^^^^^^^^^^^^^^^^^^  
metadata.rb:3:1: R: Chef/Sharing/DefaultMetadataMaintainer: Metadata contains default maintainer information from the cookbook generator. Add actual cookbook maintainer information to the metadata.rb. (https://docs.chef.io/workstation/cookstyle/chef_sharing_defaultmetadatamaintainer)  
maintainer_email 'you@example.com'  
^^^^^^^^^^^^^^^^^^^^  
recipes/default.rb:6:2: C: [Correctable] Layout/TrailingWhitespace: Trailing whitespace detected.  
(https://rubystyle.guide#no-trailing-whitespace) ...
```

In this example, 5 files were inspected. Depending on how you typed your code into the cookbook's files, you should see quite a few non-critical syntax violations.

'cd' command

Execute on: **workstation**

Execute from: **anywhere**

'cookstyle' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation**

GL: Run cookstyle at the recipes level



```
> cd recipes  
> cookstyle
```

```
Inspecting 2 files  
CC  
  
Offenses:  
  
default.rb:6:2: C: [Correctable] Layout/TrailingSpace  
(https://rubystyle.guide#no-trailing-whitespace)  
#  
^  
  
default.rb:8:1: C: [Correctable] Layout/EmptyComment: Source code comment is empty.  
#  
^^  
  
default.rb:8:2: C: [Correctable] Layout/TrailingWhitespace: Trailing whitespace detected.  
(https://rubystyle.guide#no-trailing-whitespace)  
#  
^ ...
```

Now you should see fewer files inspected and fewer offenses because 'cookstyle' only inspected the files in the recipes directory.

'cd' command

Execute on: **workstation**

Execute from: : /home/chef/chef-repo/cookbooks/workstation

'cookstyle' command

Execute on: **workstation**

Execute from: /home/chef/chef-repo/cookbooks/workstation/recipes

GL: Run cookstyle on an individual file



```
> cookstyle default.rb
```

```
Inspecting 1 file
C

Offenses:

default.rb:6:2: C: [Correctable] Layout/Trailing
(/https://rubystyle.guide#no-trailing-whitespace)
#
^

default.rb:8:1: C: [Correctable] Layout/EmptyComment: Source code comment is empty.
#
^^

default.rb:8:2: C: [Correctable] Layout/TrailingWhitespace: Trailing whitespace detected.
(/https://rubystyle.guide#no-trailing-whitespace)
#
^ ...
```

While still in ~\cookbooks\workstation\recipes, run
cookstyle default.rb
You should see only offenses in the file specified.

'cookstyle' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/cookbooks/workstation/recipes**



Review

Review Questions

1. What command should you use to create a cookbook?
2. What is the benefit of using include_recipe?
3. Where do you set a cookbook version?
4. What is cookstyle and from where can you get it?

Answers:

1. chef generate cookbook <Path_If_Needed> COOKBOOKNAME
2. You get to use other people's cookbooks, or multiple cookbooks within a single recipe.
3. metadata.rb
4. `cookstyle` is an easy-to-use linting tool that's included with Chef Workstation.

Review



Q&A

What questions can we answer for you?

- Cookbooks
- Recipes
- Run-lists
- include_recipe method

What questions can we help you answer?

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



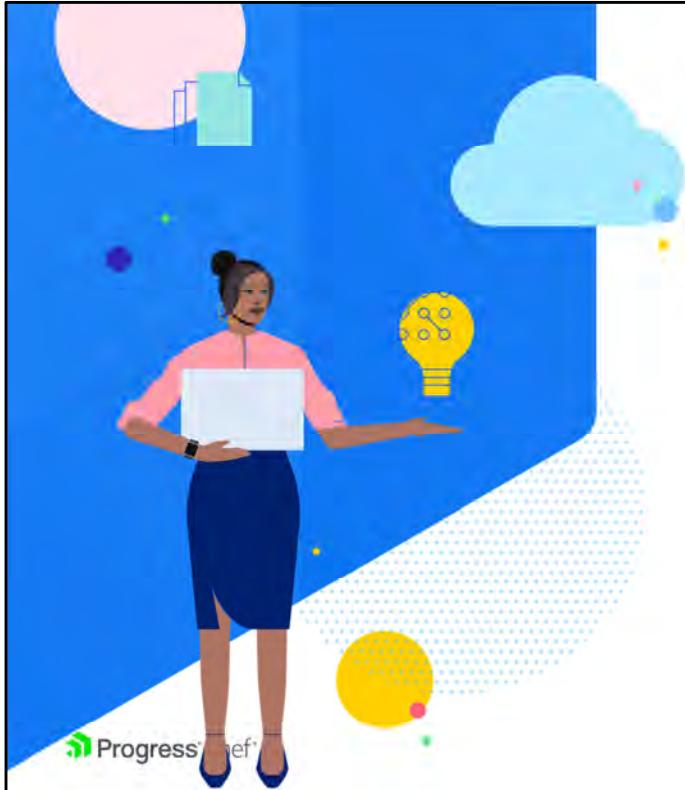


Ohai and the Node Object

Finding and Displaying Information About a System



4- 1



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

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.
4- 2

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

Concept



Managing a large number of servers

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?

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.



Concept

Some useful system data

- platform
- hostname
- memory
- CPU - MHz

Some data that would be useful to capture:

platform
hostname
memory
CPU megahertz of a system.



Concept

Ohai!

Ohai is a tool for collecting system configuration data, which it then provides to Chef Infra Client to use in cookbooks. Chef Infra Client runs Ohai at the start of every Chef Infra run to determine system state.

The attributes that Ohai collects are called automatic attributes. Chef Infra Client uses these attributes to ensure that nodes are in the desired state after each configuration run.

<https://docs.chef.io/ohai/>

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

<https://docs.chef.io/ohai/>

Concept



Ohai!

The types of attributes Ohai collects include but are not limited to:

- Operating System
- Network
- Memory
- Disk
- CPU

<https://docs.chef.io/ohai/>



Concept

All about the system

Ohai queries the operating system with a number of commands.

The data is presented in a series of key => value pairs and written in JSON (JavaScript Object Notation).

<https://docs.chef.io/ohai/>



© 2022 Progress Software Corporation and/or its subsidiaries and/or affiliates. All rights reserved.

4- 7

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

<https://docs.chef.io/ohai/>

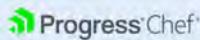
Concept



The Node Object

The node object is a representation of our system. It stores all the attributes found about the system.

<https://docs.chef.io/nodes/#attributes>



© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.

4- 8

The node object is a representation of our system. It stores all the 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, and so on.

Let's look at using the node object to retrieve the platform, hostname, total memory, and cpu megahertz.

Extra Notes if asked: If you're in local mode (`chef-client --local-mode` or `chef-client -z`) then the node object should be `/home/chef/nodes`. If you're not using local mode (just `chef-client`) then it doesn't get created locally but is uploaded to Chef Server.

`client-client` calls `ohai` at the start of its run to gather up all the info in that JSON doc. Then at the end of the chef run it adds to it any attributes that came from the cookbooks. So ``node object = (ohai data) + (cookbook attributes)``

Concept



ohai + chef-client = <3

chef-client and chef-apply automatically executes ohai and stores the data about the node in an object we can use within the recipes named 'node'.

<https://docs.chef.io/ohai/>

These values are available in our recipes because `chef-client` automatically executes Ohai. This information is stored within a variable we call 'the node object'.

Exercise



GL: Details about the node

Objectives:

- Discover attributes about the system with Ohai
- Update the web page file contents, in the "apache" cookbook, to include system details
- Update the cookbook's version number
- Apply the updated recipe and verify the results

Displaying system details in the default web page definitely sounds useful.

When deploying a server it would be nice if the test page displayed some details about the system. Together, let's walk through finding out these details, and then update the content attribute of the file resource to include this new content.

GL: Running Ohai

```
> ohai

{
  "kernel": {
    "name": "Linux",
    "release": "4.14.287-215.504.amzn2.x86_64",
    "version": "#1 SMP Wed Jul 13 21:34:43 UTC 2022",
    "machine": "x86_64",
    "processor": "x86_64",
    "os": "GNU/Linux",
    "modules": [
      "xt_conntrack": {
        "size": "16384",
        "refcount": "2"
      },
      "ipt_MASQUERADE": {
        "size": "16384",
        "refcount": "2"
      }
    ]
  }
}
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai

```
> ohai | more
```

```
{  
  "kernel": {  
    "name": "Linux",  
    "release": "4.14.287-215.504.amzn2.x86_64",  
    "version": "#1 SMP Wed Jul 13 21:34:43 UTC 2022",  
    "machine": "x86_64",  
    "processor": "x86_64",  
    "os": "GNU/Linux",  
    "modules": {  
      "xt_conntrack": {  
        "size": "16384",  
        "refcount": "2"  
      },  
      "ipt_MASQUERADE": {  
        "size": "16384",  
        "refcount": "2"  
      }  
    ...  
  }
```

In this task we are piping the output to `more` since the output is so extensive. Just use your spacebar to page down.

In this task we are piping the output to 'more' since the output is so extensive.

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai to show the platform

```
> ohai platform
```

```
[  
  "amazon"  
]
```

You can also limit the output using
an attribute like in this example.

We can specify specific node attributes by providing the node attribute's key.

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai to show the hostname

```
> ohai hostname
```

```
[  
  "ip-172-31-73-167"  
]
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai to show the memory

```
> ohai memory
```

```
{  
  "swap": {  
    "cached": "0kB",  
    "total": "0kB",  
    "free": "0kB"  
  },  
  "hugepages": {  
    "total": "0",  
    "free": "0",  
    "reserved": "0",  
    "surplus": "0"  
  },  
  "directmap": {  
    "4k": "143272kB",  
    "2M": "876544kB",  
  ...  
}
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai to show the total memory

```
> ohai memory/total
```

```
[  
  "979152kB"  
]
```

GL: Running Ohai to show the CPU

```
> ohai cpu | more
```

```
{
  "numa_node_cpus": {
    "0": [
      0,
      1
    ]
  },
  "vulnerability": {

  },
  "architecture": "x86_64",
  "cpu_opmodes": [
    "32-bit",
    "64-bit"
  ],
  "byte_order": "little endian",
  "cpus": 2,
  "cpus_online": 2,
  ...
}
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai to show the first CPU

```
> ohai cpu/0

{
  "vendor_id": "GenuineIntel",
  "family": "6",
  "model": "85",
  "model_name": "Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz",
  "stepping": "7",
  "mhz": "2499.996",
  "bogomips": "4999.99",
  "cache_size": "36608 KB",
  "physical_id": "0",
  "core_id": "0",
  "cores": "1",
  "flags": [
    "3dnowprefetch",
    "abm",
    "adx",
    ...
}
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Running Ohai to show the first CPU MHz

```
> ohai cpu/0/mhz
```

```
[  
  "2499.996"  
]
```

Execute on: **workstation**

Execute from: **anywhere**

Concept



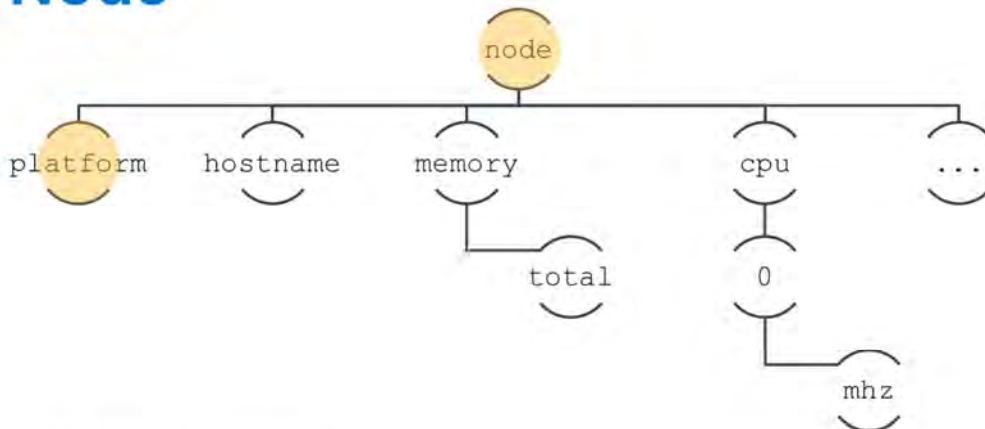
A Closer Look at the Node Object

As mentioned previously, the node object is accessible within recipes as well as from the command line.

Let's take a look at the syntax.

<https://docs.chef.io/nodes/#attributes>

The Node



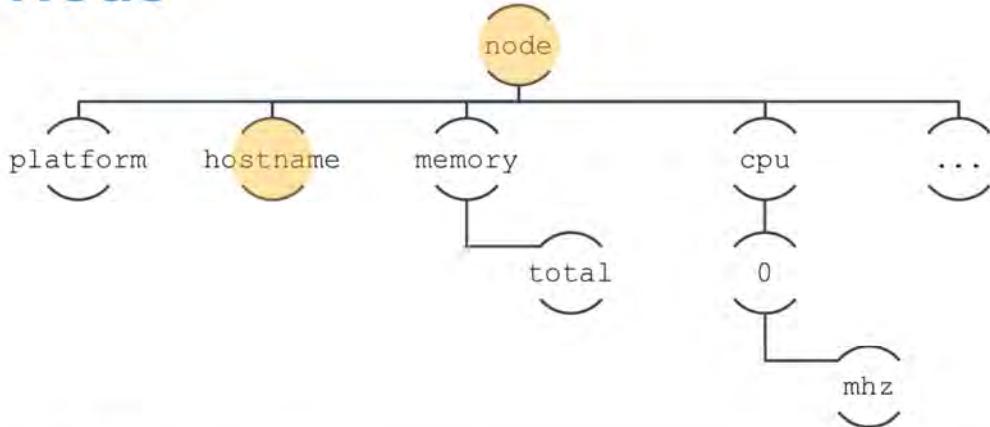
```
CLI: ohai platform
```

```
RECIPE: node['platform']
```

```
OUTPUT: amazon
```

Here is a visual representation of the node object. To access the 'platform' node attribute from within a recipe we would use the syntax found here.

The Node



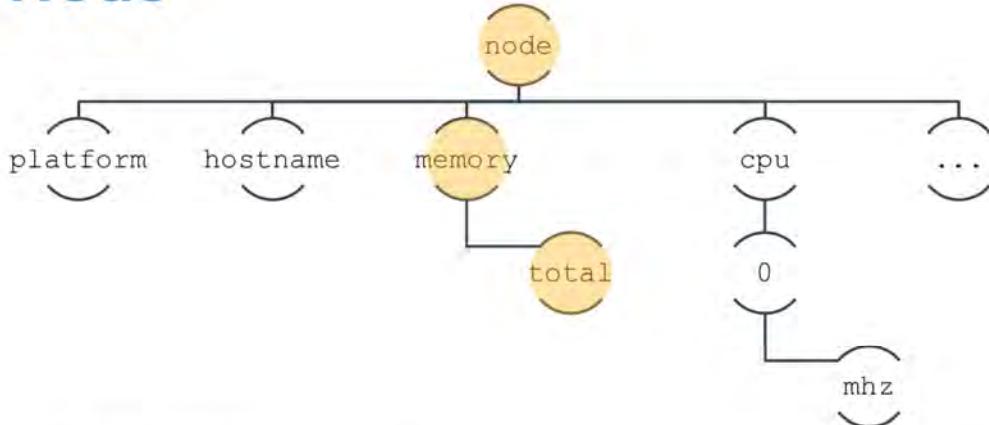
```
CLI: ohai hostname
```

```
RECIPE: node['hostname']
```

```
OUTPUT: "ip-172-31-73-167"
```

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

The Node



```
CLI: ohai memory/total
```

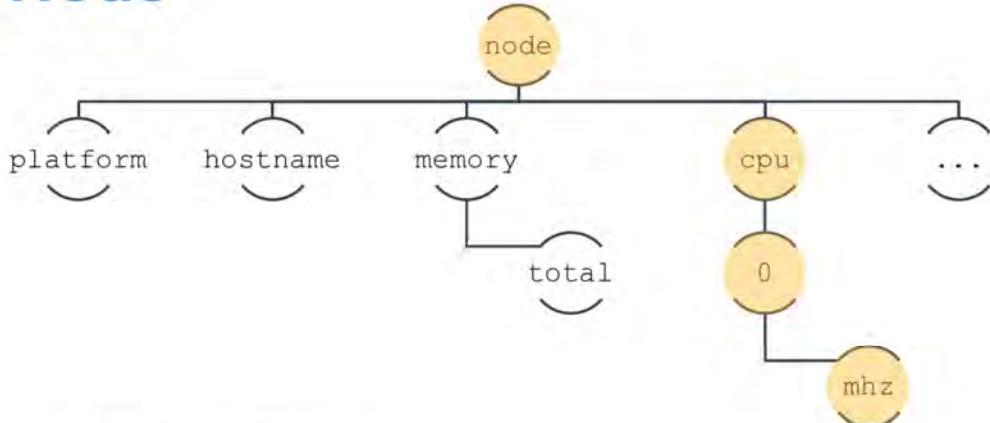
```
RECIPE: node['memory']['total']
```

```
OUTPUT: "979152kB"
```

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 the subset of keys and values at that level, and then immediately select to return the total value.

The Node



```
CLI: ohai cpu/0/mhz
```

```
RECIPE: node['cpu']['0']['mhz']
```

```
OUTPUT: "2499.996"
```

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



Concept

String Interpolation

```
I have 4 apples  
  
apple_count = 4  
puts "I have #{apple_count} apples"
```

http://en.wikipedia.org/wiki/String_interpolation#Ruby

String interpolation allows us to access variables within strings. Using this we can resolve a variable like 'apple_count' to the value assigned to it.

Concept



String Interpolation

```
I have 4 apples  
  
apple_count = 4  
puts "I have #{apple_count} apples"
```

http://en.wikipedia.org/wiki/String_interpolation#Ruby

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



Concept

String Interpolation

```
I have 4 apples  
  
apple_count = 4  
puts "I have #{apple_count} apples"
```

http://en.wikipedia.org/wiki/String_interpolation#Ruby

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.

Exercise



GL: Details About the Node

Objectives:

- ✓ Discover attributes about the system with Ohai
- Update the web page file contents, in the "apache" cookbook, to include system details
- Update the cookbook's version number
- Apply the updated recipe and verify the results

Displaying system details in the default web page definitely sounds useful.

GL: Details about the node

cookbooks/apache/recipes/server.rb

```
package 'httpd'

file '/var/www/html/index.html' do

  content "<h1>Hello, world!</h1>
<h2>PLATFORM: #{node['platform']}</h2>
<h2>HOSTNAME: #{node['hostname']}</h2>
<h2>MEMORY:  #{node['memory']['total']}
```



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

4-29

Let's update the content property first to use double quotes. Then we want to use String Interpolation and insert the value of the various node object values for platform, hostname, memory, and cpu.

Execute on: **workstation**

Final recipe:

```
package 'httpd'

file '/var/www/html/index.html' do

  content "<h1>Hello, world!</h1>
<h2>PLATFORM: #{node['platform']}</h2>
<h2>HOSTNAME: #{node['hostname']}</h2>
<h2>MEMORY:  #{node['memory']['total']}
```

Exercise



GL: Details About the Node

Objectives:

- ✓ Discover attributes about the system with Ohai
- ✓ Update the web page file contents, in the "apache" cookbook, to include system details
 - Update the cookbook's version number
 - Apply the updated recipe and verify the results

Displaying system details in the default web page definitely sounds useful.

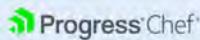


Concept

Cookbook Versions

A cookbook version represents a set of functionality that is different from the cookbook on which it is based.

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



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

4-31

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 hello message. Now the page displays the hello message with additional details about the system. The changes that we finished are new features of the cookbook.

Concept

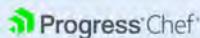


Semantic Versions

Given a version number **MAJOR.MINOR.PATCH**
increment the:

- **MAJOR** version when you make backwards incompatible API changes
- **MINOR** version when you add functionality in a backwards-compatible manner
- **PATCH** version when you make backwards-compatible bug fixes and refactoring of code

<http://semver.org>



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

4- 32

Cookbooks use semantic versions. 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.

Concept



Major, Minor, or Patch?

What kind of changes did you make to the cookbook?

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

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. That sounds like a Minor change based on the definitions provided by the Semantic Versioning documentation.

GL: Update the cookbook version

cookbooks/apache/metadata.rb

```
name 'apache'  
maintainer 'The Authors'  
maintainer_email 'you@example.com'  
license 'All Rights Reserved'  
description 'Installs/Configures apache'  
version '0.2.0'  
chef_version '>= 16.0'
```

Exercise



GL: Details About the Node

Objectives:

- ✓ Discover attributes about the system with Ohai
- ✓ Update the web page file contents, in the "apache" cookbook, to include system details
- ✓ Update the cookbook's version number
- Apply the updated recipe and verify the results

Displaying system details in the default web page definitely sounds useful.

GL: Return home and apply apache cookbook

```
> cd /home/chef/chef-repo
> sudo chef-client --local-mode -r "recipe[apache]"

Resolving cookbooks for run list: ["apache"]
Synchronizing cookbooks:
- apache (0.2.0)
Installing cookbook gem dependencies:
Converging 3 resources
Recipe: apache::server
* yum_package[httpd] action install (up to date)
* file[/var/www/html/index.html] action create
- update content in file /var/www/html/index.html from 17d291 to 6685bc
--- /var/www/html/index.html 2022-09-14 20:53:18.512331478 +0000
+++ /var/www/html/.chef-index20220919-14026-jqv4ii.html 2022-09-19 18:14:36.294437456 +0000
@@ -1,2 +1,6 @@
<h1>Hello, world!</h1>
+ IPADDRESS: 172.31.73.167
+ HOSTNAME : ip-172-31-73-167
+ MEMORY   : 979152kB
+ CPU       : 2499.996
* service[httpd] action enable (up to date)
* service[httpd] action start (up to date)

Running handlers:
Running handlers complete
Infra Phase complete, 1/4 resources updated in 03 seconds
```



© 2023 Progress Software Corporation. All rights reserved.

4-36

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 'apache' cookbook's default recipe.

'cd' command

Execute on: **workstation**
Execute from: **anywhere**

'chef-client' command

Execute on: **workstation**
Execute from: **anywhere**

GL: Verify the default page returns the details

```
> cat /var/www/html/index.html
```

```
<h1>Hello, world!</h1>
PLATFORM: amazon
HOSTNAME : ip-172-31-73-167
MEMORY   : 979152kB
CPU       : 2499.996
```

GL: curl the apache web server

```
> curl localhost
```

```
<h1>Hello, world!</h1>
PLATFORM: amazon
HOSTNAME : ip-172-31-73-167
MEMORY    : 979152kB
CPU       : 2499.996
```

Exercise



GL: Details About the Node

Objectives:

- ✓ Discover attributes about the system with Ohai
- ✓ Update the web page file contents, in the "apache" cookbook, to include system details
- ✓ Update the cookbook's version number
- ✓ Apply the updated recipe and verify the results

Displaying system details in the default web page definitely sounds useful.

Review Questions



Review

1. What is the node object and when is this generated?
2. How are the details about the system available within a recipe?
3. What is the major difference between a single-quoted string and a double-quoted string?

Answer these questions.

1. The Node Object is a .json representation of all the system attributes of a node. i.e. The specific OS, kernel, version, etc., the internal and external hostnames and IPs, and the filesystems. It's generated at the beginning of a chef-client run, when Ohai runs!
2. You can use `node[attributes]` in a recipe to access details about the system.
3. In Ruby, single-quoted strings don't allow you to use symbols; in order to interpolate code within a string, you need to use double-quoted strings.

Review



Q&A

What questions can we help you answer?

- Ohai
- Node Object
- Node Attributes
- String Interpolation

With that we have added all of the requested features.

What questions can we help you answer?

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





Testing Cookbooks – Prelude

Setting up our test environment in advance of
this lesson



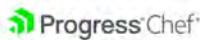
5- 1

Before we start this lesson

Lets run a few commands that could save us about 15-20 minutes later

Don't worry about what we're doing – we'll explain as we go through the lesson

Just follow along



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

GL: Move to the home dir

```
> cd /home/chef
```

Execute on: **workstation**

Execute from: **Anywhere**

GL: Replace the kitchen config

```
> cp ~/kitchen-template.yml ~/chef-repo/cookbooks/workstation/kitchen.yml
```

For anyone who has used ChefDK in the past, in the newer Chef Workstation we are using, the leading dot on the kitchen.yml file is no longer needed.

Copy the template kitchen configuration file from the home directory into the workstation cookbook directory. This will replace the existing kitchen configuration with the one set up to work within Amazon EC2.

cp ~/kitchen-template.yml ~/chef-repo/cookbooks/workstation/kitchen.yml

Execute on: **workstation**

Execute from: **Anywhere**

GL: Add your name and company

~/chef-repo/cookbooks/workstation/kitchen.yml

... CONTENT OMITTED FOR CLARITY...

```
instance_type: m3.large
tags:
  # Replace YOURNAME and YOURCOMPANY here
  Name: "Chef Training Node for YOURNAME, YOURCOMPANY" ← Line 14 – Add your name and company
  created-by: "test-kitchen"
  user: <%= ENV['USER'] %>
```

Edit the file to insert your name and company information.

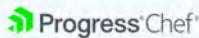
GL: Add chef-client 17 requirements

~/chef-repo/cookbooks/workstation/kitchen.yml

... CONTENT OMITTED FOR CLARITY...

```
provisioner:  
  name: chef_zero  
  cookbook_path: /home/chef/chef-repo/cookbooks  
  client_rb:  
    chef_license: accept  
    product_name: chef  
    product_version: 17
```

← Add these lines below line 20.



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

5- 6

```
provisioner:  
  name: chef_zero  
  cookbook_path: /home/chef/chef-repo/cookbooks  
  client_rb:  
    chef_license: accept  
    product_name: chef  
    product_version: 17
```

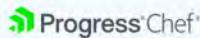
GL: Add your cookbook

`~/chef-repo/cookbooks/workstation/kitchen.yml`

... CONTENT OMITTED FOR CLARITY...

```
suites:  
- name: default  
  run_list:  
    # Replace with the name of the COOKBOOK  
    - recipe[COOKBOOK::default]  
  attributes:
```

Replace 'COOKBOOK' with 'workstation' in the suites section.



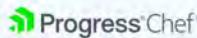
© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

5- 7

Edit the file to insert your name and company information.

GL: Create the Test Kitchen instance

```
> cd ~/chef-repo/cookbooks/workstation
> kitchen create
----> Starting Test Kitchen (v3.2.2)
----> Creating <default-CentOS-74-100>...
      Detected platform: centos version 7.4 on x86_64. Instance Type: t2.small.
...
Instance <i-083fdec5159801822> requested.
  Polling AWS for existence, attempt 0...
    EC2 instance <i-083fdec5159801822> created.
. Waited 35/600s for instance <i-083fdec5159801822> to become ready.
Waiting for SSH service on ec2-34-224-15-154.compute-1.amazonaws.com:22, retrying
in 3 seconds
  Waiting for SSH service on ec2-34-224-15-154.compute-1.amazonaws.com:22,
retrying in 3 seconds
    [SSH] Established
    Finished creating <default-CentOS-74-100> (0m51.69s).
----> Test Kitchen is finished. (0m53.15s)
```



Execute `kitchen converge` to validate that our workstation cookbook's default recipe is able to converge on the Linux instance.

Instructor Note: This process can easily take 5 minutes to complete the kitchen create command

'cd' command

Execute on: **workstation**

Execute from: **anywhere**

'kitchen' command

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation/**



Testing Cookbooks

Validating Our Recipes in Virtual Environments





Objectives

After completing this module, you should be able to:

- Use Test Kitchen to verify that your recipes converge on a virtual instance
- Refer to the InSpec documentation
- Write and execute tests

© 2020 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

5- 10

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

Concept



We should test cookbooks

As we start to define our infrastructure as code we also need to start thinking about testing it.

We have an automated way to ensure code accomplishes the intended goal and help the team understand its intent. It's called Test Kitchen.

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?

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 less of a chance of breaking something. Whether returning to the cookbook code tomorrow or in six months.

Steps to verify cookbooks

Create Virtual Machine

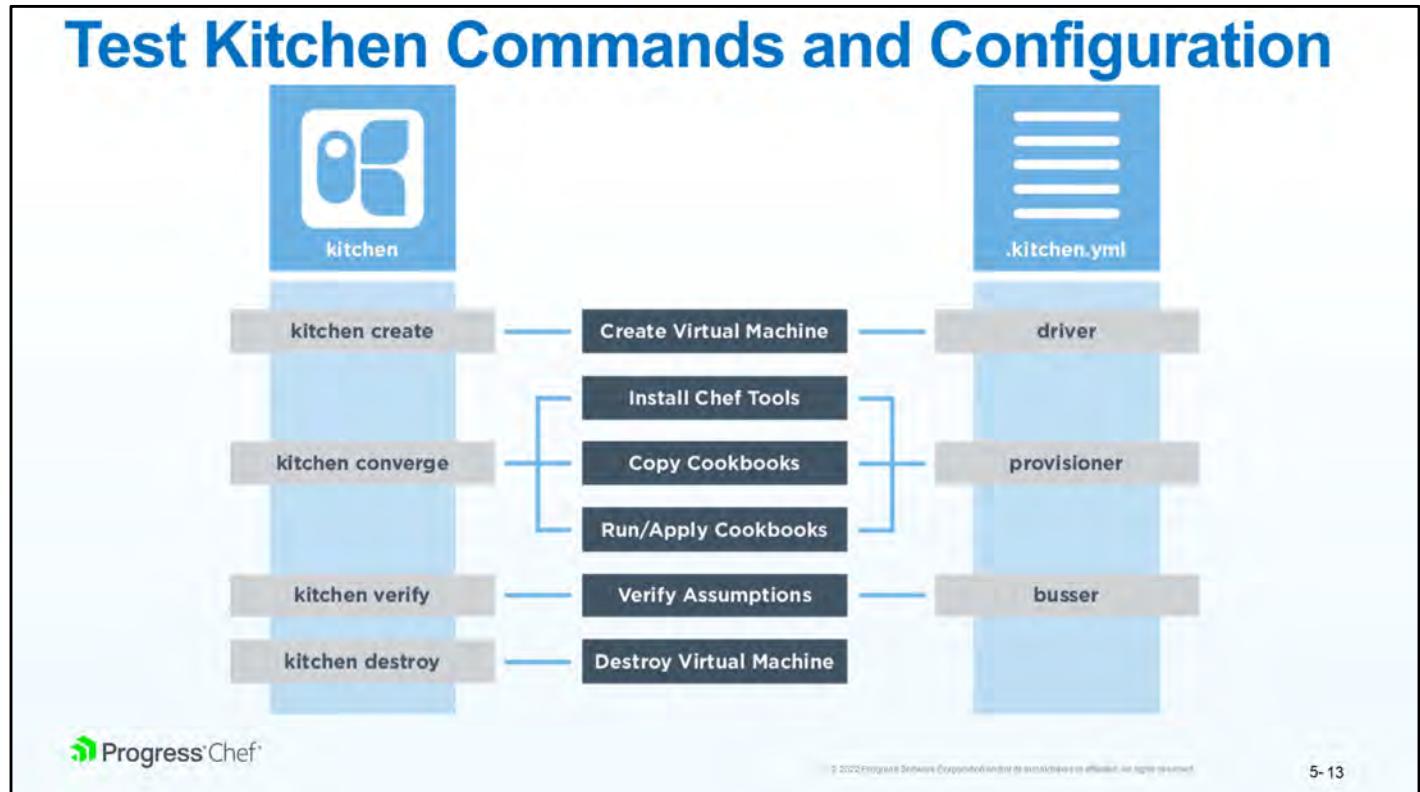
Install Chef Tools

Copy Cookbooks

Run/Apply Cookbooks

Verify Assumptions

Destroy Virtual Machine



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

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

On the right are the kitchen configuration fields that map to the stages of the testing lifecycle.

These commands and the configuration file will be explained in more detail.

Exercise



Test Configuration

Objectives:

- *Configure the "workstation" cookbook to test against Linux platform*
- *Apply the "workstation" cookbook's default recipe to that virtual machine*

What are we running in production? Maybe I could test the cookbook against a virtual machine.

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.



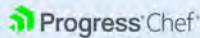
Concept

kitchen.yml

When chef generates a cookbook, a default kitchen.yml is created. It contains kitchen configuration for the driver, provisioner, platform, and suites.

Reminder: The newer Chef Workstation software no longer uses a leading dot in the kitchen.yml file name.

<https://kitchen.ci/docs/getting-started/kitchen-yml/>



© 2022 Progress Software Corporation and its subsidiaries or affiliates. All rights reserved.

5- 15

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.

<https://kitchen.ci/docs/getting-started/kitchen-yml/>

Demo: The kitchen driver

~/chef-repo/cookbooks/workstation/kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

# Uncomment the following verifier...
# default verifier)
# verifier:
#   name: inspec

# KITCHEN CONFIGURATION CONTINUES...
```

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

Today we're using an EC2 driver that will spin up an instance in AWS that we can run Test Kitchen against.

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.

Instructor Note: Testing on this remote workstation requires that we use a completely different kitchen configuration file. This configuration file is generated for use on local machines which is not ideal when working on a virtual machine in the cloud.

Demo: The kitchen provisioner

```
~/chef-repo/cookbooks/workstation/kitchen.yml
```

```
---
```

```
driver:
```

```
  name: vagrant
```

```
provisioner:
```

```
  name: chef_zero
```

```
# Uncomment the following verifier...
```

```
# default verifier)
```

```
# verifier:
```

```
#   name: inspec
```

```
# KITCHEN CONFIGURATION CONTINUES...
```

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`.

Demo: The kitchen platforms

```
~/chef-repo/cookbooks/workstation/kitchen.yml
```

```
# TOP PART OF KITCHEN CONFIGURATION

platforms:
  - name: ubuntu-14.04
  - name: centos-7.1

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

This is a list of Operating 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.

None of these platforms are ones that we are interested in supporting. We can specify various different Operating Systems and Versions like Windows.

Demo: The kitchen suites

~/chef-repo/cookbooks/workstation/kitchen.yml

```
# TOP PART OF KITCHEN CONFIGURATION

platforms:
  - name: ubuntu-14.04
  - name: centos-7.1

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'.

Demo: The kitchen suites

```
~/chef-repo/cookbooks/workstation/kitchen.yml
```

```
# TOP PART OF KITCHEN CONFIGURATION

platforms:
  - name: ubuntu-14.04
  - name: centos-7.1

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.



Concept

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.

Example: View the Test Kitchen matrix

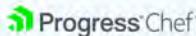
```
> kitchen list
```

Instance	Driver	Provisioner	Last Action
default-ubuntu-1204	Vagrant	ChefZero	<Not Created>
default-centos-65	Vagrant	ChefZero	<Not Created>

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

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

REMOTE

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.

5-22

In this example, if we were to run this command on a local workstation, 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 and second the CentOS 6.5 platform.

Example: View the Test Kitchen matrix

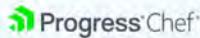
```
> kitchen list
```

Instance	Driver	Provisioner	Last Action
default-ubuntu-1204	Vagrant	ChefZero	<Not Created>
default-centos-65	Vagrant	ChefZero	<Not Created>

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

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

REMOTE

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.5-23

Here we are highlighting the centos 6.5 platform.



Concept

Virtualization

Vagrant is great for local development but when building cookbooks on a virtual machine in the cloud we will not need to use a local virtualization tool.

Instead, we will ask Amazon EC2 to provision nodes for us to test against.

Test Kitchen is a tool that you will often use on your local workstation. That is why the default configuration file uses vagrant. Vagrant allows Test Kitchen to communicate with VirtualBox to provision test instance that you can easily spin up and tear down. However, VirtualBox does not work on virtual instances. We are currently using a virtual instance in the Amazon EC2 cloud. So instead we are going to configure Test Kitchen to provision virtual instances for us in the Amazon EC2 cloud.

We have provided a template file that has all the settings properly configured for our Training VM's in the Amazon EC2 cloud. This is file we copied into the cookbook at the beginning of this module. Now we will edit it with specific values.

Review: You added your name and company

~/chef-repo/cookbooks/workstation/kitchen.yml

```
---
```

```
driver:
```

```
  name: ec2
```

```
  # ... OTHER DRIVER DETAILS ...
```

```
  instance_type: m3.large
```

```
tags:
```

```
  # Replace YOURNAME and YOURCOMPANY here
```

```
  Name: "Chef Training Node for YOURNAME, YOURCOMPANY"
```

```
  created-by: "test-kitchen"
```

```
  user: <%= ENV['USER'] %>
```

```
# ... REMAINDER OF THE CONFIGURATION FILE ...
```

You edited the file to insert your name and company information.

Review: You added chef-client 17 requirements

~/chef-repo/cookbooks/workstation/kitchen.yml

... CONTENT OMITTED FOR CLARITY...

```
provisioner:  
  name: chef_zero  
  cookbook_path: /home/chef/chef-repo/cookbooks  
client_rb:  
  chef_license: accept  
  product_name: chef  
  product_version: 17
```

This addition allows the chef license to be auto-accepted later when you use kitchen converge.

Review: You added your cookbook

```
~/chef-repo/cookbooks/workstation/kitchen.yml
```

```
# ... TOP PART OF THE CONFIGURATION FILE ...

suites:
  - name: default
    run_list:
      # Replace with the name of the COOKBOOK
      - recipe[workstation::default]
    attributes:
```

Edit the file to define the correct test suite. In this instance we want to test the 'workstation' cookbook's 'default' recipe.

GL: Ensure you are in ~\cookbooks\workstation

```
> cd ~/chef-repo/cookbooks/workstation
```

GL: Look at the Test Kitchen matrix

```
> kitchen list
```

Instance	Driver	Provisioner	Verifier	Transport	Last Action	Last Error
default-CentOS-74-100	Ec2	ChefInfra	Inspec	Ssh	Created	<None>

Run the `kitchen list` command to display our test matrix. You should see a single instance. This example is as if you had run `kitchen create` at the beginning of this module.

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation**

Exercise



Test Configuration

Objectives:

- ✓ Configure the "workstation" cookbook to test against a Linux platform
- Apply the "workstation" cookbook's default recipe to that virtual machine

What are we running in production? Maybe I could test the cookbook against a virtual machine.

Alright, the 'workstation' cookbook configuration has been updated properly and we verified that with `kitchen list`. Now to test our cookbook's recipe against that system we are going to need to use Test Kitchen.



Concept

Kitchen Create

kitchen
create

kitchen
converge

kitchen
verify

```
> kitchen create [INSTANCE|REGEXP|all]
```

Create one or more instances.

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 ec2 driver to create a Linux instance.

Instructor Note: The command does allow you to create specific instances by name or all instances that match a provided criteria.

Concept



Kitchen Converge

kitchen
create

kitchen
converge

kitchen
verify

```
> kitchen converge [INSTANCE|REGEXP|all]
```

Create the instance (if necessary) and
then apply the run list to one or more
instances.

Creating an image gives us an 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.

Instructor Note: It also, like the `kitchen create` commands, defaults to all instances when executed without any parameters and is capable of accepting parameters to converge a specific instance or all instances that match the provided criteria.

GL: Converge the cookbook

```
> kitchen converge
```

```
Using Policyfile 'workstation' at revision ...
Converging 2 resources
  Recipe: workstation::setup
    * yum_package[tree] action install
      - install version 0:1.6.0-10.el7.x86_64 of package tree
    * file[/etc/motd] action create
      - update content in file /etc/motd from e3b0c4 to 4b096f
        --- /etc/motd      2013-06-07 14:31:32.000000000 +0000
        +++ /etc/.chef-motd20221027-5680-ymrf0m6      2022-10-27 21:07:56.522652487 +0000
        @@ -1,6 +1,12 @@
        +Property of ...
        +
        + PLATFORM: centos
        + HOSTNAME : ip-172-31-47-253
        + MEMORY   : 1880520kB
        + CPU       : 2400.039

  Running handlers:
  Running handlers complete
  Infra Phase complete, 2/2 resources updated in 20 seconds
```

We will discuss Policyfiles later in this course but for now, `kitchen converge` created the Policyfile for us.

Execute `kitchen converge` to validate that our workstation cookbook's default recipe is able to converge on the Linux instance.

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation**

Exercise



Test Configuration

Objectives:

- ✓ Configure the "workstation" cookbook to test against a Linux platform
- ✓ Apply the "workstation" cookbook's default recipe to that virtual machine

What are we running in production? Maybe I could test the cookbook against a virtual machine.

We successfully configured the workstation cookbook to use the Amazon EC2 driver and target a Linux Instance.



Concept

Test Kitchen

What does this test when kitchen converges a recipe?

And what does it NOT test when kitchen converges a recipe?

So what does this test when kitchen converges a recipe?

What does it NOT test when kitchen converges a recipe?

Instructor Note: Converging the recipe is able to validate that our recipe is defined without error. However, converging a particular recipe does not validate that the intended goal of the recipe has been successfully executed.



Concept

Test Kitchen

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

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

Instructor Note: Converging the instance ensured that the recipe was able to install a package, write out a file, and start and enable a service. But was it able to check to see if the system was configured correctly, or if our instance is serving up our custom home page? This is where the 'kitchen verify' command comes in.

Exercise



GL: The First Test

Objectives:

- Write a test that asserts that we have installed tree when the "workstation" cookbook's default recipe is applied
- Execute the test that we have written

Converging seems to validate that the recipe runs successfully. But does it assert what actually is installed?

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.

To do that we are going to need to learn a few more kitchen commands and a way to express our expectations in a language called InSpec.



Concept

InSpec

InSpec tests your servers' actual state by executing commands locally or via SSH, via WinRM, via Docker API and so on against a test instance.

<https://docs.chef.io/inspec/>



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

5- 38

InSpec is one of many possible test frameworks that Test Kitchen supports. It is a popular choice for those doing Chef cookbook development because InSpec is written by Chef and 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. InSpec adds a number of helpers to RSpec to make it easy to test the state of a system.

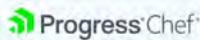
<https://docs.chef.io/inspec/>

Example: Is tree installed?

```
describe 'workstation::default' do

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

end
```

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.5- 39

Here is an InSpec example.

In this test we verify that the `tree` package is indeed installed in our Test Kitchen virtual machine.



Concept

Where do Tests Live?

```
~/chef-repo/cookbooks/workstation/test/integration/default/default_test.rb
```

By default, Test Kitchen will look for tests to run under this directory.

Note: Tests can actually live wherever you want them to live. The kitchen.yml gives you the flexibility to point to various places as it just loads tests based on the parameters.

Let's take a moment to describe the reason behind this directory for the tests. Within our cookbook we define a test directory and within that test directory we define another directory named 'recipes'. This is the same path value specified within our kitchen configuration file in the 'suites' section.

Note: In older versions of ChefDK (and subsequently in cookbooks created from those older versions of ChefDK), this directory is under "smoke" instead of being under "integration", such as: <cookbook_name>\test\smoke\default\default_test.rb

GL: Replace the existing file contents

```
~/chef-repo/cookbooks/workstation/test/integration/default/default_test.rb
```

```
describe 'workstation::default' do

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

end
```

The content of the previous test file provided InSpec example. We need to replace the existing content with the above content.

```
describe "workstation::default" do

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

end
```

Exercise



GL: The First Test

Objectives:

- ✓ Write a test that asserts that we have installed tree when the "workstation" cookbook's default recipe is applied
- Execute the test that we have written

Converging seems to validate that the recipe runs successfully. But does it assert what actually is installed?

We have now defined our first test. Now it is time to execute that test against our test instance.



Concept

Kitchen Verify

kitchen
create

kitchen
converge

kitchen
verify

```
> kitchen verify [INSTANCE|REGEXP|all]
```

Create, converge, and verify one or more instances.

The third kitchen command is 'kitchen verify'.

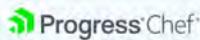
To verify an instance means to: create virtual or cloud instances (if needed); then converge the instance (if needed); and then execute a collection of defined tests against the instance.

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

Instructor Note: 'kitchen verify' works as the other commands do with regard to parameters and targeting instances.

GL: Ensure you are in the cookbook

```
> cd ~/chef-repo/cookbooks/workstation
```

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.5- 44

Change into the workstation cookbook directory.

Execute on: **workstation**

Execute from: **anywhere**

GL: Running the Test

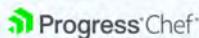
```
> kitchen verify

----> Starting Test Kitchen (v3.2.2)
----> Setting up <default-CentOS-74-100>...
      Finished setting up <default-CentOS-74-100> (0m0.00s).
----> Verifying <default-CentOS-74-100>...
      Loaded tests from {:path=>"~.home.chef.chef-
repo.cookbooks.workstation.test.integration.default"}

Profile: tests from {:path=>"~.home.chef.chef-
repo.cookbooks/workstation/test/integration/default"} (tests from
{:path=>"~.home.chef.chef-repo.cookbooks.workstation.test.integration.default"})
Version: (not specified)
Target: ssh://chef@ec2-54-146-60-92.compute-1.amazonaws.com:22

System Package tree
  ✓  is expected to be installed

Test Summary: 1 successful, 0 failures, 0 skipped
Finished verifying <default-CentOS-74-100> (0m2.88s).
```

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.5-45

Now verify our expectation using the `kitchen verify` command. If you have defined the test correctly and the system is in the desired state you should see a summary of the execution, showing 1 example completed with 0 failures.

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation**

Exercise



Lab: Additional Test

Objectives:

- *Add a test that validates that git is installed*
- *Add a test that validates a file resource ownership*
- *Run kitchen verify to validate the test meets the expectations that you defined*

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 will 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.

Instructor Note: They already tested the tree package but they have not tested the git package, or the file resource ownership. (We have not installed git on the test kitchen instance but we will test for it to show a test kitchen failure). The learner is not required to test all of the packages or a particular set of conditions with the file resource. This section is intentionally open and left to the choice of the learner. When reviewing this material with the learner the answers that follow are not the 'correct' solution; they are one solution.

Instructor Note: Allow 15 minutes to complete this exercise.

GL: Add another test that verifies git is installed

```
~/chef-repo/cookbooks/workstation/test/integration/default/default_test.rb
```

```
describe 'workstation::default' do

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

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

end
```

```
describe 'workstation::default' do
```

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

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

```
end
```

GL: Add another test verifying ownership of a file

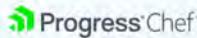
```
~/chef-repo/cookbooks/workstation/test/integration/default/default_test.rb

describe 'workstation::default' do

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

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

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

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.5-48

```
describe 'workstation::default' do

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

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

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

GL: The full test we will verify

```
~/chef-repo/cookbooks/workstation/test/integration/default/default_test.rb
```

```
describe 'workstation::default' do

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

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

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

Lab: Running the Test

```
> kitchen verify
```

```
----> Starting Test Kitchen (v3.2.2)
----> Verifying <default-CentOS-74-100>...
     Loaded tests from {:path=>"./home.chef"...
...
System Package tree
✓  is expected to be installed
✗  is expected to be installed
expected that `System Package git` is installed
✓  is expected to be owned by "root"

Test Summary: 2 successful, 1 failure, 0 skipped
>>>> -----Exception-----
>>>> Class: Kitchen::ActionFailed
>>>> Message: 1 actions failed.
>>>>   Verify failed on instance <default-CentOS-74-100>. Please see
.kitchen/logs/default-CentOS-74-100.log for more details
>>>> -----
>>>> Please see .kitchen/logs/kitchen.log for more details
>>>> Also try running `kitchen diagnose --all` for configuration
```

In this case the red text is not an error with test kitchen itself. It is indicating that 'git' is not installed on the test kitchen instance.

Now verify our expectation using the 'kitchen verify' command. If you have defined the tests correctly and the system is in the state it currently is in, you should see a summary of the execution, showing 2 examples successful and one example is a failure.

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation**

Lab: Examining the output

>

```
System Package tree
✓  is expected to be installed
✗  is expected to be installed
expected that `System Package git` is installed
✓  is expected to be owned by "root"

...
describe 'workstation::default' do
  describe package('tree') do
    it { should be_installed }
  end

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

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

5/51

Here is a side-by-side comparison of our test and the results of the test.

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation**



Concept

Kitchen Destroy



The fourth kitchen command is `kitchen destroy`.

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

Instructor Note: It works as all the other commands do with regard to parameters and targeting instances.



Concept

Kitchen Test

kitchen
destroy

kitchen
create

kitchen
converge

kitchen
verify

kitchen
destroy

```
> kitchen test [INSTANCE|REGEXP|all]
```

Destroys (for clean-up), creates, converges, verifies and then destroys one or more instances.

There is 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 first immediately destroyed. Then with a clean instance it performs 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.

Exercise



Lab: kitchen destroy

We are done with Test Kitchen so please execute 'kitchen destroy' to terminate the EC2 instance used in this module.

Lab: Run `kitchen destroy` to clean everything up

```
> kitchen destroy
```

```
----> Starting Test Kitchen (v3.2.2)
----> Destroying <default-CentOS-74-100>...
      EC2 instance <i-0e937c2f8fe53a81f> destroyed.
      Finished destroying <default-CentOS-74-100> (0m0.27s).
----> Test Kitchen is finished. (0m1.69s)
```

Important: Please be sure to run `kitchen destroy` to help us save instance costs.

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks/workstation**



Review

Review Questions

1. Why do you have to run kitchen within the directory of the cookbook?
2. Where would you define additional platforms?
3. Why would you define a new test suite?

Answer these questions.

1. We run kitchen commands from the cookbook directory because Test Kitchen is built as a unit tester, and we define units roughly as cookbooks.
2. Additional platforms for Test Kitchen are defined in the Platforms section of the kitchen.yml file
3. You might define a new test suite if you had a single cookbook but multiple recipes that helped accomplish the overall goal. For instance, in the chef-client community cookbook, there's several different methods you could choose from to configure Chef Client.



Review

Q&A

What questions can we help you answer?

- Test Kitchen
- kitchen commands
- kitchen configuration
- InSpec

What questions can we help you answer?

Generally or specifically about test kitchen, kitchen commands, kitchen configuration, or InSpec.





Templates

Desired State vs. Data



6- 1



Objectives

After completing this module, you should be able to:

- Explain when to use a template resource
- Create a template file
- Use ERB tags to display node data in a template
- Define a template resource

©2023 Progress Software Corporation and/or its subsidiaries. All rights reserved.

6- 2

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

Concept



Cleaner recipes

Previously, we updated our 'apache' cookbook to display information about our node.

We added the following content to the file resource in the setup recipe....

We were successful in displaying the details about the system instead of using hard-coded values. We added that content to the content attribute of the file resource that generates the Default page for the webserver.

Demo: The 'apache' server recipe

~/cookbooks/apache/recipes/server.rb

```
package 'httpd'

file '/var/www/html/index.html' do
  content "<h1>Hello, world!</h1>
  PLATFROM: #{node['platform']}
  HOSTNAME : #{node['hostname']}
  MEMORY    : #{node['memory']['total']}
  CPU       : #{node['cpu']['0']['mhz']}"
end

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

When you consider the syntax needed to edit this file, like when to use double quotes, or single quotes, etc., manually editing a file like this can become difficult to do.

Also, Ruby string interpolation to insert the value of a variable works for simple variables but doesn't allow you to insert any logic into what value should be used, or when.

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.

Also this is using a `file` resource. The trouble with this is it isn't scalable past anything more than a few words/lines. Once you have more than that it becomes unmanageable, especially if there are characters in there that you need to escape, like `\\`, `/`, ``", ``` etc.

Concept



Manual updating

This manual process of editing the recipe file is definitely error-prone. Especially because a human has to edit the file again before it is deployed.

So, if you are copying and pasting large amounts of text content into a double-quoted string it will be important to check that every time.

This sounds 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.



Concept

What we need

We need the ability to store the data in another file, which is in the native format of the file we are writing out but that still allows us to insert Ruby code...

...specifically, the node attributes we have defined.

It is better to store this data in another file. The file would be native to whatever format is required so it 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.



Concept

Template

A cookbook template is an Embedded Ruby (ERB) template that is used to generate files. Templates may contain Ruby expressions and statements.

Use the template resource to add cookbook templates to recipes; place the corresponding Embedded Ruby (ERB) template in a cookbook's /templates directory.

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

Let's explore templates.

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

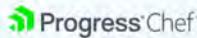
<https://docs.chef.io/resources/template/>

Example: Template file's source matches up

```
> ls cookbooks/apache/templates/default  
/home/chef/chef-repo/cookbooks/apache/templates
```

index.html.erb

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

© 2022 Progress Software Corporation. All rights reserved.6- 8

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.



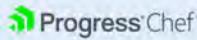
Concept

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/resources/template/>



© 2022 Progress Software Corporation. All rights reserved. All rights reserved.

6- 9

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.

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

Exercise



GL: Cleaner Recipes

- Create a template with `chef generate`
- Define the contents of the ERB template
- Change the file resource to the template resource in the 'apache' cookbook
- Use `chef-client` to apply the "apache" cookbook's "default" recipe
- Update the "apache" cookbook's version for this patch

Adding the node attributes to the index page did make it harder to read the recipe.

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

Let's start by creating the actual template file and then we will update the recipe.

GL: What can chef generate do?

```
> chef generate --help

Usage: chef generate GENERATOR [options]

Available generators:
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
helpers   Generate a cookbook helper file in libraries
resource   Generate a custom resource
repo      Generate a Chef policy repository
policyfile Generate a Policyfile for use with the install/push commands
```

Let's ask for help about the 'generate' subcommand.

Execute on: **workstation**

Execute from: **anywhere**

GL: What can chef generate template do?

```
> chef generate template --help

Usage: chef generate template [path/to/cookbook] NAME [options]
      -C, --copyright cookbook file           Name of the copyright holder -
      defaults to 'The Authors'
      -m, --email cookbook file              Email address of the author -
      defaults to ...
      -a, --generator-arg KEY=VALUE         Use to set arbitrary attribute KEY to
      VALUE in the ...
      -I, --license LICENSE                all_rights, mit, gplv2, gplv3 - defaults
      to ...
      -s, --source SOURCE_FILE             Copy content from SOURCE_FILE
      -g GENERATOR_COOKBOOK_PATH,          Use GENERATOR_COOKBOOK_PATH for the
      code_generator
      --generator-cookbook
```

Also, let's 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.

Execute on: **workstation**

Execute from: **anywhere**

GL: Use chef to generate a template

```
> cd /home/chef/chef-repo
> chef generate template cookbooks/apache index.html

Recipe: code_generator::template
 * directory[cookbooks/apache/templates] action create
   - create new directory cookbooks/apache/templates
 * template[cookbooks/apache/templates/index.html.erb] action create
   - create new file cookbooks/apache/templates/index.html.erb
   - update content in file cookbooks/apache/templates/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. The template file we want to create will be named **index.html.erb**, so we name it **index.html** in this command - the 'chef' command will automatically append the '.erb' extension for us.

'cd' command

Execute on: **workstation**

Execute from: **anywhere**

'chef' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

GL: Lets look at the template file

```
> ls -l cookbooks/apache/templates
```

```
total 0
-rw-rw-r-- 1 chef chef 0 Sep 19 17:41 index.html.erb
```

Exercise



GL: Cleaner Recipes

- ✓ Create a template with `chef generate`
- Define the contents of the ERB template
- Change the file resource to the template resource in the 'apache' cookbook
- Use `chef-client` to apply the "apache" cookbook's "default" recipe
- Update the "apache" cookbook's version for this patch

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

Let's start by creating the actual template file and then we will update the recipe.



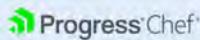
Concept

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>



© 2022 Progress Software Corporation and/or its subsidiaries. All rights reserved.

6- 16

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.

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

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.

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.

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

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.

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 does not display the result.

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

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 displays 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.

Concept



The Angry Squid

<%=

The starting tag is different. It has an equals sign. This means to print (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.

GL: Add code from file resource

cookbooks/apache/templates/index.html.erb

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>PLATFORM: #{node['platform']}</h2>
    <h2>HOSTNAME: #{node['hostname']}</h2>
    <h2>MEMORY:   #{node['memory']['total']}</h2>
    <h2>CPU Mhz:  #{node['cpu'][0]['mhz']}</h2>
  </body>
</html>
```

Don't add this code
until the next slide.
But do compare this
to the next slide.

With that in mind let's 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.

GL: Replace string interpolation with ERB

```
cookbooks/apache/templates/index.html.erb
```

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>PLATFORM: <%= node['platform'] %></h2>
    <h2>HOSTNAME: <%= node['hostname'] %></h2>
    <h2>MEMORY:   <%= node['memory']['total'] %></h2>
    <h2>CPU Mhz: <%= node['cpu']['0']['mhz'] %></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.

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>PLATFORM: <%= node['platform'] %></h2>
    <h2>HOSTNAME: <%= node['hostname'] %></h2>
    <h2>MEMORY:   <%= node['memory']['total'] %></h2>
    <h2>CPU Mhz: <%= node['cpu']['0']['mhz'] %></h2>
  </body>
</html>
```

Exercise



GL: Cleaner Recipes

- ✓ Create a template with `chef generate`
- ✓ Define the contents of the ERB template
- Change the file resource to the template resource in the 'apache' cookbook
- Use chef-client to apply the "apache" cookbook's "default" recipe
- Update the "apache" cookbook's version for this patch

GL: Remove the existing content property

`cookbooks/apache/recipes/server.rb`

```
package 'httpd'

file '/var/www/html/index.html' do

  content "<h1>Hello, world!</h1>
<h2>PLATFORM: #{node['platform']}</h2>
<h2>HOSTNAME: #{node['hostname']}</h2>
<h2>MEMORY:   #{node['memory']['total']}</h2>
<h2>CPU Mhz:  #{node['cpu'][0]['mhz']}</h2>
"
end
#... SERVICE RESOURCE..."
```

Let's open the apache cookbook's recipe named 'server.rb'.

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.

GL: Change file to template

`cookbooks/apache/recipes/server.rb`

```
package 'httpd'

template '/var/www/html/index.html' do
end

#...SERVICE RESOURCE...
```

GL: Add the source property

`cookbooks/apache/recipes/server.rb`

```
package 'httpd'

template '/var/www/html/index.html' do
  source 'index.html.erb'
end

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

The final code in server.rb should look like this:

```
package 'httpd'

template '/var/www/html/index.html' do
  source 'index.html.erb'
end

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

Exercise



GL: Cleaner Recipes

- ✓ Create a template with `chef generate`
- ✓ Define the contents of the ERB template
- ✓ Change the file resource to the template resource in the 'apache' cookbook
 - Use `chef-client` to apply the "apache" cookbook's "default" recipe
 - Update the "apache" cookbook's version for this patch

GL: Apply the cookbook

```
> cd /home/chef/chef-repo  
> sudo chef-client --local-mode -r "recipe[apache]"
```

```
Converging 3 resources  
Recipe: apache::server  
  * yum_package[httpd] action install (up to date)  
  * template[/var/www/html/index.html] action create  
    - update content in file /var/www/html/index.html from 6685bc to 986198  
      --- /var/www/html/index.html 2022-09-19 18:24:31.810040015 +0000  
      +++ /var/www/html/.chef-index20220919-24772-20ut8.html 2022-09-19 20:11:33.641736566 +0000  
      @@ -1,6 +1,10 @@  
      -<h1>Hello, world!</h1>  
      - PLATFORM: amazon  
      - HOSTNAME : ip-172-31-73-167  
      - MEMORY   : 979152kB  
      - CPU       : 2499.996  
      +<html>  
      + <body>  
      +   <h1>Hello, world!</h1>  
      +   <h2>PLATFORM: amazon</h2>  
      +   <h2>HOSTNAME: ip-172-31-73-167</h2>  
      +   <h2>MEMORY: 979152kB</h2>  
      +   <h2>CPU Mhz: 2499.996</h2>  
      + </body> ...
```

6-30

Apply the apache cookbook's default recipe to the local system.

'cd' command

Execute on: **workstation**

Execute from: **anywhere**

'chef-client' command

Execute on: **workstation**

Execute from: **/home/chef/chef-repo**

Exercise



GL: Cleaner Recipes

- ✓ Create a template with `chef generate`
- ✓ Define the contents of the ERB template
- ✓ Change the file resource to the template resource in the 'apache' cookbook
- ✓ Use chef-client to apply the "apache" cookbook's "default" recipe
- Update the "apache" cookbook's version for this patch

GL: Update the version number

`cookbooks/apache/metadata.rb`

```
name 'apache'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures apache'
version '0.2.1'
chef_version '>= 16.0'
```

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

GL: Test the web server

```
> curl localhost

<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>PLATFORM: amazon</h2>
    <h2>HOSTNAME: ip-172-31-73-167</h2>
    <h2>MEMORY: 979152kB</h2>
    <h2>CPU Mhz: 2499.996</h2>
  </body>
</html>
```

Test the web server.

Execute on: **workstation**

Execute from: **Anywhere**

Exercise



GL: Cleaner Recipes

- ✓ Create a template with `chef generate`
- ✓ Define the contents of the ERB template
- ✓ Change the file resource to the template resource in the 'apache' cookbook
- ✓ Use chef-client to apply the "apache" cookbook's "default" recipe
- ✓ Update the "apache" cookbook's version for this patch



Concept

Summary

In previous modules we were using Ruby string interpolation to insert the value of a variable.

This works for simple variables but doesn't allow you to insert any logic into what value should be used, or when.

Concept



Summary

In previous modules we were also using a `file` resource in our recipe file. But that isn't scalable past anything more than a few words or lines.

Once you have more than that, it can become unmanageable, especially if there are characters that you need to escape, like `\'`, `\'`, `\"`, `\"` etc. So it's better to have the logic in one file (the recipe) and the data in another (the template).

Review



Review Questions

1. What is the benefit of using a template over defining the content within a recipe?
2. What are the drawbacks of using a template over defining the content within a recipe?
3. What is an ERB template?

Answer these questions.

1. Benefits: abstracting large chunks of config settings to a template gives you much smaller, more manageable recipes, and makes it a lot easier to make and test changes to those recipes.
2. Drawbacks: it's another layer that someone new to Chef or the team has to find and understand.
3. An Embedded Ruby (ERB) template allows Ruby code to be embedded inside a text file within specially formatted tags.



Questions?

Q&A

What questions can we help you answer?

- Resources
- Templates
- ERB

What questions can we help you answer?

Generally or specifically about resources, templates, and ERB.



Progress® Chef™

© 2013 Progress Software Corporation. All rights reserved. All trademarks contained herein are the property of Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

6- 39

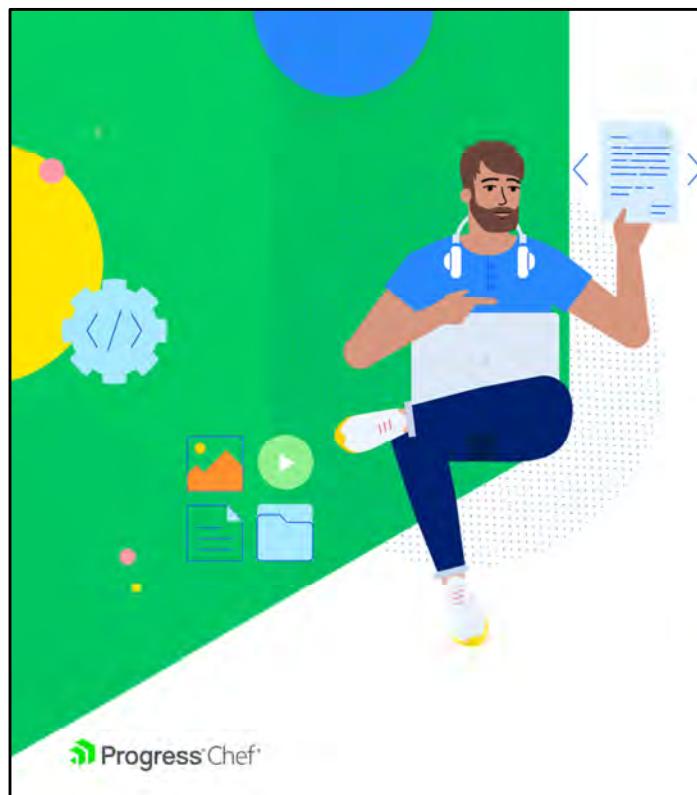


Accessing Chef-Infra Server And Chef Automate

Get to Know the Benefits



8- 1



Objectives

After completing this module, you should be able to:

- Describe the purpose of the Chef Infra Server
- Connect to a Chef Server
- Log in to the Chef Automate UI
- View your Organization in Chef Automate

© 2022 Progress Software Corporation and/or its subsidiaries and/or affiliates. All rights reserved.

8 2

In this module you will learn how to connect your local workstation to Chef Infra Server.

Chef Infra Server is integrated into the Chef Automate UI. Connect your local workstation (laptop) to a Chef Server and then Login to the Chef Automate Web GUI UI which will be provided by the Instructor.

Managing additional systems

To manage another system, you would need to:

1. Log into and provision a new node within your company or appropriate cloud provider with the appropriate access to login to administer the system.
2. Install the Chef tools.
3. Transfer the cookbook(s) to the new node.
4. Run chef-client on the new node to apply the cookbook's recipe(s).

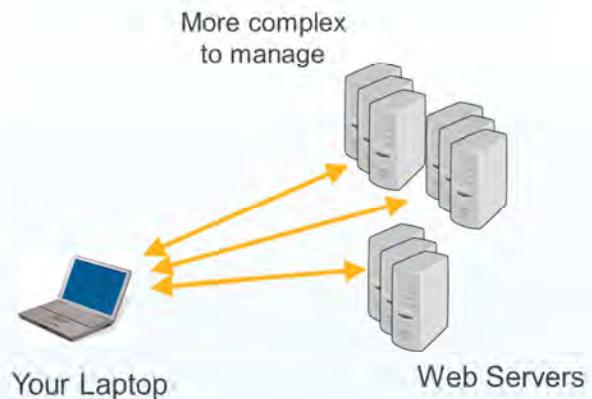
As an exercise, roughly estimate the time it would take to accomplish this series of steps of preparing another node without Chef Infra Server.

Managing additional systems

Now



Future



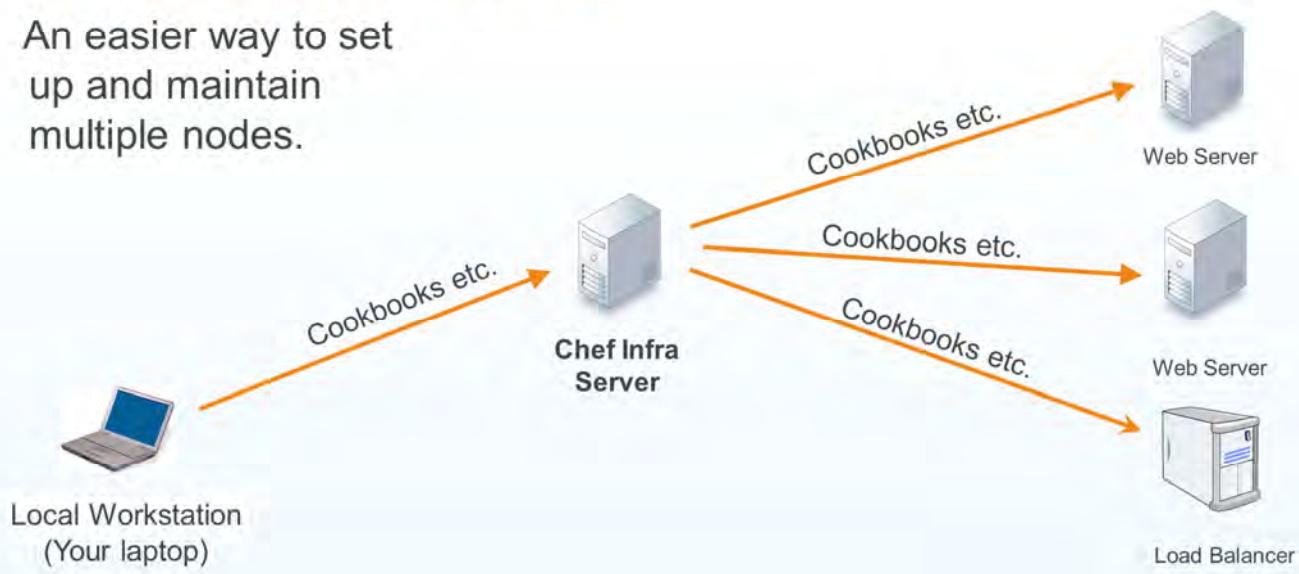
So the overall time required to setup a new instance is not a massive time investment. This manual process will definitely take its toll when requirements demand you manage more than a few additional nodes.

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 up with all of the web requests. You will need to provision additional machines as demand increases.

The Chef Infra Server

An easier way to set up and maintain multiple nodes.



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

The Chef Infra Server is designed to help us manage multiple nodes in this situation. The Chef Infra Server acts as a hub for configuration data. The Chef Infra 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, load balancers, etc., use 'chef-client' to ask the Chef Infra 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 Infra Server). In a production environment, the 'chef-client' runs should be configured in an automated mode—polling the Chef Infra Server for updates at set intervals and then applying any configuration changes (we'll configure this later in the course). This scalable approach distributes the configuration effort throughout the organization.

Concept

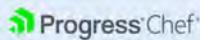


Chef Automate

Chef Automate is an enterprise platform that allows developers, operations and security engineers to collaborate effortlessly on delivering application & infrastructure changes at the speed of business.

A Single Source of Truth for All Environments.

<https://www.chef.io/products/chef-automate>



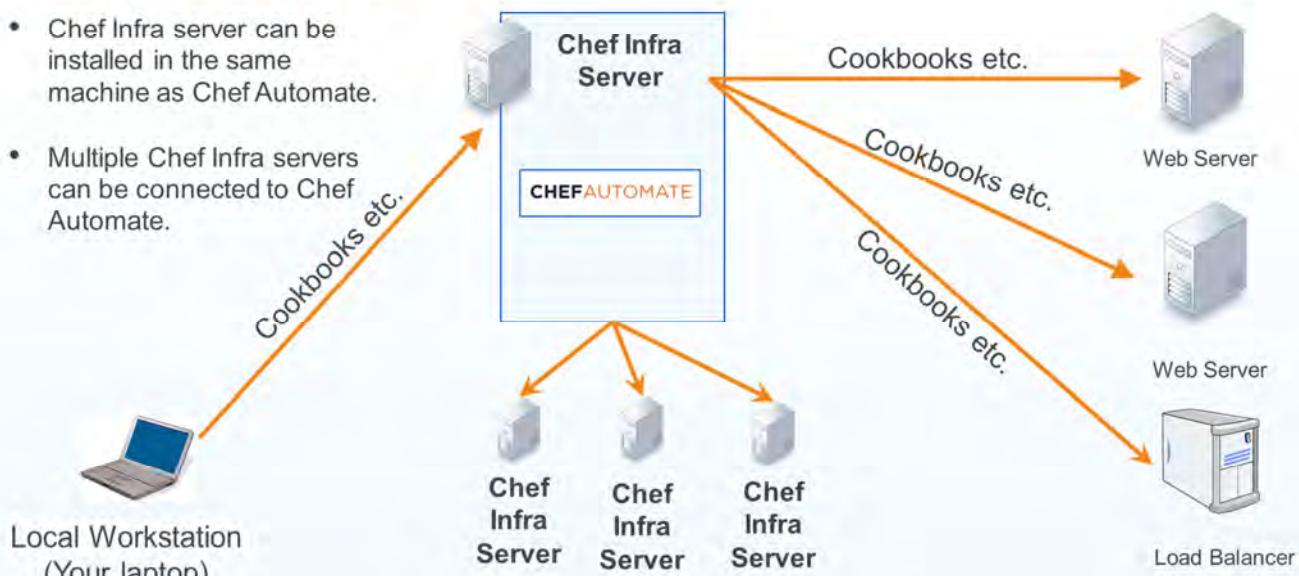
© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

8 6

<https://www.chef.io/products/chef-automate>

Chef Automate

- Chef Infra server can be installed in the same machine as Chef Automate.
- Multiple Chef Infra servers can be connected to Chef Automate.



The Chef Infra Server is designed to help us manage multiple nodes in this situation. It acts as a hub for configuration data. It stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by 'chef-client'.

Chef Automate Can be connected to multiple Chef Infrastructure servers. Chef Infra Server can also be deployed in the same machine where Chef Automate is deployed, <Our course is using chef automate and infra server on same VM > In our next lab we will be connecting your local workstation (your laptop) with a Chef Infra server.

Exercise



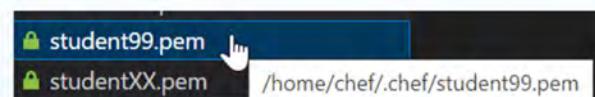
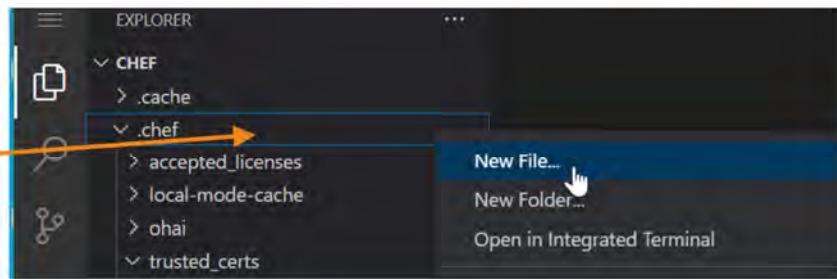
GL: Chef Infra Server and Chef Automate

- Create a client.pem file in the home/chef/.chef directory
- Configure the 'knife' command
- Verify successful communication with Chef Server
- Login to the Chef Automate UI server and Verify Organization details.

Create a .pem file on your assigned workstation

Steps

- In VS Code, right-click the .chef folder, select New File, and name it using your student ID. For example, student99.pem.
- In that new file, paste the contents of the .pem key the instructor gave you.



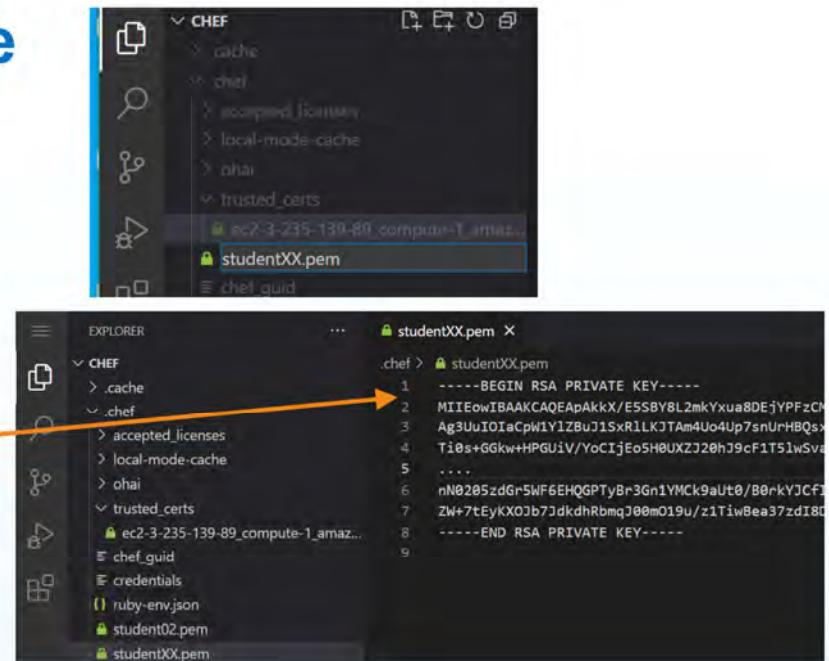
The default configuration directory of chef is placed in the user's home directory with directory name ".chef", for example (~/.chef). [/home/chef/.chef](#) in this course. Chef-client and knife by-default looks for configuration files under the .chef directory. So that's why we will be copying private key contents to the file named "studentXX.pem".

Instructor note: Please share the location of the pem file with all the participants. You can share them anyway you like but one example would be to put the pem files in Onedrive and share a link.

Create a .pem file

Steps

- In VS Code, right-click the .chef folder, select New File, and name it using your student ID. For example, student99.pem.
- In that new file, paste the contents of the .pem key the instructor gave you.



The screenshot shows the VS Code interface with the following details:

- Explorer Panel:** Shows the project structure under the "CHEF" folder. It includes ".cache", ".chef" (which contains "accepted_licenses", "local-mode-cache", and "trusted_certs"), "ohai", "ruby-env.json", "student02.pem", and "studentXX.pem".
- Editor Panel:** Displays the contents of the "studentXX.pem" file. The file starts with a private RSA key header: "-----BEGIN RSA PRIVATE KEY-----". Below this, there is a long string of encoded data: MIIEdwIBAAKCAQEA... (approximately 100 lines of text).

Exercise



GL: Chef Infra Server and Chef Automate

- ✓ *Create a client.pem file in the /home/chef/.chef directory*
- *Configure the 'knife' command*
- *Verify successful communication with Chef Server*
- *Login to the Chef Automate UI server and Verify Organization details.*



Concept

Knife

knife is a command-line tool that provides an interface between a local chef-repo and the Chef Infra Server. knife helps users to manage:

- Nodes
- Cookbooks and recipes
- Policyfiles, and Data Bags
- The installation of Chef Infra Client onto nodes (bootstrapping)
- Searching of indexed data on the Chef Infra Server

<https://docs.chef.io/workstation/knife/>

<https://docs.chef.io/workstation/knife/>

GL: Configure Knife

```
> knife configure init-config
```

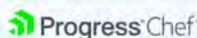
```
WARNING: No knife configuration file found. See https://docs.chef.io/config\_rb/ for details.
```

```
Please enter the chef server URL: [https://ip-172-31-73-167.ec2.internal/organizations/myorg] https://ec2-3-235-139-89.compute-1.amazonaws.com/organizations/studentXX
```

```
Please enter an existing username or clientname for the API: [chef] studentXX  
*****
```

```
You must place your client key in:  
/home/chef/.chef/student01.pem  
Before running commands with Knife
```

Be sure to add
/organizations/studentXX
to the Automate URL.

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.8 / 13

In this slide you will be configuring knife utility on your workstation so that workstation can talk to Chef Infra server.

The knife utility is installed as part of the Chef Workstation installation, which you have done in a previous module.

You will be executing the command “**knife configure init-config**” to configure knife. It will ask for a Chef server URL, which the Instructor will share.

In the URL make sure to replace XX with the number assigned to each participant and append organizations in the URL (such as "student77").

INSTRUCTOR NOTE: Please share the Chef Automate URL with participants.

‘cd’ command

Execute on: **workstation**

Execute from: **anywhere**

‘knife’ command

Execute on: **workstation**

Execute from: **/home/chef/.chef**

GL: Validating Knife configuration

```
> cat credentials
```

```
[default]
client_name      = 'studentXX'
client_key       = '/home/chef/.chef/studentXX.pem'
chef_server_url = 'https://ec2-3-235-139-89.compute-1.amazonaws.com/organizations/studentXX'
```

Exercise



GL: Chef Infra Server and Chef Automate

- ✓ *Create a client.pem file in the ~/.chef directory*
- ✓ *Configure 'knife'*
- *Verify successful communication with Chef Server*
- *Login to the Chef Automate UI server and Verify Organization details.*

GL: Verification of connection

```
> cd /home/chef  
> knife ssl fetch
```

```
WARNING: Certificates from ec2-3-235-139-89.compute-1.amazonaws.com will be  
fetched and placed in your trusted_cert  
directory (/home/chef/.chef/trusted_certs).
```

```
Knife has no means to verify these are the correct certificates. You  
should  
verify the authenticity of these certificates after downloading.  
Adding certificate for ec2-3-235-139-89_compute-1_amazonaws_com in  
/home/chef/.chef/trusted_certs/ec2-3-235-139-89_compute-1_amazonaws_com.crt
```

To fetch the ssl certificates, use the **knife ssl fetch** subcommand to copy SSL certificates from an HTTPS server to the **trusted_certs_dir** directory (.chef directory) that is used by knife and Chef Infra Client to store trusted SSL certificates. When these certificates match the hostname of the remote server, running knife ssl fetch is the only step required to verify a remote server that is accessed by either knife or Chef Infra Client.

Execute on: **workstation**

Execute from: **/home/chef**

GL: Verification of connection

```
> knife ssl check
```

```
Connecting to host ec2-3-235-139-89.compute-1.amazonaws.com:443
Successfully verified certificates from `ec2-3-235-139-89.compute-1.amazonaws.com'
```

GL: Verification of connection

```
> knife client list
```

```
studentXX-validator
```

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.8 / 18

This shows that we are successfully connected to our Chef Infra Server. It will show us the client list which is available in your organization in the Chef Infra Server.

You will see your client (your workstation) listed as 'studentXX-validator'.

Execute on: **workstation**

Execute from: **/home/chef**

Exercise



GL: Chef Infra Server and Chef Automate

- ✓ Create a client.pem file in the ~/.chef directory
- ✓ Configure 'knife'
- ✓ Verify successful communication with Chef Server
- Login to the Chef Automate UI server and Verify Organization details.

Log in and Access Chef Automate

Steps

- Sign in to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Go to Clients tab to view client list



Note: Login to Chef Automate UI using the username and password shared by the Instructor.

Instructor note :

Please share below credentials with participants:

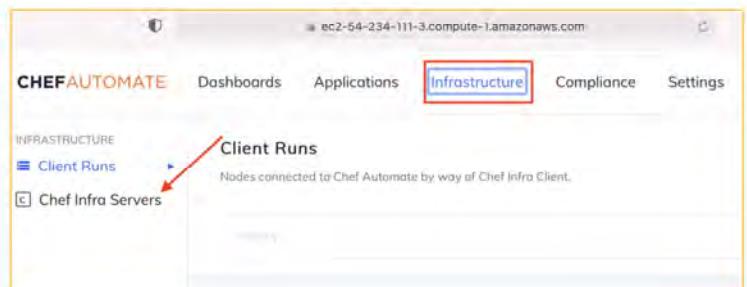
Username: studentXXuser-ca

Password: studentXX

Log in and Access Chef Automate

Steps

- Sign in to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Go to Clients tab to view client list



Log in and Access Chef Automate

Steps

- Sign in to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- **Click on Listed Chef Infra Server - 'cheftraining'**
- Click on the organization - 'studentxx'
- Go to Clients tab to view client list

Name	FQDN	IP Address	Number Of Orgs
cheftraining	ec2-54-234-111-3.compute-1.amazonaws.com	54.234.111.3	30

Log in and Access Chef Automate

Steps

- Sign in to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- **Click on the organization - 'studentxx'**
- Go to Clients tab to view client list

The screenshot shows the 'Chef Infra Servers' interface with the URL 'Chef Infra Servers > cheftraining'. It displays the 'cheftraining' organization with FQDN 'ec2-54-234-111-3.compute-1.amazonaws.com' and IP Address '54.234.111.3'. The 'Orgs' tab is selected. In the 'Details' section, there is a table with three columns: 'Name' (containing 'student01'), 'Admin' (containing 'student01'), and 'Projects' (containing 'student01project'). A red arrow points to the 'student01' entry in the 'Name' column.

Note: Login to Chef Automate UI using the username and password shared by the Instructor.

Log in and Access Chef Automate

Steps

- Sign in to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- **Go to Clients tab to view client list**

The screenshot shows the Chef Infra Automate interface. The URL is 'Chef Infra Server > Organizations > student01'. The navigation bar includes 'Cookbooks', 'Roles', 'Environments', 'Data Bags', 'Clients' (which is highlighted with a red box), 'Nodes', and 'Policyfiles'. Below the navigation bar, there's a search bar and a 'Create Client' button. The main area is titled 'Select Client' and contains a table with one row. The table has columns for 'Name' and 'Actions'. The 'Name' column contains the value 'student01-validator'. A red arrow points to this value. The 'Actions' column contains a small icon.

To check that you have configured knife with the correct organization on your workstation machine and logged in with the correct credentials, click on the 'Clients' tab. You should see the same client validator that was visible when you executed the 'knife client list' command.

NOTE: If you see variations in the student validator key from the cli and the web GUI, please notify your Instructor.

Exercise



GL: Chef Infra Server and Chef Automate

- ✓ Create a client.pem file in the ~/.chef directory
- ✓ Create a directory named "chef-repo" on your workstation
- ✓ Configure 'knife'
- ✓ Verify successful communication with Chef Server
- ✓ Login to the Chef Automate UI server and Verify Organization details.



Questions?

Q&A

What questions can we answer for you?





Bootstrap a node

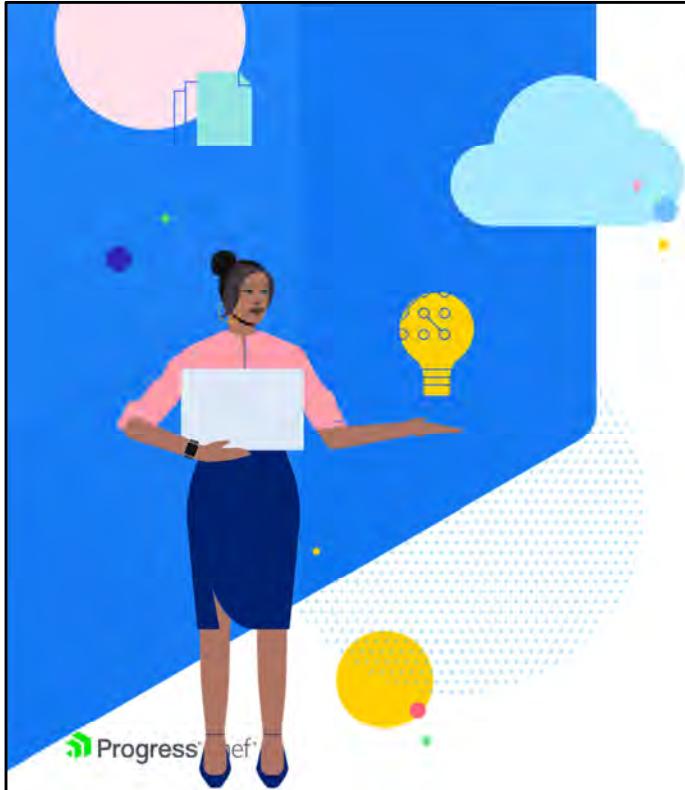
Tell Chef which nodes to manage



1W and 1L

Instructor Note: The 1W means each student will need their `iis_web` node node to bootstrap and install IIS web Server on.

We'll also be bootstrapping the day 2 Linux `apache_web` node.



Objectives

After completing this module, you should be able to

- Clone cookbook from a GitHub repository
- Bootstrap two nodes
- Verify the nodes were bootstrapped in Chef Automate

© 2022 Progress Software Corporation and/or its affiliated companies. All rights reserved.

8 2

In this module you will learn about the purpose of the Chef Infra Server, clone cookbooks from a GitHub repository, and connect your local workstation (laptop) to a Chef Infra Server. You will also bootstrap 2 nodes that will eventually become web servers.



Concept

Bootstrapping

A node is any physical, virtual, or cloud device that is configured and maintained by an instance of Chef Infra Client.

Bootstrapping installs Chef Infra Client on a target system so that it can run as a client and sets the node up to communicate with a Chef Infra Server.

https://docs.chef.io/install_bootstrap/

A node is any physical, virtual, or cloud device that is configured and maintained by an instance of Chef Infra Client.

Bootstrapping installs Chef Infra Client on a target system so that it can run as a client and sets the node up to communicate with a Chef Infra Server.

https://docs.chef.io/install_bootstrap/

Concept



Bootstrapping

Often, the node you are bootstrapping may not have Chef installed. It may also not have details of where the Chef Infra Server is located or the credentials to securely talk to that Server.

To add those credentials we can **bootstrap** that node to install all those components.

We want to add a new instance as a node within our organization. Often times, the node you are bootstrapping may not have Chef installed on it. It also probably does not know where the Chef Infra Server is or have the credentials to even talk to it securely. We could manually configure a node but there is an easier way of doing that through a process called 'bootstrapping'.

Bootstrapping will install Chef if necessary and then configure the node to talk securely to a specified Chef Infra Server.

Exercise



GL: Download Cookbooks and Bootstrap Nodes

- *Download and copy the required cookbooks to your assigned workstation*
- *Confirm that you can connect to the Chef Infra Server*
- *Bootstrap two nodes*

We'll download a copy of an IIS cookbook and test our connection to Chef Infra Server

We are going to need a copy of the myiis cookbook and eventually get this up to the Chef Infra Server.



Concept

GL: Code Repository

This GitHub repository contains copies of cookbooks to be used in this course including the 'apache' cookbook you created.

<https://github.com/chef-training/foundations-cookbook-repo>

<https://github.com/chef-training/foundations-cookbook-repo>

The 'apache' cookbook that were created in a previous and an and 'myiis' cookbook can be found here. These may not be exact copies of cookbooks that you created but will function for the purpose of the class.

GL: Use your terminal, move to /home/chef, and ls

```
> cd /home/chef  
> ls
```

```
chef-repo  goodbye.rb  hello.rb  nodes
```

Use your VS Code's terminal and navigate to the /home/chef directory.

Execute on: **workstation**

Execute from: **anywhere**

GL: Clone the repo

```
> git clone https://github.com/chef-training/foundations-cookbook-repo.git

Cloning into 'foundations-cookbook-repo'...
remote: Enumerating objects: 110, done.
remote: Counting objects: 100% (44/44), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 110 (delta 8), reused 28 (delta 4), pack-reused 66
Receiving objects: 100% (110/110), 15.62 KiB | 3.90 MiB/s, done.
Resolving deltas: 100% (21/21), done.
```

Use your VS Code's terminal and navigate to the `/home/chef` directory.

The run `git clone https://github.com/chef-training/foundations-cookbook-repo.git``

Execute on: **workstation**

Execute from: **/home/chef**

GL: Clone the repo

```
> cd foundations-cookbook-repo  
> ls
```

```
apache myiis README.md
```

Use your VS Code's terminal and navigate to the /home/chef directory.

Execute on: **workstation**

Execute from: **/home/chef**

GL: Move the myiis cookbook to the cookbooks dir

```
> mv myiis ~/chef-repo/cookbooks/  
> ls
```

```
README.md
```

Use your VS Code's terminal and navigate to the /home/chef directory.

Execute on: **workstation**

Execute from: **/home/chef/foundations-cookbook-repo**

GL: Clone the code repository

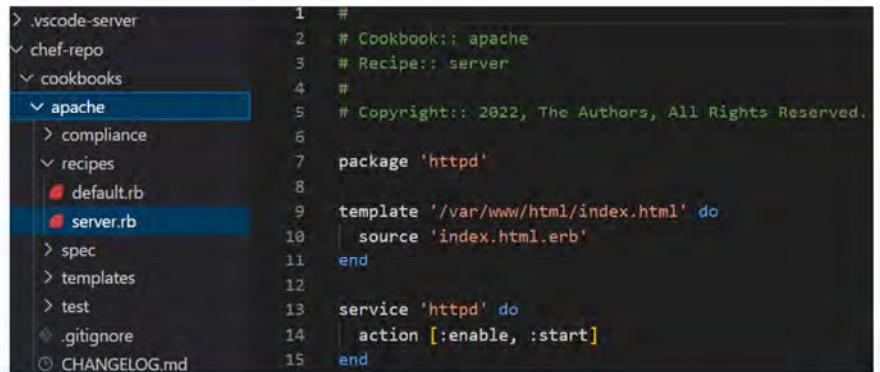
Use VS Code to ensure the cookbook was moved into the `/home/chef/chef-repo/cookbooks/` directory.

We will upload these cookbooks to Chef Infra Server later in this course via a Policyfile.

```
chef-repo > cookbooks > apache >
1  #
2  # Cookbook:: apache
3  # Recipe:: server
4  #
5  # Copyright:: 2022,
6
7  package 'httpd'
8
9  template '/var/www/h
10    source 'index.html
11
12  end
13
14  service 'httpd' do
15    action [:enable, ::
```

GL: Review the apache cookbook

If you view the **recipes** and the templates of this cookbook you should recognize it from a previous module.



The terminal shows the directory structure of a Chef cookbook named 'apache'. The 'recipes' directory contains two files: 'default.rb' and 'server.rb'. The 'server.rb' file is selected and shown in the code editor. The code defines a package for httpd, a template for index.html, and a service for httpd with actions to enable and start it.

```
1  #
2  # Cookbook:: apache
3  # Recipe:: server
4  #
5  # Copyright:: 2022, The Authors, All Rights Reserved.
6
7  package 'httpd'
8
9  template '/var/www/html/index.html' do
10    source 'index.html.erb'
11  end
12
13  service 'httpd' do
14    action [:enable, :start]
15  end
```

GL: Review the apache cookbook

If you view the recipes and the **templates** of this cookbook you should recognize it from a previous module.

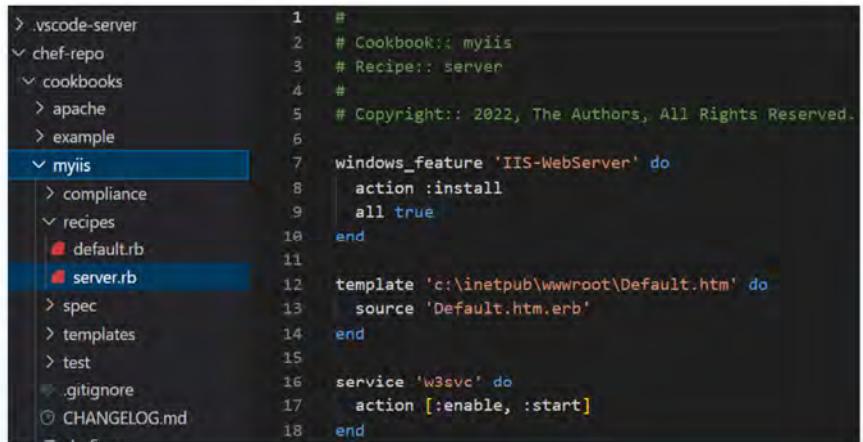


```
1 <html>
2   <body>
3     <h1>Hello, world!</h1>
4     <h2>PLATFORM: <%= node['platform'] %></h2>
5     <h2>HOSTNAME: <%= node['hostname'] %></h2>
6     <h2>MEMORY:  <%= node['memory'][('total') %]></h2>
7     <h2>CPU Mhz: <%= node['cpu'][('0')]['mhz'] %></h2>
8   </body>
9 </html>
```

GL: View the myiis cookbook

If you view the **recipes** and the templates of the myiis cookbook, you'll see the differences the myiis for Windows cookbook has compared to the previous apache cookbook.

This cookbook sets up an IIS web server for Windows servers where the apache cookbook sets up a web server for Linux servers.



The terminal window shows the directory structure of a Chef repository:

```
> .vscode-server
  < chef-repo
    < cookbooks
      > apache
      > example
      < myiis
        > compliance
        < recipes
          < default.rb
          < server.rb
        > spec
        > templates
        > test
        < .gitignore
        < CHANGELOG.md
```

Below the directory tree, the contents of the `server.rb` recipe are displayed:

```
1  #
2  # Cookbook:: myiis
3  # Recipe:: server
4  #
5  # Copyright:: 2022, The Authors, All Rights Reserved.
6
7  windows_feature 'IIS-WebServer' do
8    action :install
9    all_true
10   end
11
12 template 'c:\inetpub\wwwroot\Default.htm' do
13   source 'Default.htm.erb'
14   end
15
16 service 'w3svc' do
17   action [:enable, :start]
18 end
```

GL: View the myiis cookbook

Notice that the **file paths** and the **service name** in the myiis cookbook reflect usage on Windows servers.

The terminal shows the directory structure of a Chef repository:

```
> .vscode-server
\ chef-repo
  \ cookbooks
    > apache
    > example
    < myiis
      > compliance
      > recipes
        & default.rb
        & server.rb
      > spec
      > templates
      > test
    .gitignore
    CHANGELOG.md
```

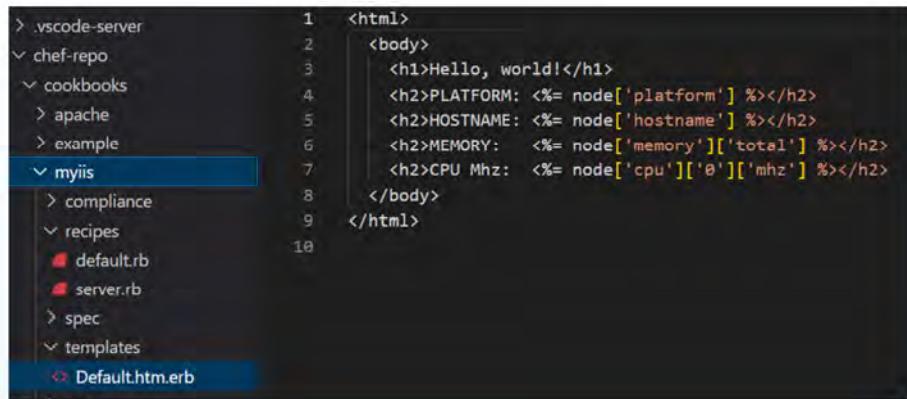
Below is the content of the `default.rb` recipe:

```
1  #
2  # Cookbook:: myiis
3  # Recipe:: server
4  #
5  # Copyright:: 2022, The Authors, All Rights Reserved.
6
7  windows_feature 'IIS-WebServer' do
8    action :install
9    all true
10   end
11
12  template 'c:\inetpub\wwwroot\Default.htm' do
13    source 'Default.htm.erb'
14  end
15
16  service 'w3svc' do
17    action [:enable, :start]
18  end
```

Two orange arrows point from the text "file paths" and "service name" in the notice above to the file path "c:\inetpub\wwwroot\Default.htm" and the service name "w3svc" respectively in the code.

GL: View the myiis cookbook

If you view the **template** of this cookbook you'll see that it's identical to that used in the apache cookbook.



```
1 <html>
2   <body>
3     <h1>Hello, world!</h1>
4     <h2>PLATFORM: <%= node['platform'] %></h2>
5     <h2>HOSTNAME: <%= node['hostname'] %></h2>
6     <h2>MEMORY: <%= node['memory']['total'] %></h2>
7     <h2>CPU Mhz: <%= node['cpu'][0]['mhz'] %></h2>
8   </body>
9 </html>
```



Concept

knife

As mentioned previously, knife is a command-line tool that provides an interface between a local chef-repo and the Chef Infra Server.

In this section we'll use 'knife node' commands.

knife is a command-line tool that allows us to request and send information to the Chef Infra Server.

knife helps users manage nodes, cookbooks, roles, environments, and more. knife does this through a series of sub-commands.

GL: 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)
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Run 'knife node list'

```
$ knife node list
```



We will use knife for a number of tasks in a moment.

Run "knife node list" to see that you have no nodes currently registered with your Chef Infra Server. At this point the results should be blank but this shows that we can connect to our Chef Infra Server.

Execute on: **workstation**

Execute from: **anywhere**

Exercise



GL: Download Cookbooks and Bootstrap Nodes

- ✓ Download and copy the required cookbook to your assigned workstation
- ✓ Confirm that you can connect to the Chef Infra Server
- Bootstrap two nodes

GL: Bootstrap your nodes

In this lab you will use a new Windows instance and bootstrap it as a managed node. You will do the same to Day 1 Linux node as well.

You'll need the FQDN or Public IP of those instances to perform this lab.

At this point we will be placing a Windows machine under management to act as our IIS Web server.

GL: Run 'knife bootstrap –help'

```
$ knife bootstrap --help

knife bootstrap FQDN (options)
  --bootstrap-curl-options OPTIONS
                                Add options to curl when install chef-client
  --bootstrap-install-command COMMANDS
                                Custom command to install chef-client
  --bootstrap-no-proxy [NO_PROXY_URL|NO_PROXY_IP]
                                Do not proxy locations for the node being
bootstrapped
  --bootstrap-proxy PROXY_URL  The proxy server for the node being bootstrapped
  -t TEMPLATE,                Bootstrap Chef using a built-in or custom template.
Set to the full path of an erb template or use one of the built-in templates.
```

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 Infra Server; Running chef-client to apply a default run list.

Execute on: **workstation**

Execute from: **anywhere**

GL: Bootstrap your Windows node

```
> knife bootstrap -o winrm IPADDRESS -U Administrator -P PSWD -N iis_web

Connecting to 107.20.24.200 using winrm
Creating new client for iis_web
Creating new node for iis_web
Bootstrap...
[107.20.24.200] Fully Qualified Domain Name or IP
[107.20.24.200] [107.20.24.200] directory "C:\chef"...
[107.20.24.200] [107.20.24.200] d, s, n, r, a, m, h, v, p, t, u, o, f, g, l, w, x, y, z, ., ..
[107.20.24.200] [107.20.24.200] echo.url = WScript.Arguments.Named("url")
[107.20.24.200] [107.20.24.200] ...
[107.20.24.200] [107.20.24.200] c:\Users\Administrator\Documents>chef-client
boot.json
[107.20.24.200] +-----+
[107.20.24.200] +-----+ 2 product licenses accepted.
[107.20.24.200] +-----+
[107.20.24.200] Chef Infra Client, version 17.10.3
run list.
[107.20.24.200] Running handlers:
[107.20.24.200] Running handlers complete
[107.20.24.200] Infra Phase complete, 0/0 resources updated in 33 seconds
```

The licenses were accepted because we ran this command from our laptops, which already have accepted the licenses.

To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-U' flag and the password with the '-P' flag. Name the node with the '-N' flag. This is optional but makes it easier for us to communicate with the node. When we ask you to look at the details of iis_web or login to iis_web, 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.

Execute on: **workstation**

Execute from: **anywhere**

GL: Run 'knife node list'

```
$ knife node list
```

```
iis_web
```

GL: View more information about your node

```
$ knife node show iis_web
```

```
Node Name: iis_web
Environment: _default
FQDN: WIN-DQFQCUFHDCP
IP: 107.20.24.200
Run List:
Roles:
Recipes:
Platform: windows 6.3.9600
Tags:
```

You can see more information about a particular node with the command 'knife node show *nodename*'. This will display a quick summary of the node information that the Chef Infra Server stores.

Execute on: **workstation**

Execute from: **anywhere**

Chef Automate: iis_web

- [Sign In to Chef Automate](#)
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Click on Nodes tab to view iis_web



Note: Login to Chef Automate UI using username and password shared by the Instructor.

Instructor note :

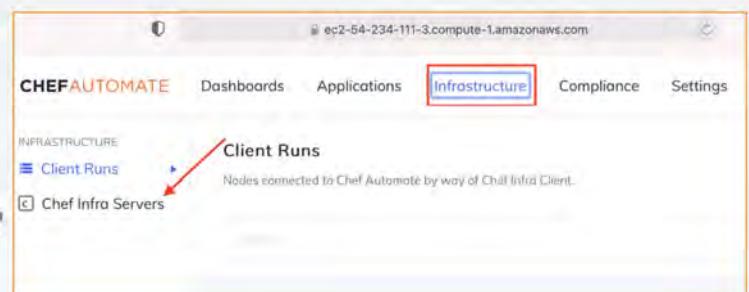
Please share below credentials with participants:

Username: studentXXuser-ca

Password: studentXX

Chef Automate: iis_web

- Sign In to Chef Automate
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Click on Nodes tab to view iis_web



Chef Automate: iis_web

- Sign In to Chef Automate
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- **Click on Listed Chef Infra Server - 'cheftraining'**
- Click on the organization - 'studentxx'
- Click on Nodes tab to view iis_web

Name	FQDN	IP Address	Number Of Orgs
cheftraining	ec2-54-234-111-3.compute-1.amazonaws.com	54.234.111.3	30

Chef Automate: iis_web

- Sign In to Chef Automate
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- **Click on the organization - 'studentxx'**
- Click on Nodes tab to view iis_web

The screenshot shows the 'Chef Infra Servers' interface with the URL 'Chef Infra Servers > cheftraining'. The server details are as follows:

FQDN	IP Address
ec2-54-234-111-3.compute-1.amazonaws.com	54.234.111.3

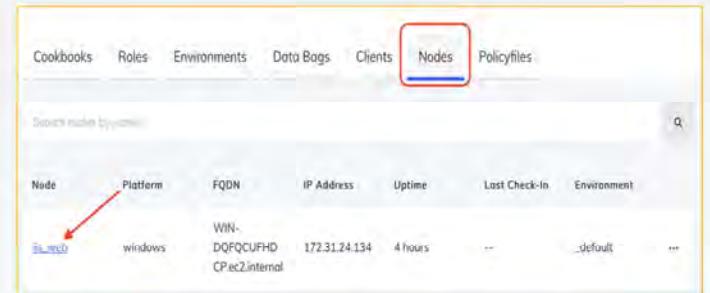
The 'Orgs' tab is selected. The 'Details' section shows:

Name	Admin	Projects
student01	student01	student01project

A red arrow points to the 'student01' node under the 'Name' column.

Chef Automate: iis_web

- Sign In to Chef Automate
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- **Click on Nodes tab to view iis_web**



Node	Platform	FQDN	IP Address	Uptime	Last Check-In	Environment	...
iis_web	windows	WIN-DQFQCUFHDCPec2.internal	172.31.24.134	4 hours	--	_default	...

GL: Bootstrap your day 2 Linux apache_web node

```
> knife bootstrap IP -U chef -P PSWD --sudo -N apache_web
Connecting to 44.205.9.57 using ssh
The authenticity of host '44.205.9.57 ()' can't be established.
fingerprint is SHA256:07AVSt3Ai6fNFGt8u/wnUGOMP1Mzo7QrG8KwLSUmI.
Are you sure you want to continue? [Y/N] Y
...
user name
password
node name

Connecting to 44.205.9.57 using ssh
Creating new client for apache_web
Creating new node for apache_web
Bootstrapping 44.205.9.57
...
[44.205.9.57] Converging 0 resources
[44.205.9.57]
Running handlers:
[44.205.9.57] Running handlers complete
[44.205.9.57] Chef Infra Client finished, 0/0 resources updated in 04 seconds
```

To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-U' flag and the password with the '-P' flag. Name the node with the '-N' flag. This is optional but makes it easier for us to communicate with the node.

When executing the command, the output will tell us what it installed and ran.

Execute on: **workstation**

Execute from: **anywhere**

GL: Run 'knife node list'

```
$ knife node list
```

```
apache_web  
iis_web
```

GL: View more information about your node

```
$ knife node show apache_web
```

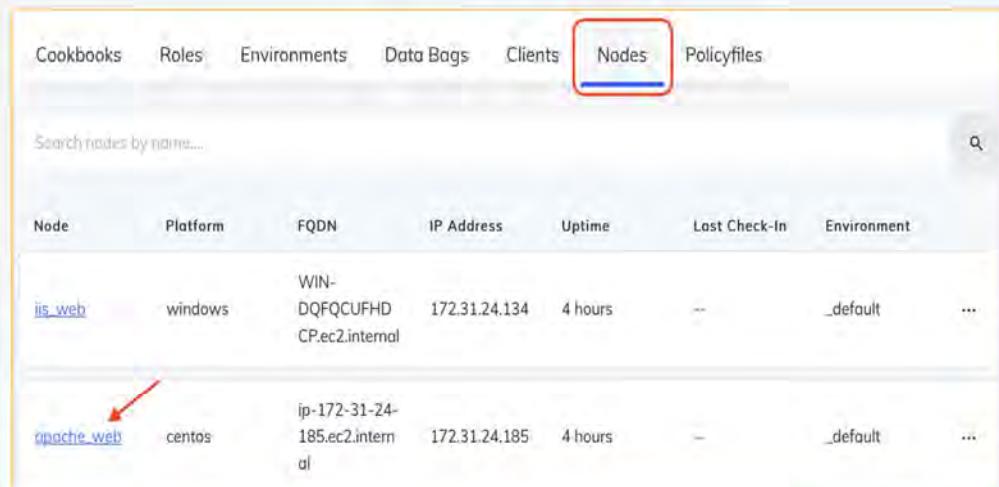
```
Node Name: apache_web
Environment: _default
FQDN: ip-172-31-77-52.ec2.internal
IP: 44.205.9.57
Run List:
Roles:
Recipes:
Platform: centos 7.6.1810
Tags:
```

You can see more information about a particular node with the command 'knife node show *nodename*'. This will display a summary of the node information that the Chef Infra Server stores.

Execute on: **workstation**

Execute from: **anywhere**

Verify the 'apache_web' node creation



Node	Platform	FQDN	IP Address	Uptime	Last Check-In	Environment	...
iis_web	windows	WIN-DQFQCUFHD.ec2.internal	172.31.24.134	4 hours	--	_default	...
apache_web	centos	ip-172-31-24-185.ec2.internal	172.31.24.185	4 hours	--	_default	...

Exercise



GL: Download Cookbooks and Bootstrap Nodes

- ✓ *Download and copy the required cookbook to your assigned workstation*
- ✓ *Confirm that you can connect to the Chef Infra Server*
- ✓ *Bootstrap two nodes*



Review

Review Questions

1. What is the primary tool for communicating with the Chef Infra Server?
2. How did you add a node to your organization?

1. knife
2. When we bootstrapped the node, its organization was set based on the pem and URL in our credentials file.

For example:

~/.chef/credentials:

```
[default]
client_name  = 'studentxx'
client_key    = 'studentxx.pem'
chef_server_url = 'https://ec2-34-228-10-12.compute-1.amazonaws.com/organizations/studentxx'
```



Questions?

Q&A

What questions can you help you answer?

- Chef Infra Server
- knife
- Bootstrapping Nodes

With all of the objectives complete, you are finished with this section. What questions can I answer for you?

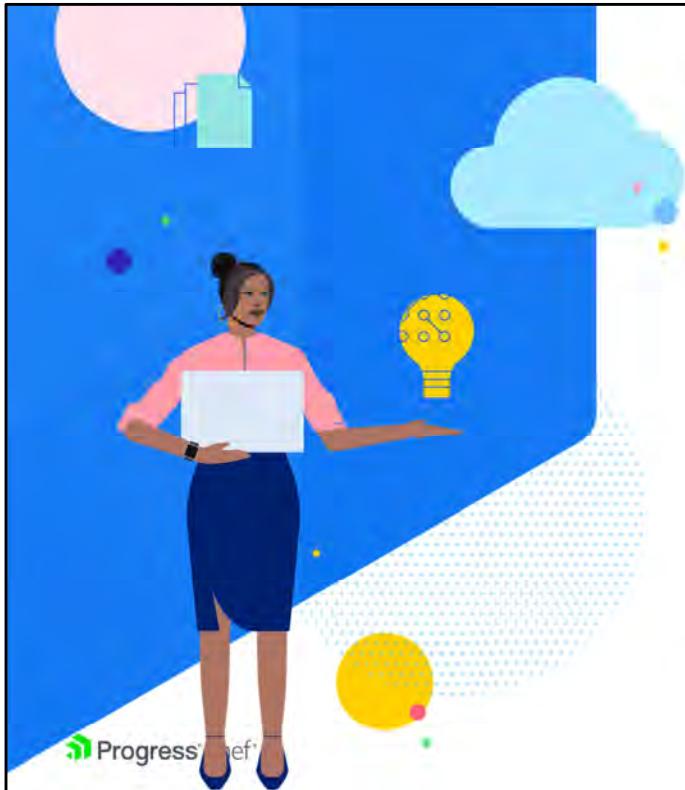




Policyfiles

Combining the functions of Berkshelf,
Environments, and Roles into a single artifact





Objectives

After completing this module, you should be able to

- Explain what Policyfiles are used for
- Use Policyfiles to set run lists for your nodes
- Describe how Policyfiles replace the legacy Roles, Environments, and Berkshelf
- Create a policyfile and a policyfile.lock.json
- Push the policyfile.lock.json to Chef Infra Server and converge a node

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

9 2

In this module you will learn how to use Policyfiles. Later in this course you will use Policyfiles when you manage a number of nodes.

Concept



Policyfiles

A Policyfile (aka Policyfile.rb) is a file that contains information about a node's:

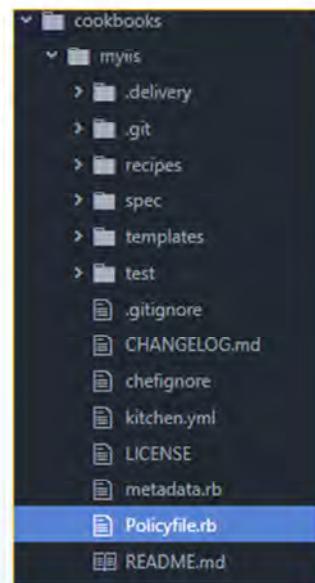
- Policy_name
- Default source for fetching cookbooks
- Run list (or multiple run lists)
- Custom cookbook paths
- Optional cookbook attributes

Policyfiles

When you generate a Chef cookbook using a modern version of Chef Workstation, a Policyfile.rb file is automatically created. (* see speaker notes below slide)

Notice there is no longer a Berksfile generated like in older versions of Chef Workstation/ChefDK.

A Policyfile is the best way to manage run lists, and community cookbook data with a single document that is uploaded to the Chef Infra Server.



*Older versions of ChefDK also support Policyfiles but the creation of the Policyfile was not automated by default. You would need to specify a -P option.
For example, `chef generate cookbook COOKBOOKNAME -P`

When Policyfiles are autogenerated, they are named Policyfile.rb, which Test Kitchen automatically recognizes. This is fine if the Policyfile is living within the cookbook. The name can be changed to a more meaningful name if it is outside of the cookbook. If this is done, it needs to be pointed out in test-kitchen and when installing and pushing. i.e., `chef install POLICYNAME` when the name is different. We will do this in future lab exercises.

Policyfile

This is an example of the Policyfile that was auto generated when the user ran `chef generate cookbook myiis`.

name: Used as not just a name for this policyfile, but it replaces the old **role** object. Use a name that reflects the purpose of the machines where the policy will run.

The name must be unique.

Nodes using this policyfile will possess the **myiis** role.

```
Policyfile.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with
7 # name 'myiis'
8
9 # Where to find external cookbooks:
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order
13 # run_list 'myiis::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'myiis', path: '..'
```

name lets you roll this policy out across hundreds or even thousands of nodes.

As you will learn later in this course, applying the concept of a role to a number of nodes greatly simplifies the configuration of those nodes. As a best practice, legacy role objects are replaced with the Policyfile **name** value.

Policyfile

default_source: Specifies where we get cookbooks if they're not specifically declared in the cookbook section below.

This will usually be the public or a private supermarket or Chef Infra Server.

default_source can also be used for an internal repository where all your cookbooks reside.

```
Policyfile.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with
7 # name 'myiis'
8
9 # Where to find external cookbooks:
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order
13 run_list 'myiis::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'myiis', path: ','
```

Policyfile

run_list: This sets the run list for any nodes using this Policyfile.

cookbook `COOKBOOK`, path: declares the non-default location where cookbooks can be found.

This may be a path to the top-level of a cookbook repository or to the ./cookbooks directory within that repository.

```
Policyfile.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with
7 # name 'myiis'
8
9 # Where to find external cookbooks:
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order
13 # specified
14 run_list 'myiis::default'
15
16 # Specify a custom source for a single cookbook:
17 cookbook 'myiis', path: ','
```

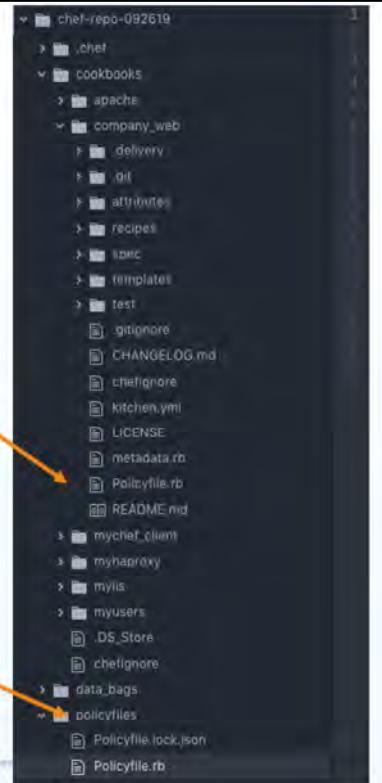
Policyfile Location

A node can have only one policy, so its policyfile will likely have a run list containing multiple cookbooks. Therefore, it wouldn't make sense for the Policyfile to be in a particular cookbook like this.

So we can use the auto-generated Policyfile as a guideline for creating our own Policyfile or simply ignore it.

We recommend you create a **policyfiles** directory at the same level as the **cookbooks** directory if it is not already present there. The latest 'chef generate repo' command does just that.

Then you can generate all your policyfiles within the **policyfiles** directory.



In theory you could generate your Policyfile anywhere you like as long as you provide the correct path to the cookbooks.

Cookbook Location

Assuming you place your policyfiles in the recommended policyfiles directory, you would need to specify the location of its cookbooks like in this example, where

'..../cookbooks/company_web' in this policyfile is saying, go up one level and then down into the cookbooks directory to find the cookbook(s).



```
company_web.json          company_web.rb
1 # Policyfile.json - describe how you want Chef Infra Client to MOLD your system
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # name that describes what the system you're building with Chef looks like
7 name 'company_web'
8
9 # where to find external cookbooks
10 default_source :supermarket
11
12 # run_list - the client will run these recipes in the order specified
13 run_list 'mychef_client::default','company_web::default','myusers::default'
14
15 # Specify a custom source for a single cookbook
16 cookbook 'company_web', path: '../cookbooks/company_web'
17 cookbook 'myiis', path: '../cookbooks/myiis'
18 cookbook 'apache', path: '../cookbooks/apache'
19 cookbook 'mychef_client', path: '../cookbooks/mychef_client'
20 cookbook 'myusers', path: '../cookbooks/myusers'
```

Cookbook Location

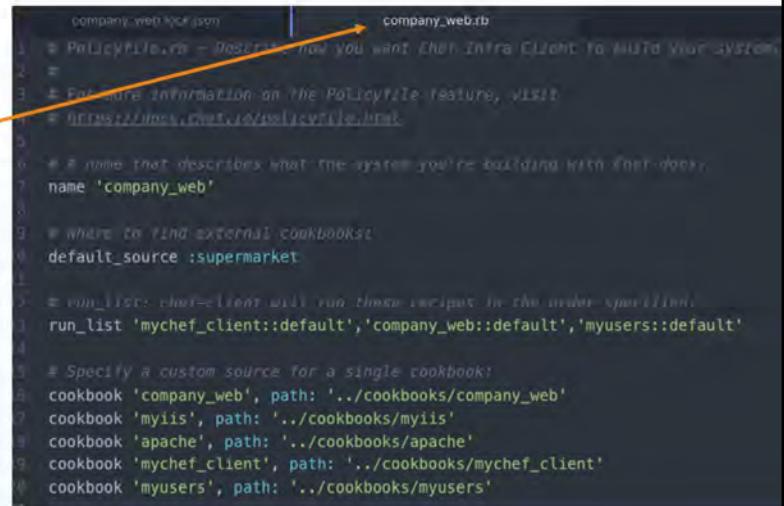
In this way, when you eventually upload your Policyfile (actually a Policyfile.lock.json) to Chef Infra Server, the required cookbooks will also be uploaded simultaneously.

```
company_web.lock.json | company_web.rb
1 # Policyfile.lock.json - describe how you want Chef Infra Client to MOLD your system
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with Chef looks like
7 name 'company_web'
8
9 # Where to find external cookbooks
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order specified
13 run_list 'mychef_client::default','company_web::default','myusers::default'
14
15 # Specify a custom source for a single cookbook
16 cookbook 'company_web', path: '../cookbooks/company_web'
17 cookbook 'myiis', path: '../cookbooks/myiis'
18 cookbook 'apache', path: '../cookbooks/apache'
19 cookbook 'mychef_client', path: '../cookbooks/mychef_client'
20 cookbook 'myusers', path: '../cookbooks/myusers'
```

Policyfile Naming

Also in this example is the correct way to name your policyfiles.

In this example we named the policyfile **company_web.rb** so we can differentiate it from other policyfiles that will reside in the **policyfiles** directory.



```
company_web.lock.json          company_web.rb
1 # Policyfile.rb - Describes how you want Chef Infra Client to MIND your system
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with Chef does.
7 name 'company_web'
8
9 # Where to find external cookbooks
10 default_source :supermarket
11
12 # run_list: chef-client will run these cookbooks in the order specified.
13 run_list 'mychef_client::default','company_web::default','myusers::default'
14
15 # Specify a custom source for a single cookbook
16 cookbook 'company_web', path: '../cookbooks/company_web'
17 cookbook 'myiis', path: '../cookbooks/myiis'
18 cookbook 'apache', path: '../cookbooks/apache'
19 cookbook 'mychef_client', path: '../cookbooks/mychef_client'
20 cookbook 'myusers', path: '../cookbooks/myusers'
```



Concept

Policyfile.lock.json

Before you upload your Policyfile to Chef Infra Server, you actually need to generate Policyfile.lock.json based on the Policyfile.rb.

In other words, we never upload the Policyfile.rb file. We only upload Policyfile.lock.json, which in turn enables the uploading of any required cookbooks.



Concept

Policyfile.lock.json

Policyfile.lock.json also identifies the checksum of each cookbook in the run_list, then creates a master checksum (the revision ID) adding up the checksum of all the cookbooks.

This revision ID is the unique identifier for this group of cookbooks and their included files. If you and I have the EXACT same cookbooks (identical down to each individual character), we'll have the same revision id.

Example: Policyfile.lock

To generate the Policyfile.lock.json if the policyfile name is **company_web**:

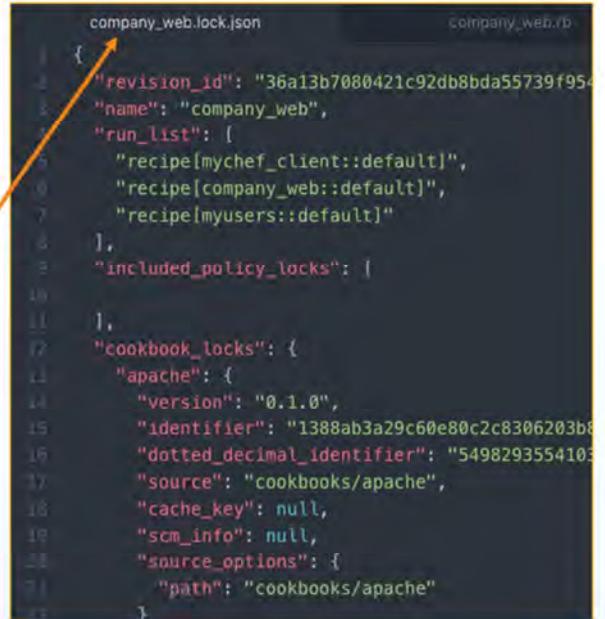
```
chef install company_web.rb
```

That will take the contents of your **company_web.rb** policyfile and convert it to the lock file.

Then you upload the **company_web.lock.json** to the Chef Infra Server with:

```
chef push policy_group company_web
```

Note: We will cover the **policy_group** in a moment.



```
company_web.lock.json          company_web.rb
{
  "revision_id": "36a13b7080421c92db8bda55739f954",
  "name": "company_web",
  "run_list": [
    "recipe[mychef_client::default]",
    "recipe[company_web::default]",
    "recipe[myusers::default]"
  ],
  "included_policy_locks": [
  ],
  "cookbook_locks": {
    "apache": {
      "version": "0.1.0",
      "identifier": "1388ab3a29c60e80c2c8306203b8",
      "dotted_decimal_identifier": "5498293554103",
      "source": "cookbooks/apache",
      "cache_key": null,
      "scm_info": null,
      "source_options": {
        "path": "cookbooks/apache"
      }
    }
  }
}
```

Policyfile.lock

When you generate the Policyfile.lock.json file, a **revision_id** is generated in the form of a hash.

That hash is the version number with which you can identify versions of this Policyfile.

```
Policyfile.lock.json
1 {
  "revision_id": "bd6c581e1a16a3e317f043dd24f4b4f55f08352e8df83a9f5290ae0ae4",
  "name": "myiis",
  "run_list": [
    "recipe[myiis::default]"
  ],
  "included_policy_locks": [],
  "cookbook_locks": {
    "myiis": {
      "version": "0.2.1",
      "identifier": "17ddb9d2c05e62af009d39b21f041e2d01cf7b",
      "dotted_decimal_identifier": "6717730919810534.12885874817378800.724424",
      "source": ".",
      "cache_key": null,
      "scm_info": {
        "scm": "git",
        "remote": null,
        "revision": "d691971666b4079580636ca8958ea928cb1ca064",
        "working_tree_clean": false,
        "published": false,
        "synchronized_remote_branches": []
      }
    },
    "source_options": {
      "path": "."
    }
  }
}
```

Exercise



GL: Generate a Policyfile

- Generate a *Policyfile* for Apache, including a run list
- Generate a *Policyfile.lock.json*
- Push a *Policyfile.lock.json* to Chef Infra Server
- Apply the *Policyfile* to a node

In this group lab we'll practice creating a policyfile

GL: Generate a Policyfile for apache

```
> cd /home/chef/chef-repo/policyfiles  
> chef generate policyfile apache  
  
Recipe: code_generator::policyfile  
  * template[/home/chef/chef-repo/policyfiles/apache.rb] action create  
    - create new file /home/chef/chef-repo/policyfiles/apache.rb  
    - update content in file /home/chef/chef-repo/policyfiles/apache.rb from  
      none to 10c22e  
      (diff output suppressed by config)
```

GL: Confirm the new Policyfile exists

```
> ls -l
```



```
total 8
-rw-rw-r-- 1 chef chef 543 Oct  6 17:41 apache.rb
-rw-r--r-- 1 chef chef 596 Sep 14 18:50 README.md
```

GL: View the Policyfile

```
> cat apache.rb

# Policyfile.rb - Describe how you want Chef Infra Client to build your system.
#
# For more information on the Policyfile feature, visit
# https://docs.chef.io/policyfile/

# A name that describes what the system you're building with Chef does.
name 'apache'

# Where to find external cookbooks:
default_source :supermarket

# run_list: chef-client will run these recipes in the order specified.
run_list 'apache::default'

# Specify a custom source for a single cookbook:
# cookbook 'example_cookbook', path: '../cookbooks/example_cookbook'
```

In the next step you'll need to add
the path to the apache cookbook.

Execute on: **workstation**

Execute from: : /home/chef/chef-repo/policyfiles

GL: Edit the new Policyfile

chef-repo/policyfiles/apache.rb

```
#...skipping for brevity...
# https://docs.chef.io/policyfile.html
# A name that describes what the system you're building with Chef does.
name 'apache'

# Where to find external cookbooks:
default_source :supermarket

# run_list: chef-client will run these recipes in the order specified.
run_list 'apache::default'

# Specify a custom source for a single cookbook:cookbook 'apache', path:
# cookbook 'example_cookbook', path: '../cookbooks/example_cookbook'
cookbook 'apache', path: '../cookbooks/apache'
```

Add the code in green.

You can edit this with
VS Code.

Notice how we need to specify the path to all dependent cookbooks.

```
# A name that describes what the system you're building with
# Chef does.
name 'apache'

# Where to find external cookbooks:
default_source :supermarket

# run_list: chef-client will run these recipes in the order
# specified.
run_list 'company_web::default'

# Specify a custom source for a single cookbook (local, not
# the Supermarket):
cookbook 'apache', path: '../cookbooks/apache'
```

Exercise



GL: Generate a Policyfile

- ✓ Generate a *Policyfile* for Apache, including a run list
- Generate a *Policyfile.lock.json*
- Push a *Policyfile.lock.json* to Chef Infra Server
- Apply the *Policyfile* to a node

In this group lab we'll practice creating a policyfile

Concept



Policyfile.rb and the Policyfile.lock.json

Now that we have our Policyfile.rb (apache.rb) we need to generate the Policyfile.lock.json (apache.lock.json) before we could upload the apache.lock.json to Chef Infra Server.

The `chef install policy_name` command creates the Policyfile.lock.json.

GL: Generate the apache.lock.json

```
> chef install apache.rb

uilding policy apache
Expanded run list: recipe[apache::default]
Caching Cookbooks...
Installing apache >= 0.0.0 from path

Lockfile written to /home/chef/chef-repo/policyfiles/apache.lock.json
Policy revision id:
e3db31b3bbc1804a8944ed05fe5c8dc6b84b369660bb176eef4922656a3d297f
```

GL: Confirm the new apache.lock.json exists

```
> ls -l
```

```
total 12
-rw-rw-r-- 1 chef chef 1018 Oct  6 17:54 apache.lock.json
-rw-rw-r-- 1 chef chef  590 Oct  6 17:46 apache.rb
-rw-r--r-- 1 chef chef  596 Sep 14 18:50 README.md
```

In practice, at this point you could upload the apache.lock.json to Chef Infra Server but we need to define one more thing before we do.

Execute on: **workstation**

Execute from: **/home/chef/chef-repo/policyfiles**

Exercise



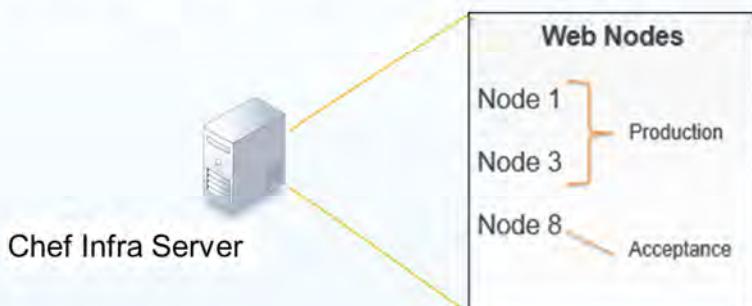
GL: Generate a Policyfile

- ✓ Generate a *Policyfile* for Apache, including a run list
- ✓ Generate a *Policyfile.lock.json*
- Push a *Policyfile.lock.json* to Chef Infra Server
- Apply the *Policyfile* to a node

In this group lab we'll practice creating a policyfile

Policy Group = Environment

At the time when you upload the Policyfile.lock.json to the Chef Infra Server is when you specify a policy group, which can act like an environment such as **production** or **acceptance**. You could also think of policy group as a way to group like servers together. **Note:** Policy group replaces the legacy environments.

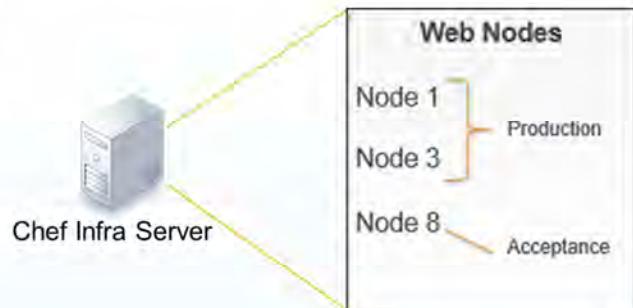


Policy Group = Environment

A policy group can best be defined as a logical separation of nodes.

For example, let's say you have a Production environment and an Acceptance environment.

You could create a **prod** policy group and an **acceptance** policy group and push your policyfile to either one, depending where you want that policyfile's node to reside.



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.

Policy Group

The first time you specify a policy group, that policy group name will be instantiated in Chef Infra Server. For example:

`chef push prod apache.lock.json` will create the policy group named **prod** and also upload the `apache.lock.json` (and its cookbooks) to the Chef Infra Server.

It will upload that `apache.lock.json` within the policy group named **prod**.

Pushing a Policyfile to Chef Infra Server

To push a policyfile to Chef Infra Server you need:

Policy name

and...

Policy group

These two items uniquely identify a single policyfile object on the Chef Infra Server.

GL: Push your apache.lock.json to prod on Chef Infra Server

```
> chef push prod apache.lock.json
```

```
Uploading policy apache (e3db31b3bb) to policy group prod  
Uploaded apache 0.1.0 (d3f5e6c1)
```

GL: Verify that your policy is on Chef Infra Server

```
> chef show-policy
```

```
apache
```

```
=====
```

```
* prod: e3db31b3bb
```

This output confirms that the policy is
on the Chef infra Server.

Execute on: **workstation**

Execute from: **anywhere**

Chef Automate: Policy (apache)

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Go to Policyfiles tab to view pushed policy



Note: Login to Chef Automate UI using username and password shared by the Instructor.

Instructor note :

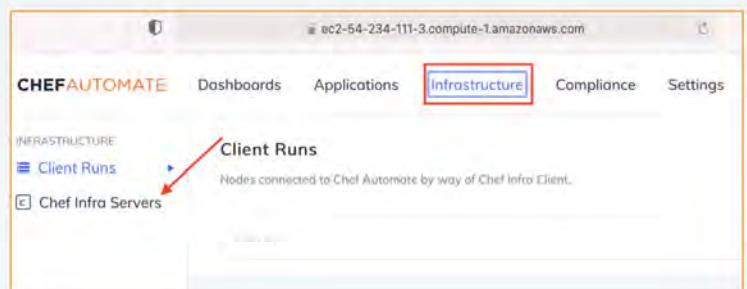
Please share below credentials with participants:

Username: studentXXuser-ca

Password: studentXX

Chef Automate: Policy (apache)

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Go to Policyfiles tab to view pushed policy



Chef Automate: Policy (apache)

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Go to Policyfiles tab to view pushed policy

Name	FQDN	IP Address	Number Of Orgs
cheftraining	ec2-54-234-111-3.compute-1.amazonaws.com	54.234.111.3	30

cheftraining

Chef Automate: Policy (apache)

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- **Click on the organization - 'studentxx'**
- Go to Policyfiles tab to view pushed policy

The screenshot shows the 'Chef Infra Servers' interface with the 'cheftraining' server selected. The top navigation bar shows 'Chef Infra Servers > cheftraining'. Below it, the server details are listed: FQDN 'ec2-54-234-111-3.compute-1.amazonaws.com' and IP Address '54.234.111.3'. There are two tabs: 'Orgs' (which is selected) and 'Details'. Under the 'Orgs' tab, there is a table with three columns: 'Name', 'Admin', and 'Projects'. The first row shows 'student01' as the name, 'student01' as the admin, and 'student01project' as the project. A red arrow points to the 'student01' entry in the 'Name' column.

Name	Admin	Projects
student01	student01	student01project

Chef Automate: Policy (apache)

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Go to Policyfiles tab to view pushed policy

The screenshot shows the Chef Automate web interface. The top navigation bar includes 'student01' and 'Projects student01project'. Below the navigation are tabs: Cookbooks, Roles, Environments, Data Bags, Clients, Nodes, and Policyfiles. The Policyfiles tab is highlighted with a red box and a red arrow points to the 'Name' column of the table below. The table lists a single item: 'apache' in the Name column and '450fb23145c2b9635544042d413ccf25fc6449640a2f46df6114a5c1a087a94f' in the Revision ID column. A search icon is visible at the top right of the table.

Exercise



GL: Generate a Policyfile

- ✓ Generate a *Policyfile* for Apache, including a run list
- ✓ Generate a *Policyfile.lock.json*
- ✓ Push a *Policyfile.lock.json* to Chef Infra Server
- Apply the *Policyfile* to a node

In this group lab we'll practice creating a policyfile

knife node policy set Command

In a moment you will apply the policyfile to a node.

To do so you will use the `knife node policy set` command.

Syntax:

```
knife node policy set NODE POLICY_GROUP POLICY_NAME
```

https://docs.chef.io/workstation/knife_node/#policy-set

https://docs.chef.io/workstation/knife_node/#policy-set

GL: Apply the apache policy to your Linux (apache_web) node

```
> knife node policy set apache_web prod apache
```

```
Successfully set the policy on node apache_web
```

node name

policy_group

policy_name

'knife node policy set' takes 3 arguments:

- * Node name
- * Policy group
- * Policy name

Execute on: **workstation**

Execute from: **anywhere**

GL: View information about your Linux node

```
$ knife node show apache_web
```

```
Node Name: apache_web
Policy Name: apache
Policy Group: prod
FQDN: ip-172-31-77-52.ec2.internal
IP: 44.205.9.57
Run List:
Recipes:
Platform: centos 7.6.1810
Tags:
```

You can see information about a particular node with the command 'knife node show *nodename*', such as 'knife node show apache_web'. This will display a summary of the node information that the Chef Infra Server stores.

Execute on: **workstation**

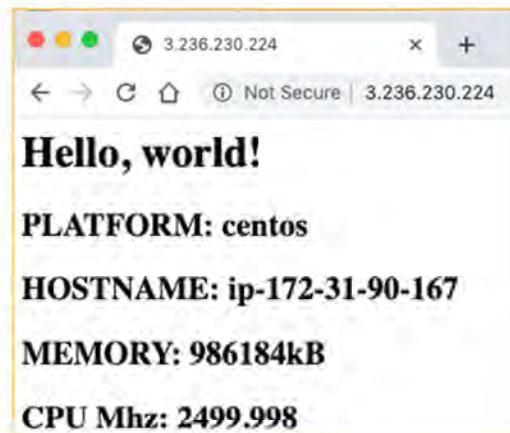
Execute from: **anywhere**

GL: Converge the Linux node with ssh

```
$ knife ssh IPADDRESS -m -x chef -P PWD 'sudo chef-client'

3.236.230.224 Starting Chef Infra Client, version 17.3.48
3.236.230.224 Using policy 'apache' at revision
'450fb23145c2b9635544042d413ccf25fc644964aa2f46df6114a5c1a087a94f'
3.236.230.224 resolving cookbooks for run list: ["apache::default@0.1.0 (1388ab3)"]
3.236.230.224 Synchronizing Cookbooks:
3.236.230.224   - apache (0.1.0)
3.236.230.224 Installing Cookbook Gems:
3.236.230.224 Compiling Cookbooks...
3.236.230.224 Converging 3 resources
3.236.230.224 Recipe: apache::server
3.236.230.224   * yum_package[httpd] action install
3.236.230.224     - install version 0:2.4.11-93.el7.centos.x86_64 of package httpd
3.236.230.224   * template[/var/www/html/index.html] action create
3.236.230.224     - create new file /var/www/html/index.html
...
3.236.230.224 Chef Infra Client finished, 4/4 resources updated in 06 seconds
```

GL: Verify that the Linux node serves the page



Exercise



GL: Generate a Policyfile

- ✓ Generate a *Policyfile* for Apache, including a run list
- ✓ Generate a *Policyfile.lock.json*
- ✓ Push a *Policyfile.lock.json* to Chef Infra Server
- ✓ Apply the *Policyfile* to a node

In this group lab we'll practice creating a policyfile

Review



Review Questions

1. What do policyfiles typically contain?
2. What can a policyfile's policy_name be used for?
3. What is a policy group?

1. A policyfile contains information about a node's source for fetching cookbooks and for setting run lists.
2. To reflect the purpose of the machines where the policy will run. In other words, policy_name can set the "role" for a node.
3. A policy group can best be defined as a logical separation of nodes into specific environments, such as production or acceptance environments.

Questions?



Q&A

What questions can we answer for you?

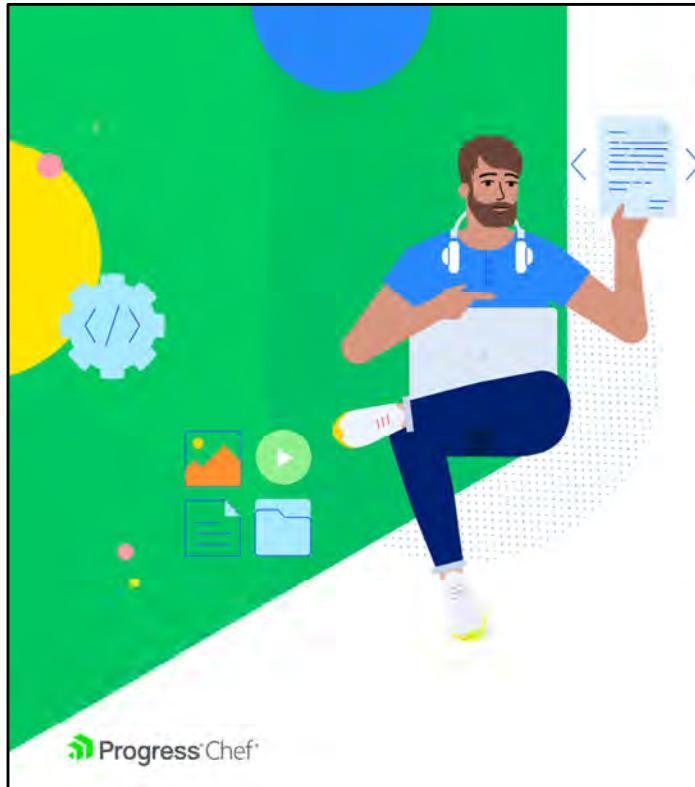




Cookbook Attributes, Attribute Files, and Dependencies

Setting Attributes within a Cookbook and
Applying the company_web Cookbook to Your
Nodes





Objectives

After completing this module, you should be able to:

- Explain where cookbook attributes reside
- Create a wrapper cookbook
- Configure dependencies between cookbooks
- Generate a new policyfile and policyfile.lock.json
- Upload the new policyfiles to Chef Infra server and converge the nodes

© 2020 Progress Software Corporation and/or its subsidiaries and/or affiliates. All rights reserved.

10- 2



Concept

Attribute files

The Node object contains many automatic attributes generated by OHAI.

You can also maintain attributes within a cookbook.

These are like variables or parameters for your cookbook and allow recipes to be data driven.

<https://docs.chef.io/attributes/>



©2022 Progress Software Corporation and/or its subsidiaries and/or affiliates. All rights reserved.

10- 3

While the node object that is generated by OHAI contains a large amount of data about the node, there will be situations where you want to create and set your own node attributes that can be used throughout your cookbooks and overridden when necessary. These user defined node attributes are much like variables or parameters that can allow for your cookbooks to be data driven.

<https://docs.chef.io/attributes/>

Concept



Best practices

- Well-written cookbooks change behavior based on attributes.
- Ideally, you don't have to modify the contents of a cookbook to use it for your specific use case.
- Look at the attributes directory for things you can override through roles to affect behavior of the cookbook.
- Of course, well written cookbooks have sane defaults, and a README to describe all this.

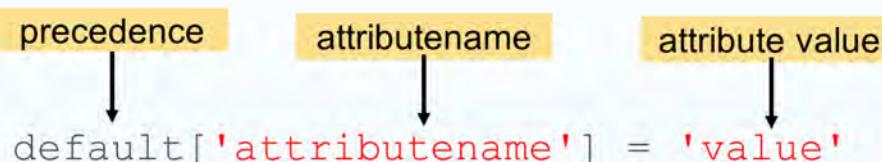
Ideally you will never have to manually edit the contents of a cookbook to be able to use it in a specific use case. The way that we can insure this is by creating cookbooks that are data driven and change their behavior based on attributes that can be overridden through the use of policy names and policy groups. When creating node attributes within your cookbook, it is a good idea to document the purpose of this within your README file.

Setting attributes in attribute files

Cookbook attributes are set in the attributes file

~/cookbooks/<cookbook>/attributes/default.rb

Format is:



Cookbook attributes are defined in an attribute file found in the 'attributes' directory of the cookbook. You first give a precedence for the attribute which will generally be default when you are first creating this attribute. We will talk more about attribute precedence later on in the course. This is then followed by the name of the attribute that you feel describes its purpose and set this equal to some value which can later be overridden.

Example: Setting package name to an attribute

```
cookbooks/apache/attributes/default.rb
```

```
default['apache']['package_name'] = 'httpd'
```

```
cookbooks/apache/recipes/default.rb
```

```
package node['apache']['package_name'] do
  action :install
end
```

We can set the name of a particular package to an attribute and then call that attribute within a recipe

In this example we see that instead of hard coding the name of the package resource, we can set this to a node attribute within the attribute file. This will resolve to 'httpd' when chef-client is run.

Example: Setting package name to an attribute

```
cookbooks/apache/attributes/default.rb
```

```
case node['platform']
when 'ubuntu'
  default['apache']['package_name'] = 'apache2'
else
  default['apache']['package_name'] = 'httpd'
end
```

Implementing conditional statements allows us to alter the control flow permitting our cookbooks to be data driven.

If we desire for our cookbook to be data driven, we can change what the `package_name` node attribute will resolve to based on platform information. If this code is executed on an ubuntu machine, the `package_name` will resolve to `'apache2'`. Otherwise, like on a Red Hat OS, this will resolve to `httpd`.

Node attribute precedence

	Attribute Files	Node/Recipe	Policy File
default	1	2	3
force_default	4	5	
normal	6	7	
override	8	9	10
force_override	11	12	
automatic from ohai highest priority			

We must set the precedence of node attributes which will allow them to be overridden when chef-client is executed. There are three locations where we can set a node attribute: Attribute files, recipes, and Policyfile.

With each location we assign a precedence like default, force_default, normal, override, and force_override. With each location and precedence level there is an assigned value. The higher the value, the higher the precedence, and whatever has the highest value will win out when chef-client is executed and the associated value for the node attribute will be used. At the top level we have automatic. Automatic node attribute precedence is reserved for node attributes collected by Ohai and cannot be overridden.

Concept



Reconfigure welcome message

Currently a welcome message is hard coded in both web server cookbooks.

What if we wanted to display a message that includes our company name utilizing a node attribute?

How could we implement this node attribute within both our 'myiis' and 'apache' cookbooks?

Let's consider this scenario. We might want have to have our company name displayed in our welcome message. However, instead of simply hard coding this in we might set this as a node attribute in case something changes and we could simply override this if necessary. But how do we implement this in both the apache and myiis cookbooks?

Exercise



GL: Reconfigure Welcome Message

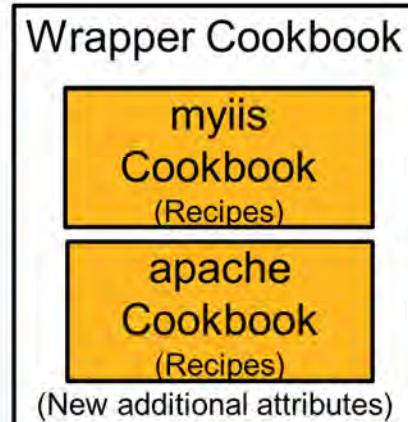
- Create a 'company_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- Create a node attribute that contains your company name
- Implement the edit_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- Create Policyfile and lock.
- Upload Policyfile.lock to the Chef Infra Server
- Converge the nodes

So we want both our web server cookbooks to display our company name.

Wrapper cookbooks

A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook(s).

It can access all of the recipes, cookbook components, and attributes found in the original cookbook(s) and implement them in new ways.



https://docs.chef.io/supermarket/install_supermarket/#create-a-wrapper

<https://www.chef.io/blog/doing-wrapper-cookbooks-right>

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.

Let's generate our wrapper cookbook named company_web.

https://docs.chef.io/supermarket/install_supermarket/#create-a-wrapper

<https://www.chef.io/blog/doing-wrapper-cookbooks-right>

GL: Generate the wrapper cookbook

```
$ cd /home/chef/chef-repo  
$ chef generate cookbook cookbooks/company_web
```

```
Generating cookbook company_web  
- Ensuring correct cookbook content
```

```
Your cookbook is ready. To setup the pipeline, type `cd  
cookbooks/company_web`, then run `delivery init`
```

Go ahead and generate a cookbook named 'company_web'. This will act as our wrapper cookbook that creates dependencies on both 'myiis' and 'apache'.

'cd' command

Execute on: **workstation**

Execute from: **anywhere**

'chef' command

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Create Dependency on apache and myiis

```
~/chef-repo/cookbooks/company_web/metadata.rb
```

```
name 'company_web'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures company_web'
version '0.1.0'
chef_version '>= 16.0'
depends 'myiis'
depends 'apache'
```

GL: Include recipe based on platform

~/chef-repo/cookbooks/company_web/recipes/default.rb

```
#  
# Cookbook:: company_web  
# Recipe:: default  
#  
# Copyright:: 2022, The Authors, All Rights Reserved.  
  
case node['platform']  
when 'windows'  
  include_recipe 'myiis::default'  
else  
  include_recipe 'apache::default'  
end
```

Because we have set dependencies for the 'myiis' and 'apache' cookbooks we may now include the default recipes from these cookbooks. By using a case statement that checks the platform of the node, when this is executed on a windows machine it will apply the 'myiis' cookbook's default recipe. Otherwise, as on a Linux machine, it will apply the 'apache' default recipe.

```
case node['platform']  
when 'windows'  
  include_recipe 'myiis::default'  
else  
  include_recipe 'apache::default'  
end
```

Exercise



GL: Reconfigure Welcome Message

- ✓ Create a 'company_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- Create a node attribute that contains your company name
- Implement the edit_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- Create Policyfile and lock.
- Upload Policyfile.lock to the Chef Infra Server
- Converge the nodes

So we want both our web server cookbooks to display our company name.

GL: Generate the default attribute file

```
$ cd ~/chef-repo
$ chef generate --help
```

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

The chef generate command is also capable of creating an attribute file with the corresponding 'attributes' directory if it doesn't already exist.

'cd' command

Execute on: **workstation**

Execute from: **anywhere**

'chef' command

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Generate the default attribute file

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

```
Usage: chef generate attribute [path/to/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@example.com'
      -a, --generator-arg KEY=VALUE    Use to set arbitrary attribute KEY to
      VALUE in the code_generator cookbook
      -h, --help                         Show this message
```

'chef' command

Execute on: **workstation**

Execute from: **Anywhere**

GL: Generate the default attribute file

```
$ chef generate attribute cookbooks/company_web default

recipe: code_generator::attribute
  * directory[cookbooks/company_web/attributes] action create
    - create new directory cookbooks/company_web/attributes
  * template[cookbooks/company_web/attributes/default.rb] action create
    - create new file cookbooks/company_web/attributes/default.rb
    - update content in file
cookbooks/company_web/attributes/default.rb from none to e3b0c4
  (diff output suppressed by config)
```

Let's generate a default attribute file where we can set our own node attributes.

Execute on: **workstation**

Execute from: **~\chef-repo**

GL: Set the Company Name as an Attribute

```
cookbooks/company_web/attributes/default.rb
```

```
default['company_web']['company_name'] = 'Your Company Name'
```

You can replace the 'Your Company Name' value with **Chef** or any name you like.

Now that we have our default attribute file, let's define a node attribute called 'company_name' and set this to whatever value you like.

```
default['company_web']['company_name'] = 'Your Company Name'
```

Exercise



GL: Reconfigure Welcome Message

- ✓ Create a 'company_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- ✓ Create a node attribute that contains your company name
 - Implement the edit_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
 - Create Policyfile and lock.
 - Upload Policyfile.lock to the Chef Infra Server
 - Converge the nodes

So we want both our web server cookbooks to display our company name.

Concept



Using the company_name Attribute

We are now able to apply a different default recipe based on whether the node's platform is Windows or Centos, but how do we update the respective template file to display the company_name attribute for both the 'myiis' and 'apache' cookbooks?

We are able to have access to this new node attribute within our cookbook, but how do we utilize it from both the 'apache' and 'myiis' cookbooks considering that they use different template resources to create the welcome page?



Concept

edit_resource

A recipe can find a resource in the resource collection, and then edit it by using the `edit_resource` method. If a resource block with the same name exists in the resource collection, it will be updated with the contents of the resource block.

https://docs.chef.io/infra_language/editing_resources/#edit_resource

One way we might go about changing the template resource is by using the `edit_resource` method. When you apply a run list to a node, it creates what's known as a resource collection. This resource collection is a list of all the resources that will be applied to the node and using the `edit_resource` method we can change some of the properties of these resources. Perhaps we could change the source for the template resource?

https://docs.chef.io/infra_language/editing_resources/#edit_resource

GL: View the server recipes

```
/myiis/recipes/server.rb
```

```
windows_feature 'web-server' do
  action :install
end

template 'c:\inetpub\wwwroot\Default.htm' do
  source 'Default.htm.erb'
end

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

```
/apache/recipes/server.rb
```

```
package 'httpd'

template '/var/www/html/index.html' do
  source 'index.html.erb'
end

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



We want to use a new source for the template resource for both our cookbooks

The 'myiis' and 'apache' cookbooks use different ERB templates for the creation of the homepage. We want to change this and use a new template that uses our new 'company_name' node attribute.

GL: Edit the template resource for myiis

```
~/chef-repo/cookbooks/company_web/recipes/default.rb

case node['platform']
when 'windows'
    include_recipe 'myiis::default'

    edit_resource(:template, 'c:\inetpub\wwwroot\Default.htm') do
        source 'homepage.erb'
        cookbook 'company_web'
    end
#else statement...
```

GL: Edit the template resource for apache

```
~/chef-repo/cookbooks/company_web/recipes/default.rb

#When Statement...
else
  include_recipe 'apache::default'

  edit_resource(:template, '/var/www/html/index.html') do
    source 'homepage.erb'
    cookbook 'company_web'
  end
end
```

Within the 'else' statement we edit the template resource for the apache default recipe as well as providing the same source for the resource.

GL: View the default recipe

```
~/chef-repo/cookbooks/company_web/recipes/default.rb

case node['platform']
when 'windows'
  include_recipe 'myiis::default'
  edit_resource(:template, 'c:\inetpub\wwwroot\Default.htm') do
    source 'homepage.erb'
    cookbook 'company_web'
  end

else
  include_recipe 'apache::default'
  edit_resource(:template, '/var/www/html/index.html') do
    source 'homepage.erb'
    cookbook 'company_web'
  end
end
```

This is what the code will look like when the default recipe is complete.

```
case node['platform']
when 'windows'
  include_recipe 'myiis::default'
  edit_resource(:template, 'c:\inetpub\wwwroot\Default.htm') do
    source 'homepage.erb'
    cookbook 'company_web'
  end

else
  include_recipe 'apache::default'
  edit_resource(:template, '/var/www/html/index.html') do
    source 'homepage.erb'
    cookbook 'company_web'
  end
end
```

GL: Generate the template

```
$ chef generate template cookbooks/company_web homepage

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

Use '**chef generate template**' to create a template in the company_web cookbook directory. The file we want to create is named as homepage.erb

Execute on: **workstation**

Execute from: **~\chef-repo**

GL: Update the template file

```
~/chef-repo/cookbooks/company_web/templates/homepage.erb
```

```
<html>
  <body>
    <h1><%= node['company_web']['company_name'] %> Welcomes You!</h1>
    <h2>PLATFORM: <%= node['platform'] %></h2>
    <h2>HOSTNAME: <%= node['hostname'] %></h2>
    <h2>MEMORY: <%= node['memory']['total'] %></h2>
    <h2>CPU Mhz: <%= node['cpu'][0]['mhz'] %></h2>
  </body>
</html>
```

Note: We are adding all this code but we are also highlighting the ['company_web']['company_name'] attributes for the discussion below.

Now at last we use our 'company_name' node attribute. This is essentially the same html page that was generated from the apache and myis cookbooks with the exception that we now have a new welcome message making use of our 'company_name' node attribute.

```
<html>
  <body>
    <h1><%= node['company_web']['company_name'] %> Welcomes You!</h1>
    <h2>PLATFORM: <%= node['platform'] %></h2>
    <h2>HOSTNAME: <%= node['hostname'] %></h2>
    <h2>MEMORY: <%= node['memory']['total'] %></h2>
    <h2>CPU Mhz: <%= node['cpu'][0]['mhz'] %></h2>
  </body>
</html>
```

Exercise



GL: Reconfigure Welcome Message

- ✓ Create a 'company_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- ✓ Create a node attribute that contains your company name
- ✓ Implement the edit_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
 - Create Policyfile and lock.
 - Upload Policyfile.lock to the Chef Infra Server
 - Converge the nodes

So we want both our web server cookbooks to display our company name.

Concept



Policyfile.rb and the Policyfile.lock.json

Now that we have our company_web cookbook in our chef-repo, we can create our Policyfile.rb and then generate our Policyfile.lock.json as we discussed in the previous module.

We'll name our Policyfile **company_web**.

GL: Generate the company_web Policyfile

```
> cd ~/chef-repo  
> chef generate policyfile policyfiles/company_web
```

```
Recipe: code_generator::policyfile  
  * template[C:/Users/Administrator/chef-repo/policyfiles/company_web.rb]  
action create  
  - create new file C:/Users/Administrator/chef-  
repo/policyfiles/company_web.rb  
  - update content in file C:/Users/Administrator/chef-  
repo/policyfiles/company_web.rb from none to c0bae7  
    (diff output suppressed by config)
```

‘cd’ command

Execute on: **workstation**

Execute from: **anywhere**

‘chef’ command

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Verify that the Policyfile exists

```
> ls -l policyfiles
```

```
total 16
-rw-rw-r-- 1 chef chef 1018 Oct  6 17:57 apache.lock.json
-rw-rw-r-- 1 chef chef  590 Oct  6 17:46 apache.rb
-rw-rw-r-- 1 chef chef  641 Oct  7 16:39 company_web.rb
-rw-r--r-- 1 chef chef  596 Sep 14 18:50 README.md
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Edit the company_web.rb Policyfile

~/chef-repo/policyfiles/company_web.rb

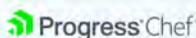
```
#...skipping for brevity...
# https://docs.chef.io/policyfile.html
# A name that describes what the system you're building with Chef does.
name 'company_web'

# Where to find external cookbooks:
default_source :supermarket

# run_list: chef-client will run these recipes in the order specified.
run_list 'company_web::default'

# Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
```

Add the code in green.



© 2022 Progress Software Corporation. All rights reserved. All rights reserved.

10- 33

Notice how we need to specify the path to all dependent cookbooks too (myiis and apache).

```
# Policyfile.rb - Describe how you want Chef Infra Client to
build your system.
#
# For more information on the Policyfile feature, visit

# A name that describes what the system you're building with
Chef does.
name 'company_web'

# Where to find external cookbooks:
default_source :supermarket

# run_list: chef-client will run these recipes in the order
specified.
run_list 'company_web::default'

# Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
```

GL: Generate the company_web.lock.json

```
> chef install policyfiles/company_web.rb

Building policy company_web
Expanded run list: recipe[company_web::default]
Caching Cookbooks...
Installing company_web >= 0.0.0 from path
Installing myiis      >= 0.0.0 from path
Installing apache     >= 0.0.0 from path

Lockfile written to C:/Users/Administrator/chef-
repo/policyfiles/company_web.lock.json
Policy revision id:
f182397fa9ba967cf9c8749806ee9fcc04b16387b455583bb0ac2503d7c6e3b8
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Verify the company_web.lock.json exists

```
> ls -l policyfiles
```

```
total 20
-rw-rw-r-- 1 chef chef 1018 Oct  6 17:57 apache.lock.json
-rw-rw-r-- 1 chef chef  590 Oct  6 17:46 apache.rb
-rw-rw-r-- 1 chef chef 2462 Oct  7 16:42 company_web.lock.json
-rw-rw-r-- 1 chef chef  790 Oct  7 16:42 company_web.rb
-rw-r--r-- 1 chef chef  596 Sep 14 18:50 README.md
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Move to the policyfiles directory

```
> cd policyfiles
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Push the Policyfile to the Chef Infra Server

```
> chef push prod company_web.lock.json
```

```
Uploading policy company_web (d341c1185f) to policy group prod
Using    apache      0.1.0 (d3f5e6c1)
Uploaded company_web 0.1.0 (138758ac)
Uploaded myiis       0.2.1 (cd078444)
```

Execute on: **workstation**

Execute from: **~/chef-repo/policyfiles**

GL: Verify the Policyfile is on Chef Infra Server

```
> chef show-policy
```

```
apache
=====
* prod: e3db31b3bb

company_web
=====
* prod: d341c1185f
```

Here we can see that the **company_web** policy has been uploaded to Chef Infra Server and is in the **prod** policy group.

Execute on: **workstation**
Execute from: **anywhere**

Exercise



GL: Reconfigure Welcome Message

- ✓ Create a 'company_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- ✓ Create a node attribute that contains your company name
- ✓ Implement the edit_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- ✓ Create Policyfile and lock.
- ✓ Upload Policyfile.lock to the Chef Infra Server
- Converge the nodes

So we want both our web server cookbooks to display our company name.

GL: Apply the policy to your Windows node

```
> knife node policy set iis_web prod company_web
```

```
Successfully set the policy on node iis_web
```

node name

policy_group

policy_name

'knife node policy set' takes 3 arguments:

- * node name
- * policy group
- * policy name

Execute on: **workstation**

Execute from: **anywhere**

GL: View information about your Windows node

```
$ knife node show iis_web
```

```
Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP
IP: 107.20.24.200
Run List:
Recipes:
Platform: windows 6.3.9600
Tags:
```

You can see more information about a particular node with the command 'knife node show iis_web'. This will display a summary of the node information that the Chef Infra Server stores.

Execute on: **workstation**

Execute from: **anywhere**

GL: Converge your Windows node via winrm

```
$ knife winrm IPADDRESS -m -x Administrator -P PASSWORD "chef-client"

...
Synchronizing Cookbooks:
- myiis (0.2.1)
  3.88.178.251 - apache (0.1.0)
  3.88.178.251
  3.88.178.251 - company_web (0.1.0)
  3.88.178.251 Installing Cookbook Gems:
  3.88.178.251 Compiling Cookbooks...
* windows_service[w3svc] action start (up to date)

...
3.88.178.251 Running handlers:
3.88.178.251 Running handlers complete
3.88.178.251 Chef Infra Client finished, 2/4 resources updated in 48 seconds
```

So if you want to execute "chef-client" run for your iis_web node, you should write out this command. You would need to provide the username to log into the system, the password for that system (or an SSH key if you are using one), and then finally the command to execute. For more security, you should use API or ssh keys and forego specifying a username and password.

If you have a cookbook error and need to update and re-push your policyfile, do so in this order:

1. Update your code
2. Re-run 'chef update company_web.rb'
3. Re-run 'chef push prod company_web.lock.json'
4. Re-converge the node using the 'knife winrm' command

GL: Apply the policy to your Linux node

```
> knife node policy set apache_web prod company_web
```

```
Successfully set the policy on node apache_web
```

node name

policy_group

policy_name

'knife node policy set' takes 3 arguments:

- * node name
- * policy group
- * policy name

Execute on: **workstation**

Execute from: **anywhere**

GL: View information about your Linux node

```
$ knife node show apache_web
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-62-68.ec2.internal
IP: 18.206.64.141
Run List:
Recipes:
Platform: centos 7.6.1810
Tags:
```

You can see information about a particular node with the command 'knife node show *nodename*'. This will display a summary of the node information that the Chef Infra Server stores.

Execute on: **workstation**

Execute from: **anywhere**

GL: Converge the Linux node with ssh

```
$ knife ssh IPADDRESS -m -x chef -P PWD 'sudo chef-client'

...
Synchronizing Cookbooks:
- myiis (0.2.1)
  3.88.178.251 - apache (0.1.0)
  3.88.178.251
  3.88.178.251 - company_web (0.1.0)
  3.88.178.251 Installing Cookbook Gems:
  3.88.178.251 Compiling Cookbooks...
* windows_service[w3svc] action start (up to date)

...
3.88.178.251 Running handlers:
3.88.178.251 Running handlers complete
3.88.178.251 Chef Infra Client finished, 2/4 resources updated in 48 seconds
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Verify policy applied to nodes

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Click on Nodes tab and select node iis_web
- View Information associated with node



Note: Login to Chef Automate UI using username and password shared by the Instructor.

Instructor note :

Please share below credentials with participants:

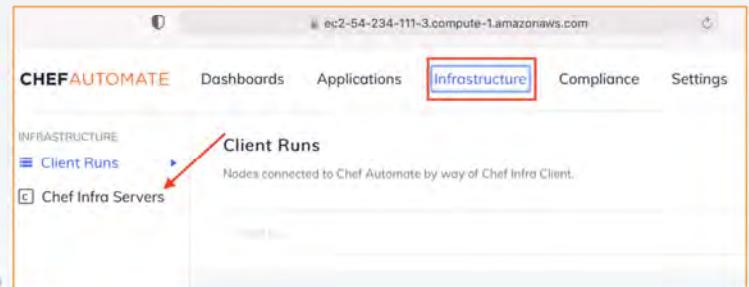
Username: studentXXuser-ca

Password: studentXX

NOTE: If you are unable to login (getting an 'Unauthorized' error or a 'Bad relay state' error), try clearing your browser cache or try using a different browser.

GL: Verify policy applied to nodes

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Click on Nodes tab and select node iis_web
- View Information associated with node



GL: Verify policy applied to nodes

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- **Click on Listed Chef Infra Server - 'cheftraining'**
- Click on the organization - 'studentxx'
- Click on Nodes tab and select node iis_web
- View Information associated with node

Name	FQDN	IP Address	Number Of Orgs
cheftraining	ec2-54-234-111-3.compute-1.amazonaws.com	54.234.111.3	30

GL: Verify policy applied to nodes

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- **Click on the organization - 'studentxx'**
- Click on Nodes tab and select node iis_web
- View Information associated with node

The screenshot shows the 'Chef Infra Servers' interface with the URL 'Chef Infra Servers > cheftraining'. The organization 'cheftraining' is selected. The FQDN is listed as 'ec2-54-234-111-3.compute-1.amazonaws.com' and the IP Address is '54.234.111.3'. Below this, there are tabs for 'Orgs' and 'Details'. The 'Orgs' tab is active. A table lists nodes under the 'Details' tab, showing columns for 'Name', 'Admin', and 'Projects'. The node 'student01' is highlighted with a red arrow pointing to its 'Name' column. The 'Admin' column for student01 is 'student01' and the 'Projects' column is 'student01project'.

Name	Admin	Projects
student01	student01	student01project

GL: Verify policy applied to nodes

- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- **Click on Nodes tab and select node iis_web**
- View Information associated with node

The screenshot shows the Chef Automate web interface. At the top, it says "student01" and "Projects student01project". Below that is a navigation bar with tabs: Cookbooks, Roles, Environments, Data Bags, Clients, **Nodes**, and Policyfiles. The "Nodes" tab is highlighted with a red box. Underneath the tabs, there is a search bar with placeholder text "Search nodes by name...". The main area displays a table of nodes. The columns are: Node, Platform, FQDN, IP Address, Uptime, Last Check-in, and Environment. One row is selected, showing the node "iis_web" (Platform: windows, FQDN: WIN-DQFQCUFH0, IP Address: 172.31.24.134, Uptime: 1 day, Last Check-in: --, Environment: prod). A red arrow points to the "iis_web" node in the list.

GL: Verify policy applied to nodes

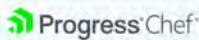
- Sign In to Chef Automate server
- Click on Infrastructure tab and Go to Chef Infra Servers (Under Left Navigation Panel)
- Click on Listed Chef Infra Server - 'cheftraining'
- Click on the organization - 'studentxx'
- Click on Nodes tab and select node iis_web
- **View Information associated with node**

The screenshot shows the 'Nodes' page in the Chef Infra Servers interface. The URL is 'Chef Infra Servers > Organizations > Nodes > iis_web'. The node name 'iis_web' is displayed at the top. Below it, there are two sections: 'NODE INFORMATION' and 'METADATA'. In the 'NODE INFORMATION' section, the 'Environment' is listed as 'prod' and the 'Policy Name' is listed as 'company-web'. In the 'METADATA' section, the 'Chef Server' is 'cheftraining' and the 'Chef Organization' is 'student01'. At the bottom, there are tabs for 'Details' (which is selected), 'Run list', and 'Attributes'. A dropdown menu for 'Environment' is open, showing 'prod' as the current selection. The footer of the interface includes the Progress Chef logo and the text '© 2022 Progress Software Corporation. All rights reserved. All rights reserved.'

GL: Verify policy applied to nodes

View Information associated with node.

The screenshot shows the Chef Infra Server interface with the URL [Chef Infra Servers > Organizations > Nodes > apache_web](#). The node name 'apache_web' is highlighted with a red box. Below it, the 'NODE INFORMATION' section shows 'Environment' as 'prod', 'Policy Group' as 'prod', and 'Policy Name' as 'company-web'. To the right, the 'METADATA' section shows 'Chef Server' as 'cheftraining' and 'Chef Organization' as 'student01'. At the bottom, there are tabs for 'Details', 'Run list', and 'Attributes', with 'Run list' being active. A dropdown menu under 'Run list' is set to 'prod'. A red arrow points from the 'Environment' field in the 'NODE INFORMATION' section to the 'Environment' dropdown in the 'Run list' section. Another red arrow points from the 'Policy Group' field in the 'NODE INFORMATION' section to the same dropdown.



Verify Policy applied to apache_web, to verify login to Chef Automate and open apache_web node.

if you don't remember how to do this, refer back to previous slides in this module to follow steps.

GL: Verify the webpage on iis_web



Verify that the node serves up the default html by pointing a web browser to the IP address of your Windows (iis_web) node.

GL: Verify the webpage on apache_web



Verify that the node serves up the default html by pointing a web browser to the IP address of your Linux(apache_web) node.

Exercise



GL: Reconfigure Welcome Message

- ✓ Create a 'company_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- ✓ Create a node attribute that contains your company name
- ✓ Implement the edit_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- ✓ Create Policyfile and lock.
- ✓ Upload Policyfile.lock to the Chef Infra Server
- ✓ Converge the nodes

So we want both our web server cookbooks to display our company name.



Review

Review

Attributes are like parameters to your cookbook, not hard-coded values in recipes or templates. Can you think of some other parameters that you might want to create attributes for?

Can you imagine in complex topologies where you could have multiple levels of dependencies between cookbooks?



Questions?

Q&A

What questions can we answer for you?

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





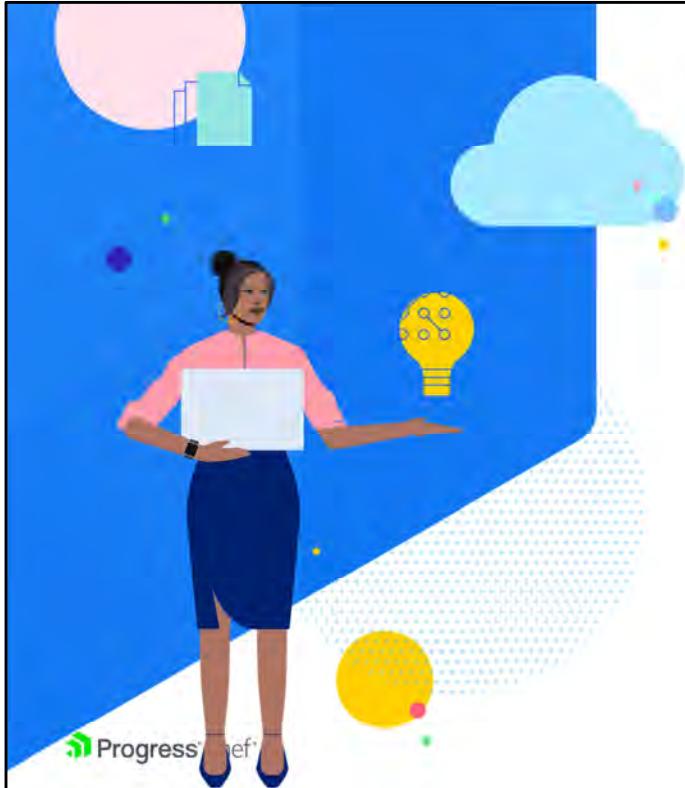
Community Cookbooks

Find, Explore and View Chef Cookbooks

1L



Instructor Note: The "1L" means you will need 1 new Linux node for each student in this module. It will be bootstrapped and used as a load balancer. Use a new fresh Linux instance for this...not the Linux node used previously in this course.



Objectives

After completing this module, you should be able to

- Find cookbooks on the Chef Supermarket
- Create a wrapper cookbook for a load balancer community cookbook
- Utilize Custom Resources
- Create and upload a Policyfile to Chef Infra Server
- Bootstrap a new node that runs the Policyfile's cookbook
- Test the load balancer

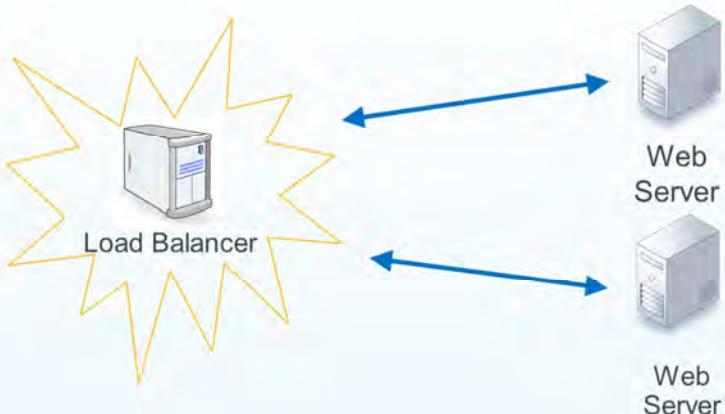
© 2019 Progress Software Corporation. All rights reserved. All rights reserved.

11- 2

Load balancer

Adding a load balancer will allow us to better grow our infrastructure.

Receives requests and relays them to other systems.



With two web servers running within our organization, it's now time to talk about the next goal to tackle. We need to setup a load balancer.

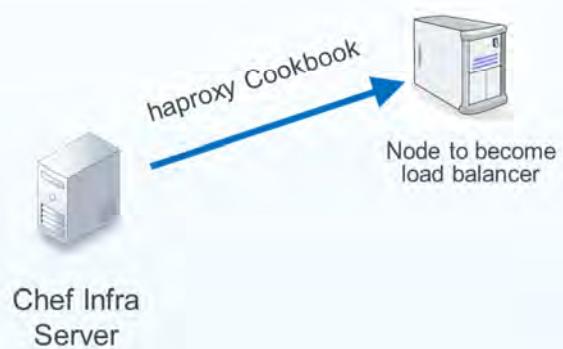
A load balancer is able to receive requests and relay them to other systems. In our case, we specifically want to use the load balancer to balance the entire traffic load between two or more systems.

This means we will need to establish a new node within our organization, install the necessary software to make the node a load balancer, and configure it so that it will relay requests to our existing nodes running apache and iis and to future nodes.

Load balancer

Work that needs to be accomplished to setup a load balancer within our infrastructure:

- Write a haproxy (load balancer) cookbook.
- We will need to establish a new node within our organization to which we apply that cookbook.



Similar to how we installed and configured apache on our first node, we could do the same thing here with a load balancer. The package name for the application will be 'haproxy', you will learn which file manages the configuration and 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 should spend some time focused on composing the cookbook recipe and testing it on our platform with our custom configuration.

Concept

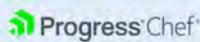


Community Cookbooks

Someone already wrote that cookbook?

Available through the community site called the Chef Supermarket

<https://supermarket.chef.io>



©2022 Progress Software Corporation and/or its affiliated companies. All rights reserved.

11- 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 everyone's time.

<https://supermarket.chef.io>

Concept



Types of Community Cookbooks

Chef Community Cookbooks fall into two broad categories: **Resource Cookbooks** and **Recipe Cookbooks**.

There is no strict naming convention, and the patterns can be mixed. Generally Resource Cookbooks provide extensions for Chef Infra by defining new Chef Resources, and Recipe Cookbooks use recipes to declare how a system should be configured. The difference is in what the end user uses in their wrapper cookbook: a custom resource or a recipe.

Concept



Types of Community Cookbooks

So far this class has focused on writing Recipe Cookbooks. We have written Recipes and declared Resources within those Recipes to specify how a system should be configured.

But you can extend Chef Infra by writing your own Resources! Now we will examine a Community Cookbook that provides an extension to Chef Infra by defining a new type of resource we can use inside of our Recipes.

Note: if you want to learn how to write your own custom resources, see the Chef Intermediate course. Your instructor can help you find this course description.

Exercise



Group Lab: Load Balancer

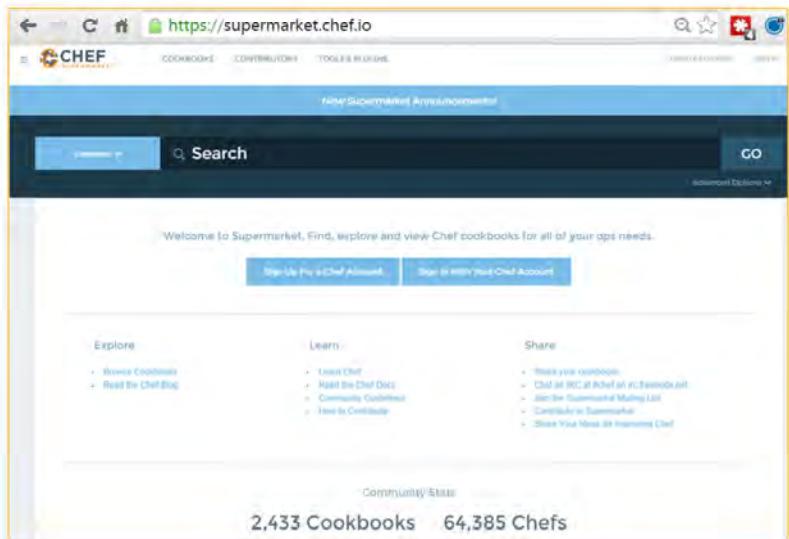
- Find a Cookbook on the Chef Supermarket to Manage a load balancer
- Configure the load balancer to send traffic to the `iis_web` and `apache_web` nodes
- Create myhaproxy Policyfile
- Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Adding a load balancer will allow us to better grow our infrastructure.

Let's find the haproxy (load balancer) cookbook within the community site to learn more about it.

GL: 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.



11- 9

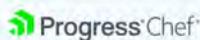
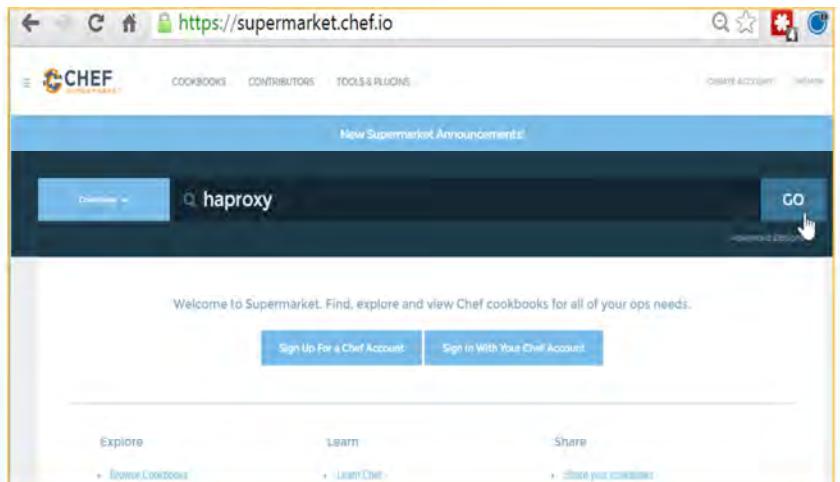
An important thing to remember is that on the community site are cookbooks managed by individuals. Chef does not verify or approve the cookbooks found in the Supermarket. These cookbooks solved problems for the original authors and then they decided to share them. This means that the cookbooks you find 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.

GL: Searching in the Supermarket

STEPS

- Visit supermarket.chef.io
- Select the search field and type in [haproxy](#) in the search field. Then click the **GO** button.
- Click the resulting [haproxy](#) link.



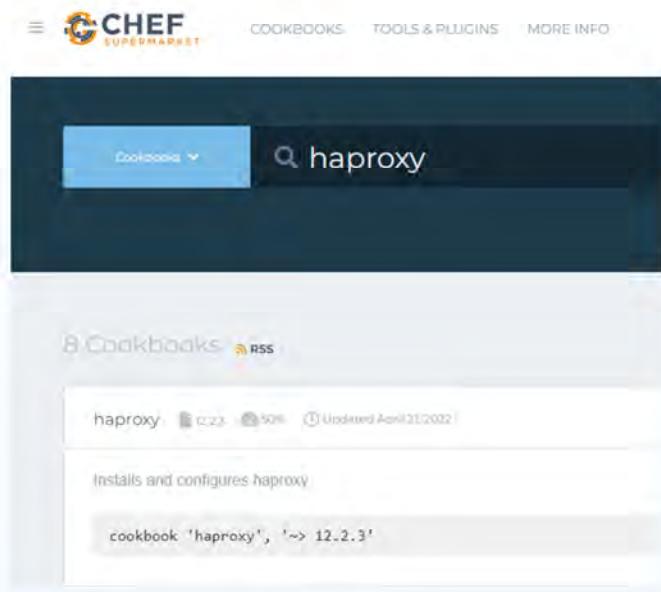
From the Supermarket main page type the search term "haproxy" and then click the GO button.

Below the search term will show us all the matching cookbooks. The "haproxy" cookbook is in that result set.

GL: Searching in the Supermarket

STEPS

- Visit supermarket.chef.io
- Select the search field and type in haproxy in the search field. Then click the **GO** button.
- Click the resulting **haproxy** link.



11-11

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.

GL: Supermarket Cookbooks

On the left, we are presented with the various ways we can install the cookbook...

On the right side we can see the individuals that maintain the cookbook...



11-12

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. Sous chefs work closely with Progress.

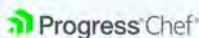
On the left, we are presented with the various ways we can install the cookbook, the README that describes information about the cookbook, any cookbooks that this cookbook may depend on, a history of the changes, and its food critic (quality) rating--which is a code evaluator for best practices.

GL: Supermarket Cookbooks

The area to focus most of your attention from the beginning is the README.

Reading and understanding the README at a glance is difficult. It is a skill that comes with time.

The screenshot shows the Chef Supermarket page for the 'haproxy Cookbook'. At the top, there are tabs for 'README', 'Description', 'Changes', and 'Utility'. The 'README' tab is selected. Below the tabs, the title 'haproxy Cookbook' is displayed, along with a green 'Chef Supermarket' badge, a blue 'cookbook v0.2.3', a green 'Issues 10', a green 'Solutions 1', and a green 'License Apache 2.0'. A brief description follows: 'Installs and configures HAProxy'. Under 'Maintainers', it says 'This cookbook is maintained by the Sous Chefs. The Sous Chefs are a community of Chef cookbook maintainers working together to maintain important cookbooks. If you'd like to know more please visit [sous-chefs.org](#) or come chat with us on the Chef Community Slack in [#supermarket](#)'. The 'Requirements' section lists 'HAProxy stable or LTS' and 'Chef 13.9+'. The 'Platforms' section lists 'debian: 9 & 10', 'ubuntu: 20.04 & 21.04', 'centos: 7 & 8', and 'centos-stream: 8'. At the bottom, there are links for 'View on GitHub', 'View on Supermarket', and 'View on Chef.io'.



11-13

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, notice the version of the chef-client that is required to run the latest version of the cookbook. If you're not using the latest major version of the client you may need to investigate older versions of the haproxy cookbook.

GL: Supermarket Cookbooks

The README defines a number of new Resources that can be used if this Cookbook is included in a node's run-list.

These Custom Resources are defined with a Cookbook's resources/ directory. A well-written README will define the actions and properties a Custom Resource accepts.

Common Resource Features

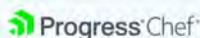
HAProxy has many configurable options available, this cookbook makes the most popular options available as resource properties.

If you wish to use a HAProxy property that is not listed the `extra_options` hash is available to take in any number of additional values.

For example, the ability to disable listeners is not provided out of the box. Further examples can be found in either `test/fixtures/recipes` or `spec/test/recipes`. If you have questions on how this works or would like to add more examples so it is easier to understand, please come talk to us on the [Chef Community Slack](#) on the #sous-chefs channel.

```
haproxy_listen 'disabled' do
  bind '0.0.0.0:1337'
  mode 'http'
  extra_options('disabled': '')
end
```

https://docs.chef.io/custom_resources/



© 2022 Progress Software Corporation. All rights reserved. All rights reserved.

11-14

A custom resource functions just like any other resource, using a name, actions and properties.

```
custom_resource "name" do
  action :action_name
  property 'property_value'
end
```

Often, common examples of using the Custom Resources will be shown in the `text/fixtures/recipes` directory.

https://docs.chef.io/custom_resources/

GL: Supermarket Cookbooks

Further down in the README you'll see the full list of Custom Resources that the haproxy Cookbooks provide.

We'll use of a number of these resources to easily configure a haproxy server and forward traffic to our web servers. This pattern provides an easy interface for consumers of the cookbook to change the way haproxy is deployed, without having to write all the logic yourself.

Resources

- [haproxy_acl](#)
- [haproxy_backend](#)
- [haproxy_cache](#)
- [haproxy_config_defaults](#)
- [haproxy_config_global](#)
- [haproxy_fastcgi](#)
- [haproxy_frontend](#)
- [haproxy_install](#)
- [haproxy_listen](#)
- [haproxy_mailer](#)
- [haproxy_peer](#)
- [haproxy_resolver](#)
- [haproxy_service](#)
- [haproxy_use_backend](#)
- [haproxy_userlist](#)

Concept



Using Community Cookbooks

Chef Community Cookbooks can be used as-is but in most cases you will want to use them as a foundation as you write your own.

Don't use forked community cookbooks in production, or you will miss out on upstream changes, and will have to rebase. Instead use **wrapper cookbooks**.

Because we want to be able to get upstream updates to these community cookbooks, rather than simply forking them we will create another wrapper cookbook.

GL: Supermarket Cookbooks

Reminder: A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook.

It can define new default values for the recipes.

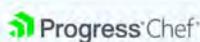
Wrapper Cookbook

haproxy
Cookbook
(Attributes)

(New additional attributes)

https://docs.chef.io/supermarket/install_supermarket/#create-a-wrapper

<https://www.chef.io/blog/doing-wrapper-cookbooks-right>



©2022 Progress Software Corporation. All rights reserved. All logos are trademarks of their respective companies.

11-17

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 and resources.

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.

Let's 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'.

https://docs.chef.io/supermarket/install_supermarket/#create-a-wrapper

<https://www.chef.io/blog/doing-wrapper-cookbooks-right>

Exercise



Group Lab: Load Balancer

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- Configure the load balancer to send traffic to the `iis_web` and `apache_web` nodes
- Create `myhaproxy` Policyfile
- Upload `myhaproxy Policyfile.lock` to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the `myhaproxy` (load balancer) cookbook

Adding a load balancer will allow us to better grow our infrastructure.

Let's find the haproxy (load balancer) cookbook within the community site to learn more about it.

Instructor Note:

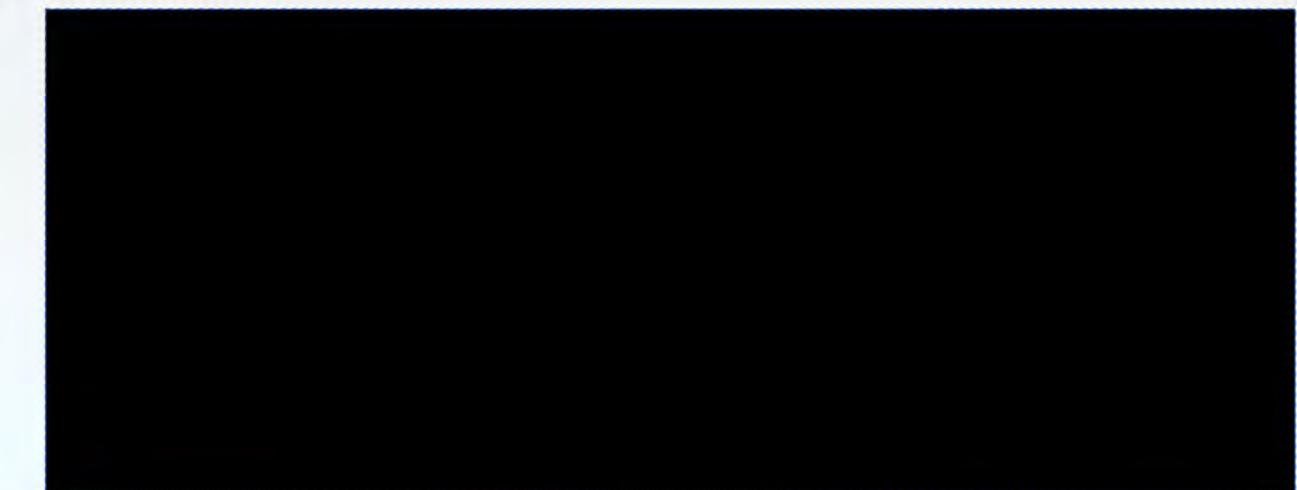
The GitHub repo for the 'myhaproxy' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-myhaproxy.git>

Because the student must provide the IP and hostname of their own web server, this must be manually added if they download a copy of the 'myhaproxy' cookbook.

GL: Returning to the Chef Repository directory



```
$ cd ~/chef-repo
```



Change to your chef-repo directory

Instructor Note: The generate cookbook command needs the path in the name of the cookbook here to ensure that it generates the cookbook in the cookbooks directory. Without it the cookbook will be generated within the root of the chef repository. If that happens simply have the learner move the newly generated cookbook into the cookbooks directory.

Execute on: **workstation**

Execute from: **anywhere**

GL: Generating a new cookbook



```
$ chef generate cookbook cookbooks/myhaproxy
```

```
Generating cookbook myhaproxy
```

```
- Ensuring correct cookbook content
```

```
Your cookbook is ready. Type `cd .\cookbooks\myhaproxy` to enter it.
```

```
There are several commands you can run to get started locally developing and testing  
your cookbook.
```

```
Why not start by writing an InSpec test? Tests for the default recipe are stored at:  
test/integration/default/default_test.rb
```

```
If you'd prefer to dive right in, the default recipe can be found at:
```

```
recipes/default.rb
```

Generate your new cookbook.

Instructor Note: The generate cookbook command needs the path in the name of the cookbook here to ensure that it generates the cookbook in the cookbooks directory. Without it the cookbook will be generated within the root of the chef repository. If that happens simply have the learner move the newly generated cookbook into the cookbooks directory.

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Creating a dependency in the cookbook

```
~/chef-repo/cookbooks/myhaproxy/metadata.rb
```

```
name 'myhaproxy'  
maintainer 'The Authors'  
maintainer_email 'you@example.com'  
license 'All Rights Reserved'  
description 'Installs/Configures myhaproxy'  
version '0.1.0'  
chef_version '>= 16.0'  
depends 'haproxy', '~> 12.2.3'
```

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.

The ' $\sim>$ ' operator is the pessimistic operator, which in this case means accept cookbook versions 12.2.x through 12.2.xxxxxx, but not including 12.3.X So, 12.2.0, 12.2.1 and 12.2.9999999 would all be acceptable, but 12.3.0 would not. This allows you to assume that anything in the 12.2 range is going to work (accepting the cookbook developer's latest patch fixes) but also assumes that 12.3 or anything higher is not acceptable until it is tested and the dependency in metadata.rb is updated.

Concept



Custom Resources

A recipe can call any recipes or custom resources from a dependency that's defined with 'depends' in the metadata.rb file.

You can call a Custom Resource from any recipe in your wrapper cookbook.

We want to use our custom resources from the haproxy community cookbook within our myhaproxy cookbook. Now that we've added a dependency, we can do this within any recipe in our wrapper cookbook.

GL: Add custom resources to default recipe

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

#
# Cookbook Name:: myhaproxy
# Recipe:: default
#
# Copyright (c) 2020 The Authors, All Rights Reserved.

haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

...
```

First, within the myhaproxy cookbook's default recipe we will use the `haproxy_install` resource to install haproxy from a package (as opposed to from source). How did we know to use 'package' instead of 'source'? We read the haproxy cookbook's `README.md` file.

Next, we use the `haproxy_frontend` resource, giving it a generic name like "http-in". This will be the proxy server that receives our web traffic. Using the "bind" property we can tell it to push its backend to port 80. We also define the name of our backend with the "default_backend" property, which we'll name "server_backend" for now. We will continue writing this recipe later in this module, but first we need to define the backend server IP's we want to forward to our frontend.

GL: Viewing help on the node show subcommand



```
$ 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
```

This new default value for the haproxy members needs to define the information about the webserver node. 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.

Execute on: **workstation**

Execute from: **anywhere**

Demo: Viewing the node's IP address



```
$ knife node show iis_web -a ipaddress
```

```
iis_web:  
ipaddress: 172.31.8.68
```

This method of retrieving the IP address is not useful if you need the external IP address. We'll show you another way in a moment.

You can display the IP address of `iis_web` with the '`-a`' flag and specifying the attribute '`ipaddress`'.

Cloud providers that generate machines for you often assign internal IP addresses, those values may not work properly.

Instructor Note: The IP addresses of the nodes that were used during the creation of this training were based on Amazon Web Services (AWS). The address reported by Ohai is often the private, internal address.

Execute on: **workstation**

Execute from: **anywhere**

Concept



Amazon EC2 Instances

The IP address and host name are unfortunately not how we can address these nodes within our recipes.

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. Our security permissions in the AWS Cloud are not set up to allow communication between nodes on the Private IP address, only on the Public IP address. In production, you could change this to allow communication on the Private IP address, creating a more secure network. See Chef's 'Deploying Infrastructure at Scale' course to learn how to do this using Terraform.

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 information about the node.

GL: Viewing the node's cloud details



```
$ knife node show iis_web -a cloud
```

```
iis_web:  
  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: ec2-54-175-413-24.compute-1.amazonaws.com  
    public_ips: 54.175.46.24  
    public_ipv4: 54.175.46.24
```

You'll need this information for the next task.

If you use 'knife node show' to display the 'cloud' attribute for iis_web, you will see the local, private, and public connection information.

Capture and write down the public hostname and the public ipv4 address of iis_web. You will need this in the recipe you are going to write.

Execute on: **workstation**

Execute from: **anywhere**

GL: Viewing the node's cloud details



```
$ knife node show apache_web -a cloud
```

```
apache_web:
  cloud:
    local_hostname: ip-172-31-57-169.ec2.internal
    local_ipv4: 172.31.57.169
    private_ips: 172.31.57.169
    provider: ec2
    public_hostname: ec2-34-1913-10-17.compute-1.amazonaws.com
    public_ips: 34.196.10.17
    public_ipv4: 34.196.10.17
```

You'll need this information for the next task.

If you use 'knife node show' to display the 'cloud' attribute for apache_web, you will see the local, private, and public connection information.

Capture and write down the public hostname and the public ipv4 address of apache_web. You will need this in the recipe you are going to write.

Execute on: **workstation**

Execute from: **anywhere**

GL: Inserting real node data into server property

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
```

```
...
haproxy_backend 'server_backend' do
  server [
    'ec2-54-175-413-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
    'ec2-34-1913-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
  ]
end

haproxy_service 'haproxy' do
  action [ :enable, :start ]
end
```

Replace the hostname value
and IP address values with
your `iis_web` and
`apache_web` node's public
host name and public IP
address.

We complete the default recipe by declaring the `haproxy_backend` custom resource. It accepts a property called "server", which we define as an array of webserver addresses.

Each address follows the pattern:

'HOSTNAME PUBLIC_IP_ADDRESS:80 maxconn 32'

Make sure there's a comma after the first server (in the slide this is `iis_web`)

Lastly, we add the `haproxy_service` resource, which will automatically accept notifications from other services. Although the default action will be "nothing", we will see it automatically restarting the service when changes to the `haproxy_backend` are made due to the way the custom resource was implemented. This resource needs to be in our resource collection so it can receive notifications from the other custom resources.

Note: The notifications aren't directly written into this recipe. You can learn more about notifications here:

https://docs.chef.io/resource_common.html#notifications

GL: Viewing the complete recipe

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

haproxy_backend 'server_backend' do
  server [
    'ec2-54-175-413-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
    'ec2-34-1913-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
  ]
end

haproxy_service 'haproxy' do
  action [ :enable, :start ]
end
```



The final default recipe for the wrapper cookbook 'myhaproxy' looks like the above but with the hostname value and the ipaddress value of [your iis_web and the apache node](#).

The haproxy action of :reload will run every time chef-client executes, in case the backend list of servers is changed. This isn't ideal, and in a later module we'll use a notification to make this better.

Save your recipe file.

```
haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

haproxy_backend 'server_backend' do
  server [
    'ec2-54-175-413-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
    'ec2-34-1913-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
  ]
]
```

```
end

haproxy_service 'haproxy' do
  action [ :enable, :start ]
end
```

Exercise



Group Lab: Load Balancer

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the `iis_web` and `apache_web` nodes
 - Create myhaproxy Policyfile
 - Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
 - Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Adding a load balancer will allow us to better grow our infrastructure.



Concept

Policyfile.rb and the Policyfile.lock.json

Now that we have our myhaproxy cookbook in our chef-repo, we can create our Policyfile.rb and then generate our Policyfile.lock.json as we discussed in previous modules.

This time we'll name our Policyfile **myhaproxy**.

GL: Generate the Policyfile and name it myhaproxy

```
> cd ~/chef-repo  
> chef generate policyfile policyfiles/myhaproxy
```

```
Recipe: code_generator::policyfile  
* template[/home/chef/chef-repo/policyfiles/myhaproxy.rb] action create  
  - create new file /home/chef/chef-repo/policyfiles/myhaproxy.rb  
  - update content in file /home/chef/chef-repo/policyfiles/myhaproxy.rb from  
none to 5d842c  
  (diff output suppressed by config
```

‘cd’ command

Execute on: **workstation**

Execute from: **anywhere**

‘chef’ command

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Verify that the Policyfile exists

```
> ls -l policyfiles
```

```
total 24
-rw-rw-r-- 1 chef chef 1018 Oct  6 17:57 apache.lock.json
-rw-rw-r-- 1 chef chef  590 Oct  6 17:46 apache.rb
-rw-rw-r-- 1 chef chef 2463 Oct  7 17:29 company_web.lock.json
-rw-rw-r-- 1 chef chef  790 Oct  7 16:42 company_web.rb
-rw-rw-r-- 1 chef chef  637 Oct 10 17:57 myhaproxy.rb
-rw-r--r-- 1 chef chef  596 Sep 14 18:50 README.md
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Edit the new myhaproxy.rb Policyfile

~/chef-repo/policyfiles/myhaproxy.rb

```
#...skipping for brevity...
# https://docs.chef.io/policyfile.html
# A name that describes what the system you're building with Chef does.
name 'myhaproxy'

# Where to find external cookbooks: Update the 'cookbook' section
default_source :supermarket

# run_list: chef-client will run these recipes in the order specified.
run_list 'myhaproxy::default'

# Specify a custom source for a single cookbook:
cookbook 'myhaproxy', path: '../cookbooks/myhaproxy'
```

Update the custom source path to the cookbook section, as highlighted in the slide.

Exercise



Group Lab: Load Balancer

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the `iis_web` and `apache_web` nodes
- ✓ Create myhaproxy Policyfile
 - Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
 - Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Adding a load balancer will allow us to better grow our infrastructure.

GL: Generate the myhaproxy.lock.json

```
> chef install policyfiles/myhaproxy.rb

Building policy myhaproxy
Expanded run list: recipe[myhaproxy::default]
Caching Cookbooks...
Installing myhaproxy >= 0.0.0 from path
Installing haproxy    12.2.3
Installing yum-epel   4.5.0

Lockfile written to /home/chef/chef-repo/policyfiles/myhaproxy.lock.json
Policy revision id: a25e4353a98e11b8119739e55c58bba6a4a139993cec7ac2aabba3027f787ad0
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Verify that the myhaproxy.lock.json exists

```
> ls -l policyfiles
```

```
total 28
-rw-rw-r-- 1 chef chef 1018 Oct  6 17:57 apache.lock.json
-rw-rw-r-- 1 chef chef  590 Oct  6 17:46 apache.rb
-rw-rw-r-- 1 chef chef 2463 Oct  7 17:29 company_web.lock.json
-rw-rw-r-- 1 chef chef  790 Oct  7 16:42 company_web.rb
-rw-rw-r-- 1 chef chef 2419 Oct 11 16:28 myhaproxy.lock.json
-rw-rw-r-- 1 chef chef  621 Oct 11 16:27 myhaproxy.rb
-rw-r--r-- 1 chef chef  596 Sep 14 18:50 README.md
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Push the myhaproxy.lock.json to Chef Infra Server

```
> chef push prod policyfiles/myhaproxy.lock.json
```

```
Uploading policy myhaproxy (a25e4353a9) to policy group prod
Uploaded haproxy 12.2.3 (e7d1c19b)
Uploaded myhaproxy 0.1.0 (e6a1b57b)
Uploaded yum-epel 4.5.0 (dad6a6c0)
```

GL: Verify the myhaproxy policy exists

```
> chef show-policy
```

```
apache
=====
* prod: e3db31b3bb

company_web
=====

* prod: a593fc2fe2

myhaproxy
=====

* prod: a25e4353a
```

Here we can see that the **myhaproxy** policy has been uploaded to Chef Infra Server and is in the **prod** policy group.

Also notice the policy name that was derived from the contents of the **myhaproxy.lock.json**.

Execute on: **workstation**
Execute from: **anywhere**

Exercise



Group Lab: Load Balancer

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the `iis_web` and `apache_web` nodes
- ✓ Create myhaproxy Policyfile
- ✓ Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Adding a load balancer will allow us to better grow our infrastructure.

GL: Bootstrap a new Linux node



```
$ knife bootstrap IPADDRESS -U chef -P PWD --sudo -N lb  
--policy-name myhaproxy --policy-group prod  
[34.196.50.77] Starting Chef Infra Client, version 17.3.48  
[34.196.50.77] Using policy 'myhaproxy' at revision  
'ff07bcb7f5f57ac1cd6c2eb3f7c7785f8d531?c035725e3c54fb0d84292c6670'  
[34.196.50.77] resolving cookbooks for run list: ["myhaproxy::default@1.0.0 (60)"]  
[34.196.50.77] Synchronizing Cookbooks:  
[34.196.50.77]  
[34.196.50.77] - haproxy (12.2.3)  
[34.196.50.77] - myhaproxy (0.1.0)  
[34.196.50.77] - yum-epel (4.1.4)  
...  
Running handlers:  
[34.196.50.77]  
[34.196.50.77] Running handlers complete [34.196.50.77] Chef Infra Client finished, 16/29  
resources updated in 29 seconds
```

node name

policy_name

policy_group

We are bootstrapping a new fresh Linux node here.

Enter 'Y' when prompted to continue.

Execute on: **workstation**

Execute from: **anywhere**

GL: Validate the run list has been set



```
$ knife node show lb
```

```
Node Name: lb
Policy Name: myhaproxy
Policy Group: prod
FQDN: ip-172-31-22-163.ec2.internal
IP: 34.196.50.77
Run List: recipe[myhaproxy::default]
Recipes: myhaproxy::default, yum-epel::default
Platform: centos 7.6.1810
Tags:
```

GL: Verify policy applied to lb

View Information associated with node.

Chef Infra Servers > Organizations > Nodes > lb

lb

NODE INFORMATION		METADATA	
Environment	prod	Chef Server	cheftronin
Policy Group	prod	Chef Organization	student01
Policy Name	myhaproxy		

Details Run list Attributes

Environment

prod

Lab: Test the Load Balancer

The screenshots show the following information:

Platform	Hostname	Memory (kB)	CPU Mhz
Windows	WIN-8694LT97S51	8388208	2400
Windows	WIN-8694LT97S51	8388208	2400
CentOS	ip-172-31-29-209	604192	1795.672

Progress Chef © 2022 Progress Software Corporation. All rights reserved. All rights reserved.

11-45

Point a web browser to the URL of your load balancer/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.

Note: The haproxy shared cache could take a few minutes before it refreshes, so you may not see the "Chef Welcomes You!" page update immediately. You may need to wait 20 or more seconds between refreshes.

To confirm haproxy is configured correctly, you could check the content of /etc/haproxy/haproxy.cfg on the load balancer node and confirm the servers are included in the pool. If you are curious, you could configure caching using the haproxy_cache custom resource: https://github.com/sous-chefs/haproxy/blob/master/documentation/haproxy_cache.md

Lab: Test the Load Balancer

The image shows three separate browser windows side-by-side, each displaying a different server configuration:

- Top Window:** Shows a CentOS server. The page title is "Chef Welcomes You!". The system details are:
 - PLATFORM: centos
 - HOSTNAME: ip-172-31-29-209
 - MEMORY: 604192kB
 - CPU Mhz: 1795.672
- Middle Window:** Shows a Windows server. The page title is "Chef Welcomes You!". The system details are:
 - PLATFORM: windows
 - HOSTNAME: WIN-8694LT97S51
 - MEMORY: 8388208kB
 - CPU Mhz: 2400
- Bottom Window:** Shows another CentOS server. The page title is "Chef Welcomes You!". The system details are:
 - PLATFORM: centos
 - HOSTNAME: ip-172-31-29-209
 - MEMORY: 604192kB
 - CPU Mhz: 1795.672

Progress Chef logo is at the bottom left. A small copyright notice at the bottom right reads: © 2022 Progress Software Corporation. All rights reserved. All rights reserved.

11-46

Point a web browser to the URL of your load balancer/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.

Exercise



Group Lab: Load Balancer

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the `iis_web` and `apache_web` nodes
- ✓ Create myhaproxy Policyfile
- ✓ Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- ✓ Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Adding a load balancer will allow us to better grow our infrastructure.

Review



Review Questions

1. What are the benefits of the Chef Super Market? And what are the drawbacks?
2. When would you use a wrapper cookbook?
3. When might you decide to not wrap the cookbook?

1. Benefits: You can use existing cookbooks instead of writing all of them.
Drawbacks: The cookbooks you find in the Supermarket may not be built or designed for your platform. They may not take into special consideration your needs and requirements or they may no longer be actively maintained. There is also no guarantee as to their quality.
2. When you want to use a community cookbook (or any other cookbook) but you need to override some attributes or functionality. You can use a wrapper cookbook instead of modifying the community cookbook or other original cookbook.
3. Possible answers: Whenever you're using a cookbook that isn't directly owned by your team because it means you have no control over changes. Whenever you're using a cookbook that might act differently in different scenarios or setups.



Questions?

Q&A

What questions can we help you answer?

- Chef Supermarket
- Wrapper Cookbooks
- Node Attributes
- knife ssh

What questions can we help you answer?

In general or about specifically about Chef Supermarket, wrapper cookbooks, node attributes or the 'knife ssh' command.

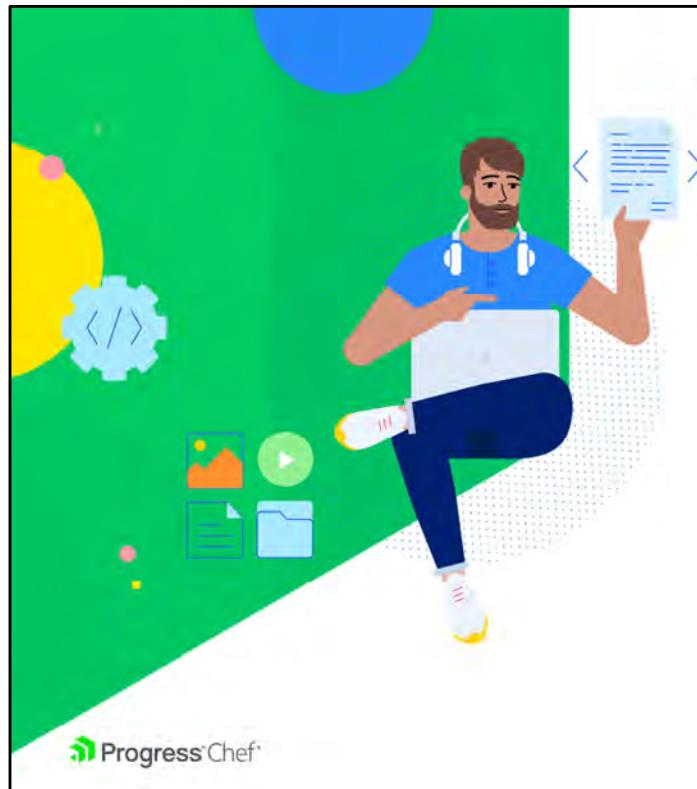




Using policy_name to Define a Role for Nodes

Using policy_name instead of role objects





Objectives

After completing this brief module, you should be able to

- Explain how policy_name replaces the legacy Role object
- Use knife search to display nodes

© 2022 Progress International Corporation. All rights reserved.

12- 2

In this brief module you will ensure your web nodes have the same policy_name to better describe them and so you can configure them in a similar manner.



Concept

policy_name

Chef users sometimes used **role** objects to describe a run list of recipes that are executed on the node and to use for node searches.

Nowadays we use the policy_name for those purposes.

For example, all nodes that possess the **company_web** policy name would be configured in a similar or identical manner.

<https://docs.chef.io/roles/>

<https://docs.chef.io/roles/>



Concept

policy_name

When you assign a common policy_name to a group of nodes, each node will receive the same cookbooks (assuming they also utilize the same policy_group).

When these nodes perform a chef-client run, they utilize recipes specified in the Policyfile run list.

<https://docs.chef.io/roles/>

Exercise



GL: Verify that all web nodes use the same policy_name

- Confirm your iis_web node has the same policy_name (company_web) as the apache_web node
- Use `knife search` to list all or specific nodes

Give your nodes a policy_name to better describe them and so we can configure them in a similar manner.

This is particularly powerful because we will no longer have to manage each of these nodes individually. Instead we can make changes to the policy file named **company_web** (and its cookbooks) and all of the nodes that have this policy file named **company_web** will update accordingly.



Concept

company_web Policy Name

In a previous module we set the `iis_web` node to use the `company_web` policy name with this command:

```
knife node policy set iis_web prod company_web
```

Now, when we update the cookbooks that are associated to the `company_web` policy name, the nodes that use the `company_web` policy name will be updated as well

We will also be able to easily search for all nodes that have the `company_web` policy name.

Note: In the next module you will learn more about using search.

GL: List your nodes

```
$ knife node list
```

```
apache_web  
iis_web  
lb
```

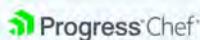
GL: Searching for all nodes with knife

```
$ knife search node *:*
```

```
Node Name: lb
Policy Name: myhaproxy
Policy Group: prod
FQDN: ip-172-31-72-132.ec2.internal
IP: 3.237.240.176
Run List: recipe[myhaproxy::default]
Recipes: myhaproxy::default, yum-epel::default
Platform: centos 7.6.1810
Tags:

Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-77-52.ec2.internal
IP: 44.205.9.57
Run List: recipe[apache::default]
Recipes: apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFgCUFHDCP
IP: 107.20.24.200
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```



Here we can see the policy_name and policy_group to which each node is assigned

Execute on: **workstation**

Execute from: **anywhere**

GL: Searching for nodes with a specific policy_name

```
$ knife search node policy_name:company_web
```

```
Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP
IP: 107.20.24.200
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:

Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-77-52.ec2.internal
IP: 44.205.9.57
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
```

Execute on: **workstation**

Execute from: **anywhere**

Exercise



GL: Verify that all web nodes use the same policy_name

- ✓ Confirm your iis_web node has the same policy_name (company_web) as the apache_web node
- ✓ Use `knife search` to list all or specific nodes

Give your nodes a policy_name to better describe them and so we can configure them in a similar manner.

This is particularly powerful because we will no longer have to manage each of these nodes individually. Instead we can make changes to the policy file named **company_web** (and its cookbooks) and all of the nodes that have this policy file named **company_web** will update accordingly.

Concept



company_web Policy Name

In the next module you will learn more about using search and you will put the company_web policy name to work for you.



Questions?

Q&A

What questions can we help you answer?

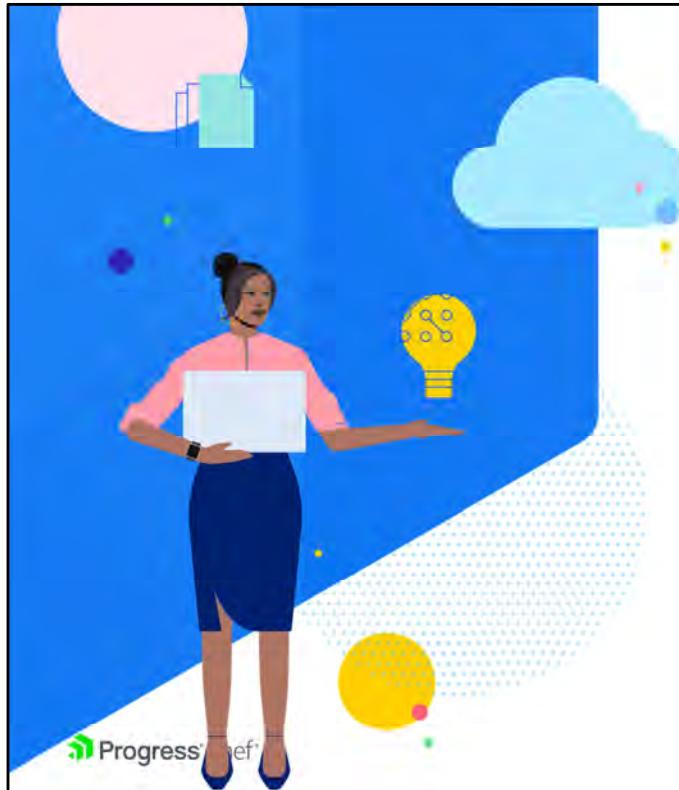




Search

Update a Cookbook to Dynamically Use
Nodes with the company_web policy name





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 String and Array dynamically
- Test that your load balancer is still balancing traffic

©2022 Progress Software Corporation. All rights reserved. "Progress" is a registered trademark of Progress Software Corporation.

13- 2

Concept



Search

To add new servers as load balancer members, we would need to bootstrap a new web server and then update our load balancer's myhaproxy cookbook recipe.

That seems inefficient to have to update a cookbook recipe manually.

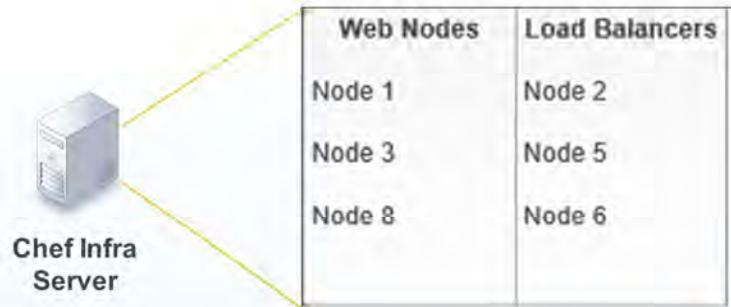
A more ideal solution would be for the recipe to instead discover all of the web servers within our organization and automatically add them to a list of available members for our load balancer.

The Chef Infra Server and Search

Chef Infra Server maintains a representation of all the nodes within our infrastructure that can be searched on.

Search is a service discovery tool that allows us to query the Chef Infra Server.

https://docs.chef.io/chef_search/
https://docs.chef.io/chef_search/#search-indexes



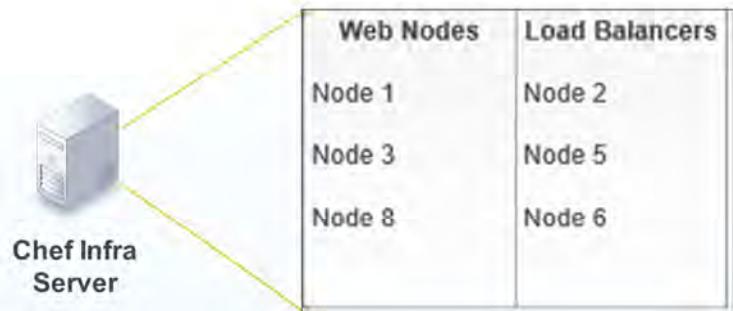
The Chef Infra Server maintains a representation of all the nodes within an infrastructure and provides a way for us to discover these systems through Search.

Search is a service discovery tool that allows us to query the Chef Infra Server across a few indexes. One such index is on our nodes.

https://docs.chef.io/chef_search/
https://docs.chef.io/chef_search/#search-indexes

The Chef Infra Server and Search

We can ask the Chef Infra Server to return all the nodes or a subset of nodes based on the query syntax that we provide it through `knife search` or within our recipes through `search`.



We can ask the Chef Infra 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.

Search Syntax

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 Infra Server and search_pattern defines what will be searched for

The indexable objects on the Chef Infra Server include role, node, client, environment, and data bag (which is an index that you can create).

Search Criteria

We may use wildcards within search so a search criteria that we could use is: `"*:*`

However, querying and returning every node is not what we need to solve our current problem.



Scenario: We want only to return a subset of our nodes... only the nodes that are web servers.

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.

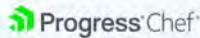
Demo: View information for all nodes

```
> knife search node *:*
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-57-169.ec2.internal
IP: 34.196.104.17
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUCUFHDCP.ec2.internal
IP: 34.195.38.226
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:

Node Name: lb
Policy Name: myhaproxy
Policy Group: prod
FQDN: ip-172-31-22-163.ec2.internal
IP: 34.196.50.77
Run List: recipe[myhaproxy::default]
Recipes: myhaproxy::default, haproxy::manual, haproxy::install_package
Platform: centos 7.6.1810
```

©2022 Progress Software Corporation and/or its subsidiaries and/or affiliates. All rights reserved.13- 8

knife search node *:*

will return see all nodes.

However, querying and returning every node is not exactly what we need to solve our current problem. In our scenario we want only to return a subset of our nodes (only the nodes that are webservers) and also we only want the IP address of those nodes, not any the other attributes returned.

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, and only the info we want for those nodes.

Execute on: **workstation**

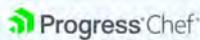
Execute from: **anywhere**

Demo: View information for all web nodes

```
> knife search node policy_name:company_web
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-57-169.ec2.internal
IP: 34.196.104.17
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 34.195.38.226
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```

©2022 Progress Software Corporation and/or its subsidiaries and/or affiliates. All rights reserved.13- 9

We can query the Chef Infra Server for specific nodes such as those with the assigned `policy_name` of `company_web`.

Execute on: **workstation**

Execute from: **anywhere**

Demo: Return public information for servers

```
> knife search node policy_name:company_web -a cloud
```

```
apache_web:  
cloud:  
  local_hostname: ip-172-31-57-169.ec2.internal  
  local_ipv4: 172.31.57.169  
  local_ipv4_addrs: 172.31.57.169  
  provider: ec2  
  public_hostname: ec2-34-1915-104-17.compute-1.amazonaws.com  
  public_ipv4: 34.196.104.17  
  public_ipv4_addrs: 34.196.104.17  
  
iis_web:  
cloud:  
  local_hostname: ip-172-31-62-51.ec2.internal  
  local_ipv4: 172.31.62.51  
  local_ipv4_addrs: 172.31.62.51  
  provider: ec2  
  public_hostname: ec2-34-195-38-226.compute-1.amazonaws.com  
  public_ipv4: 34.195.38.226  
  public_ipv4_addrs: 34.195.38.226
```

The ‘-a’ flag allows you to specify a particular attribute from the nodes returned in the search.

Execute on: **workstation**

Execute from: **anywhere**

Demo: Return public hostname for servers

```
> knife search node policy_name:company_web -a cloud.public_hostname

2 items found

apache_web:
  cloud.public_hostname: ec2-34-1915-104-17.compute-1.amazonaws.com

iis_web:
  cloud.public_hostname: ec2-34-195-38-226.compute-1.amazonaws.com
```

You can also drill deeper down into the returned attributes, in this case returning just the `public_hostname`.

Execute on: **workstation**

Execute from: **anywhere**

Search syntax within a recipe

```
web_nodes = search('node', 'policy_name:company_web')
```

creates and names a variable

assigns the value of the operation on the right into the variable on the left

the index or items to search

the search criteria - key:value

invokes the search method

The search syntax within a recipe differs from the search syntax when using 'knife search' from the command line.

```
web_nodes = search('node','policy_name:company_web')
```

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 to search through on the Chef Infra Server.

Possible values are: node, client, policy_name, and policy_group while 'role', and 'environment' still exist for backward compatibility they are not strictly valid.

The second argument is a string or variable that contains the search criteria to scope the results. This is using the notation 'key:value', in this case "policy_name:company_web".

The result of the search method is stored in a local variable that is named 'web_nodes'. Variables within Ruby are created immediately when you assign them.

Hard coding example

```
haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

haproxy_backend 'server_backend' do
  server [
    'ec2-54-175-415-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
    'ec2-34-1915-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
  ]
end

haproxy_service 'haproxy'
```

Previously, we had been hard coding the hostname and ipaddress values in our wrapped myhaproxy recipe. We can request these values from the Chef Infra Server through the `knife node show` command. The hostname and ipaddress values are captured by Ohai and sent to the Chef Infra Server. On the Chef Infra Server we can query those values when we ask about a 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'.

Exercise



GL: Dynamic Web Load Balancer

- Update the myhaproxy cookbook to dynamically use nodes with the company_web policy_name.
- Update the major version of the myhaproxy cookbook
- Upload the Cookbook
- Run chef-client on the load balancer node
- Verify the load balancer node relays requests to both web nodes

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!

In this section we'll update the load balancer's myhaproxy cookbook to dynamically use nodes with the company_web policy.

GL: Remove the hard coded servers

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
```

```
...
haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

haproxy_backend 'server_backend' do
  server [
    'ec2-54-175-415-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
    'ec2-34-1915-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
  ]
end

...
```

GL: Add the search criteria

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
```

```
...
haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

web_nodes = search('node', 'policy_name:company_web')
```

Note: We will provide the final recipe in a moment.

Replace it with an updated recipe that searches for all nodes that have the company_web policy name defined.

The search method's first parameter is asking the Chef Infra Server to look at all the nodes within our organization.

The search method's second parameter is asking the Chef Infra Server to only return the nodes that have been assigned the role company_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.

GL: Create an array to store the servers

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
```

```
...  
  
web_nodes = search('node', 'policy_name:company_web')  
  
server_array = []  
  
haproxy_backend 'server_backend' do  
  server server_array  
end  
  
...
```

Unfortunately we cannot simply assign our array of web nodes into the haproxy's members attributes because it needs a string that contains the webserver's "PUBLIC_HOSTNAME" and "PUBLIC_IPADDRESS". We will need to convert each of the web node objects into a structure that the haproxy_backend custom resource expects.

First we create an empty array and assign that empty array into a local variable named `server_array`. `server_array` is an array that we will populate with hashes that we will create. Then we will pass that array to the "server" property of the haproxy_backend resource. The community cookbook haproxy is expecting to receive an array, passed in through the custom resource variable named 'server'.
server_array -> server -> haproxy cookbook

GL: Create a loop to fill the array

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
```

```
...
web_nodes = search('node', 'policy_name:company_web')

server_array = []

web_nodes.each do |one_node|
  one_server = ""
  # TODO: Populate the array with each webserver's hostname and ipaddress
end

haproxy_backend 'server_backend' do
  server server_array
end

...
```

We need to loop through the array of all the web nodes stored in `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 performed on every item in the array. In our case we want to take each of the node objects and grab the public hostname and ipaddress returned from the search. This information comes from the node objects stored on the Chef Infra Server.

Every member of the array is visited and every member of the array runs through the block of code.

GL: Add node information to array

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

...
web_nodes = search('node', 'policy_name:company_web')

server_array = []

web_nodes.each do |one_node|
  one_server = "#{one_node['cloud']['public_hostname']}:#{one_node['cloud']['public_ipv4']}:80 maxconn
32"
  server_array.push(one_server)
end

haproxy_backend 'server_backend' do
  server server_array
end

...
```

Between the pipes we see a local variable that we are defining that exists only in the block `one_node`. This local variable, `one_node`, is a name we came up with to refer to each node in our array of `web_nodes`. When inside the block of code `web_nodes.each` (this is called a "loop") it is referred to as `one_node`. Inside the block the first thing that is created is another local variable named "one_server" which is assigned a string that contains the webserver's public hostname and ipaddress. Then the local variable "one_server" is pushed into the array named 'server_array'. This adds the one_node data (hostname, IP address and the number of allowable max connections) to the end of the array. When we are done looping through every one_node, server_array will contain a list of all these string values, one for each node object returned from the search.

GL: The updated recipe

```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb
```

```
haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end

web_nodes = search('node','policy_name:company_web')

server_array = []

web_nodes.each do |one_node|
  one_server = "#{one_node['cloud']['public_hostname']}:#{{one_node['cloud']['public_ipv4']}":80 maxconn 32"
  server_array.push(one_server)
end

haproxy_backend 'server_backend' do
  server server_array
end

haproxy_service 'haproxy' do
  action [ :enable, :start ]
end
```



```
haproxy_install 'package'
```

```
haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'server_backend'
end
```

```
web_nodes = search('node','policy_name:company_web')
```

```
server_array = []
```

```
web_nodes.each do |one_node|
  one_server = "#{one_node['cloud']['public_hostname']}:#{{one_node['cloud']['public_ipv4']}":80 maxconn 32"
  server_array.push(one_server)
end
```

```
haproxy_backend 'server_backend' do
  server server_array
end
```

```
haproxy_service 'haproxy' do
  action [ :enable, :start ]
```

end

Exercise



GL: Dynamic Web Load Balancer

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company_web policy_name.
 - Update the major version of the myhaproxy cookbook
 - Upload the Cookbook
 - Run chef-client on the load balancer node
 - Verify the load balancer node relays requests to both web nodes

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!

GL: Updating the cookbook's version number

```
~/chef-repo/cookbooks/myhaproxy/metadata.rb
```

```
name 'myhaproxy'  
maintainer 'The Authors'  
maintainer_email 'you@example.com'  
license 'All Rights Reserved'  
description 'Installs/Configures myhaproxy'  
version '1.0.0'  
chef_version '>= 16.0'  
depends 'haproxy', '~> 12.2.3'
```

Update the version to the next major release. We're setting the version number to 1.0.0.

Exercise



GL: Dynamic Web Load Balancer

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company_web policy_name.
- ✓ Update the major version of the myhaproxy cookbook
 - Upload the Cookbook
 - Run chef-client on the load balancer node
 - Verify the load balancer node relays requests to both web nodes

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!

GL: Ensure you are in the chef-repo

```
> cd ~/chef-repo
```

Execute on: **workstation**
Execute from: **anywhere**

GL: Update the Policyfile

```
> chef update policyfiles/myhaproxy.rb

Building policy myhaproxy
Expanded run list: recipe[myhaproxy::default]
Caching Cookbooks...
Installing myhaproxy >= 0.0.0 from path
Using     haproxy   12.2.3
Using     yum-epel  4.5.0

Lockfile written to C:/Users/Administrator/chef-
repo/policyfiles/myhaproxy.lock.json
Policy revision id:
abf0cb66a1ee523d3dc7283236b11f18c3f6837edd7236fe0631b0bfe094ae
```

If you ever need to update and re-push your policyfile, use the `chef update` command. For example:

```
chef update myhaproxy.rb
```

```
chef push prod myhaproxy.lock.json
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Push the Policyfile to Chef Infra Server

```
> chef push prod policyfiles/myhaproxy.lock.json
```

```
Uploading policy myhaproxy (abf0cb66a1) to policy group prod
Using    haproxy    12.2.3 (e7d1c19b)
Using    yum-epel   4.5.0  (dad6a6c0)
Uploaded myhaproxy 1.0.0  (82f1eb10)
```

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Converge the load balancer node

```
$ knife ssh 'name:lb' -x chef -P PASSWORD 'sudo chef-client'

ec2-35-170-33-199.compute-1.amazonaws.com Starting Chef Infra Client, version 17.3.48
ec2-35-170-33-199.compute-1.amazonaws.com Using policy 'myhaproxy' at revision
'9da82055ea2c960cd680d131de8e92ba773703538575e4e429a52ca13f01fbba'
ec2-35-170-33-199.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy::default@1.0.0 (e9a41c2)"]
ec2-35-170-33-199.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-35-170-33-199.compute-1.amazonaws.com   - build-essential (8.2.1)
ec2-35-170-33-199.compute-1.amazonaws.com   - haproxy (12.2.3)
ec2-35-170-33-199.compute-1.amazonaws.com   - myhaproxy (1.0.0)
...
...
ec2-35-170-33-199.compute-1.amazonaws.com Chef Infra Client finished, 6/26 resources
updated in 05 seconds
```

Converge the node 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 being updated with a new entry that contains the information of the second member (apache_web).

Execute on: **workstation**

Execute from: **anywhere**

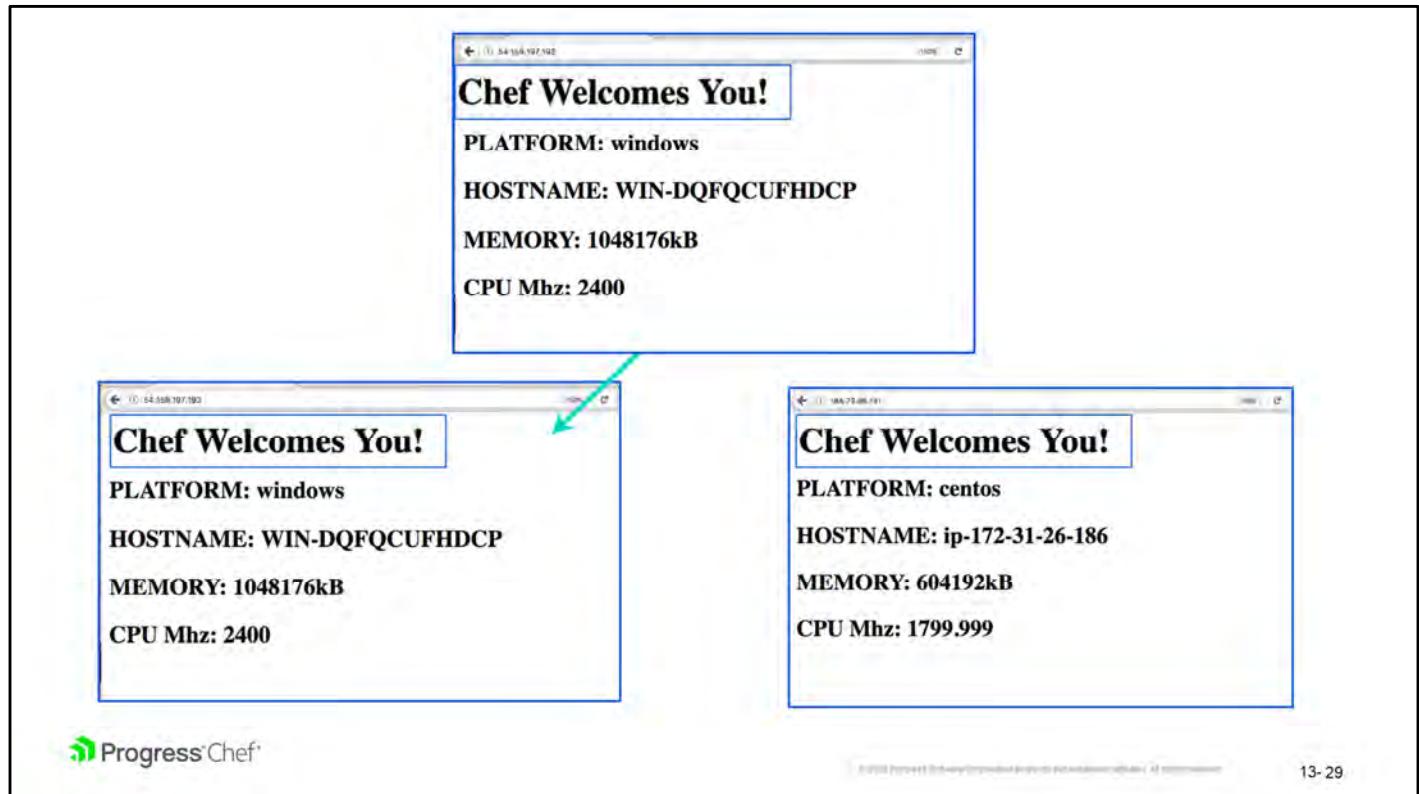
Exercise



GL: Dynamic Web Load Balancer

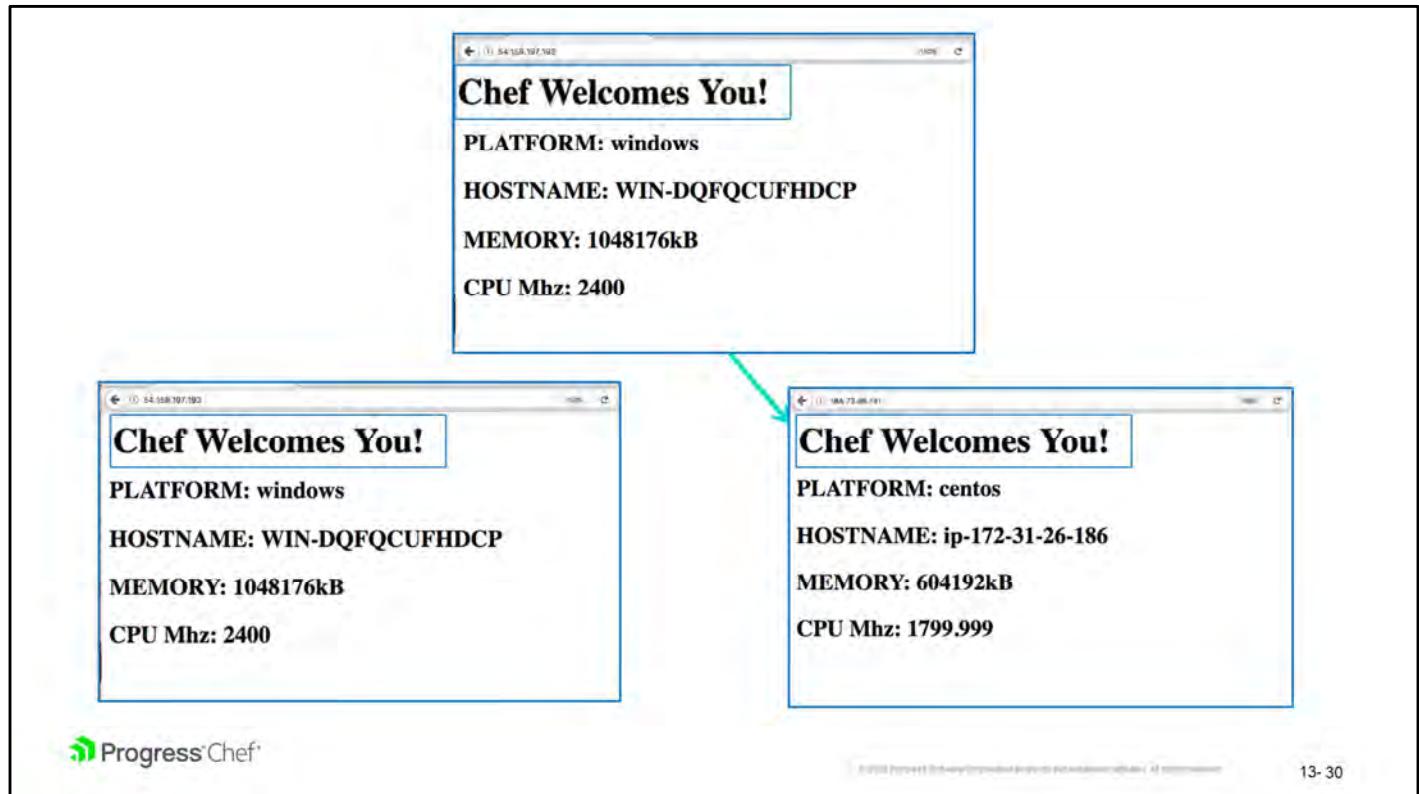
- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company_web policy_name.
- ✓ Update the major version of the myhaproxy cookbook
- ✓ Upload the Cookbook
- ✓ Run chef-client on the load balancer node
 - Verify the load balancer node relays requests to both web nodes

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!



Point a web browser to your load balancer and refresh a couple times.

Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.



Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.

Let's test that our code really works

To verify that our code is working, let's remove the load balancer configuration file, forcing our code to run.

GL: Delete the haproxy.cfg file

```
$ knife ssh 'name:lb' -x chef -P PASSWORD 'sudo rm  
/etc/haproxy/haproxy.cfg'
```

knife ssh 'name:lb' -x chef -P PASSWORD "sudo rm /etc/haproxy/haproxy.cfg"

Execute on: **workstation**

Execute from: **anywhere**

GL: Converge the load balancer

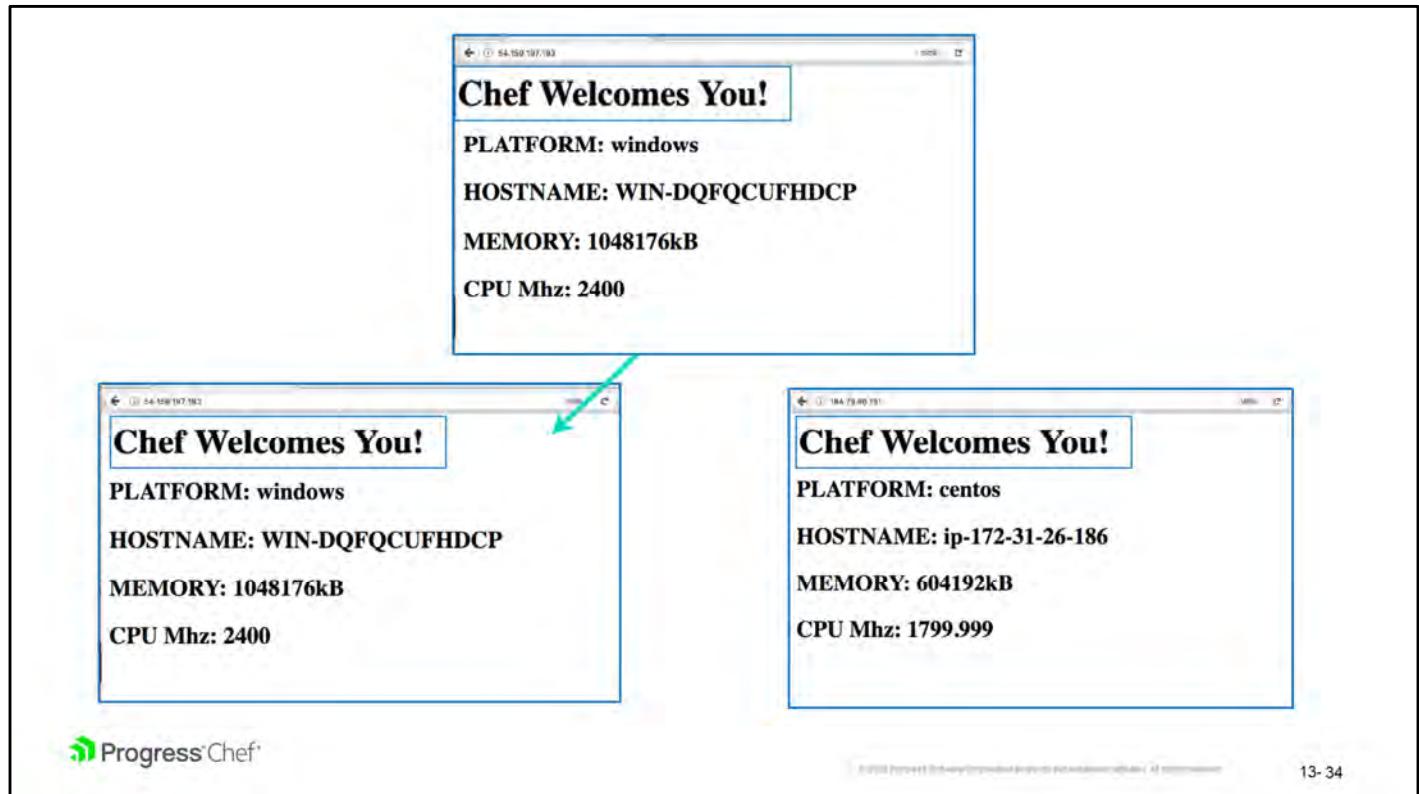
```
$ knife ssh 'name:lb' -x chef -P PASSWORD 'sudo chef-client'

ec2-34-1915-50-77.compute-1.amazonaws.com Starting Chef Infra Client, version 17.3.48
...
ec2-34-1915-50-77.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy::default@1.0.0 (e9a41c2)"]
ec2-34-1915-50-77.compute-1.amazonaws.com Synchronizing Cookbooks:
...
ec2-34-1915-50-77.compute-1.amazonaws.com * template[/etc/haproxy/haproxy.cfg] action
create
  ec2-34-1915-50-77.compute-1.amazonaws.com      - create new file /etc/haproxy/haproxy.cfg
  ec2-34-1915-50-77.compute-1.amazonaws.com      - update content in file
/etc/haproxy/haproxy.cfg from none to 4334de
  ec2-34-1915-50-77.compute-1.amazonaws.com      - suppressed sensitive resource
  ec2-34-1915-50-77.compute-1.amazonaws.com      - change mode from '' to '0644'
  ec2-34-1915-50-77.compute-1.amazonaws.com      - change owner from '' to 'haproxy'
  ec2-34-1915-50-77.compute-1.amazonaws.com      - change group from '' to 'haproxy'
...
...
```

Run 'sudo chef-client' on your lb node to apply the 'myhaproxy' cookbook.

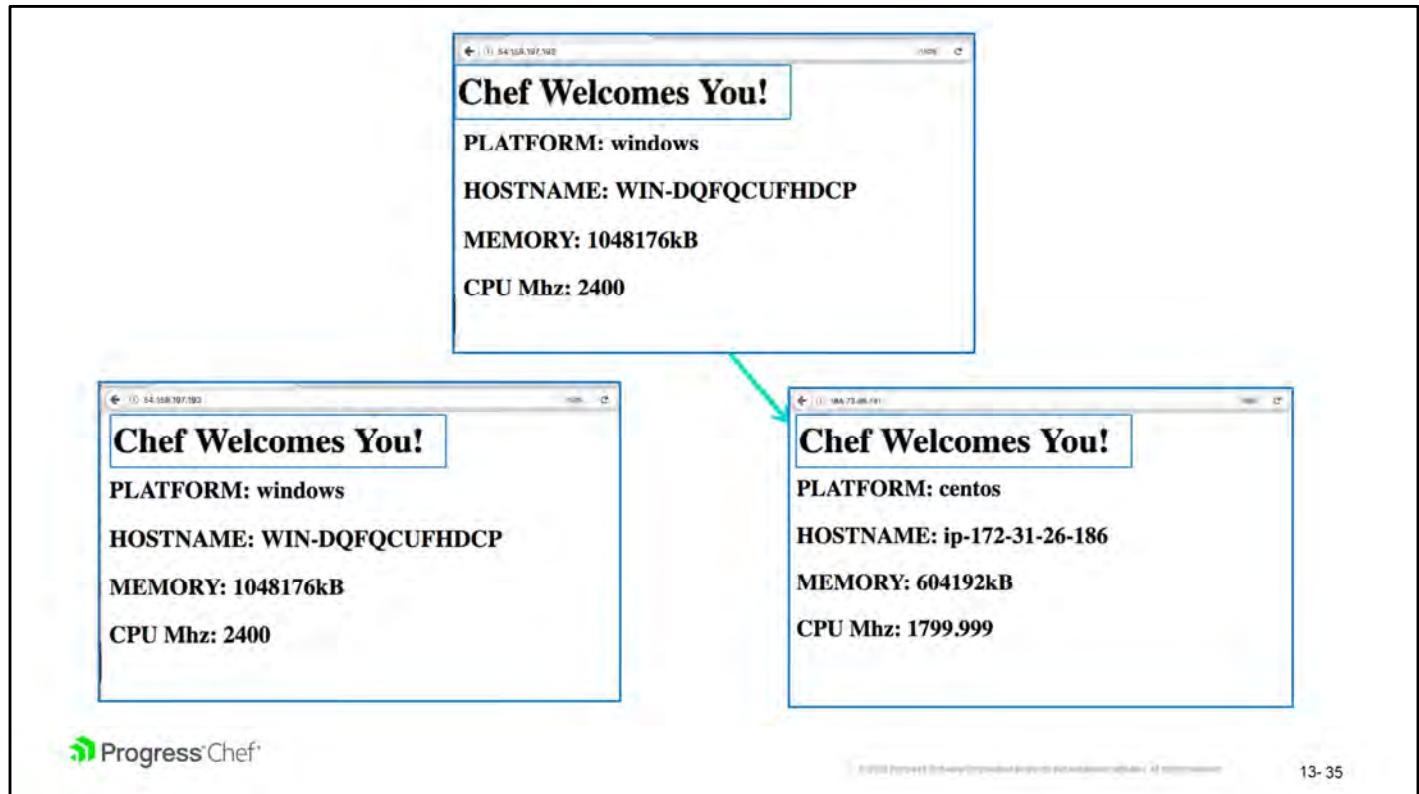
Execute on: **workstation**

Execute from: **anywhere**



Point a web browser to your load balancer and refresh a couple times.

Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.



Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.

Exercise



GL: Dynamic Web Load Balancer

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company_web policy_name.
- ✓ Update the major version of the myhaproxy cookbook
- ✓ Upload the Cookbook
- ✓ Run chef-client on the load balancer node
- ✓ Verify the load balancer node relays requests to both web nodes

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!



Review

Review Questions

1. What is the great advantage of using the following dynamic search in the load balancer's default.rb?

```
web_nodes = search('node','policy_name:company_web')

server_array = []

web_nodes.each do |one_node|
  one_server = "#{one_node['cloud']['public_hostname']}#{one_node['cloud']['public_ipv4']}:80 maxconn 32"
  server_array.push(one_server)
end
```

1. You would not need to hard code the web nodes' hostnames and IP addresses each time you need to add a web node.



Review

Review Questions

2. What is the key item that tells the load balancer how to find the web servers it's supposed to balance?

```
web_nodes = search('node','policy_name:company_web')
```

```
server_array = []
```

```
web_nodes.each do |one_node|
  one_server = "#{one_node['cloud']['public_hostname']}#{one_node['cloud']['public_ipv4']}:80 maxconn 32"
  server_array.push(one_server)
end
```

2. The policy name `company_web` shown here:

```
web_nodes = search('node','policy_name:company_web')
```



Questions?

Q&A

What questions can we help you answer?

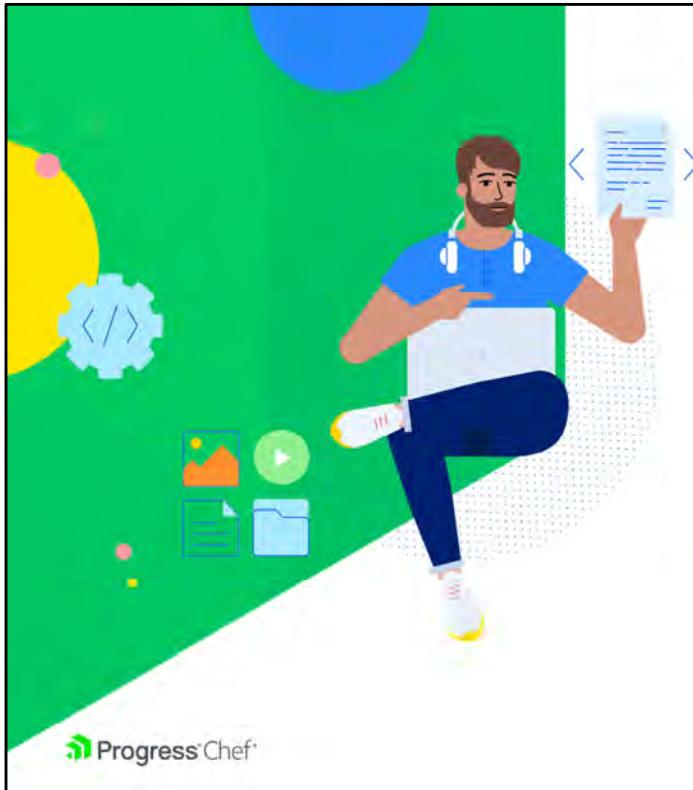




Running chef-client as a Service

Creating the mychef-client cookbook





Lesson Objectives

After completing this module, you should be able to:

- Generate a mychef-client cookbook
- Update the company_web policyfile to include the mychef-client cookbook
- Update and push the company_web Policyfile to Chef Infra Server
- Apply the updated Policyfile to our web nodes
- Run chef-client as a service/task on our web nodes

© 2022 Progress Software Corporation and/or its affiliates. All rights reserved.

14- 2

In this module we are setting up chef-client to run as a service on our Linux web node and a task on our Windows web node.



Concept

Step Back: How is chef-client Configured?

- How can I run chef-client as a service or Windows task?
- Where can I configure logging?
- How does chef-client know what Chef Server to connect to?
- How does chef-client authenticate with the Chef Server?
- How do I configure where chef-client caches?

Demo



Demo: View How chef-client is Configured

In this section you will view how chef-client is configured.

Demo: View chef-client config directory

```
$ knife ssh "os:linux" -x USER -P PWD "ls -F /etc/chef"
```

```
ec2-3-237-240-176.compute-1.amazonaws.com accepted_licenses/ client.pem      first-
boot.json  trusted_certs/
ec2-3-237-240-176.compute-1.amazonaws.com chef_guid        client.rb  ohai/
ec2-44-205-9-57.compute-1.amazonaws.com  accepted_licenses/ client.pem      first-
boot.json  trusted_certs/
ec2-44-205-9-57.compute-1.amazonaws.com  chef_guid        client.rb  ohai/
```

chef-client looks for its configuration information in the directory '/etc/chef' by default – although this is configurable itself!

This directory contains not only its own configuration file (which we'll look at shortly), but also its key to authenticate with the Chef Server, an Ohai plugins directory and a first-boot.json file. The first-boot.json file is generated from the workstation as part of the initial knife bootstrap subcommand, and contains the initial runlist that chef-client should run after Chef has been installed, and before it first registers with the Chef Server.

Execute on: **workstation**
Execute from: **anywhere**

Demo: View chef-client config file

```
$ knife ssh "os:linux" -x USER -P PWD "cat /etc/chef/client.rb"
```

```
ec2-3-237-240-176.compute-1.amazonaws.com validation_client_name "student02-validator"
ec2-3-237-240-176.compute-1.amazonaws.com chef_license "accept"
ec2-3-237-240-176.compute-1.amazonaws.com log_location STDOUT
ec2-3-237-240-176.compute-1.amazonaws.com node_name "lb"
ec2-3-237-240-176.compute-1.amazonaws.com trusted_certs_dir "/etc/chef/trusted_certs"
ec2-3-237-240-176.compute-1.amazonaws.com file_cache_path "/var/chef/cache"
ec2-3-237-240-176.compute-1.amazonaws.com file_backup_path "/var/chef/backup"
ec2-3-237-240-176.compute-1.amazonaws.com chef_server_url "https://ec2-3-235-139-89.compute-1.amazonaws.com/organizations/student02"
ec2-44-205-9-57.compute-1.amazonaws.com validation_client_name "student02-validator"
ec2-44-205-9-57.compute-1.amazonaws.com chef_license "accept"
ec2-44-205-9-57.compute-1.amazonaws.com log_location STDOUT
ec2-44-205-9-57.compute-1.amazonaws.com node_name "apache_web"
ec2-44-205-9-57.compute-1.amazonaws.com trusted_certs_dir "/etc/chef/trusted_certs"
ec2-44-205-9-57.compute-1.amazonaws.com file_cache_path "/var/chef/cache"
ec2-44-205-9-57.compute-1.amazonaws.com file_backup_path "/var/chef/backup"
```

The configuration file for chef-client is '/etc/chef/client.rb'. This file contains the URL for the Chef Server that chef-client should communicate with, i.e. the API endpoint. The validation_client_name parameter is only used with older versions of Chef Server to define what key is used for the initial authentication with the Chef Server. The file also contains the name of the node, which is used to identify the node on the Chef Server, as well as chef-client's log level and log location.

'chef-client' on the node and 'knife' on the workstation are both API clients in that they both communicate with the Chef Server over the API - the file '/etc/chef/client.rb' is the equivalent to the 'config.rb' file on the workstation.

Execute on: **workstation**

Execute from: **anywhere**

Demo: View chef-client config directory

```
$ knife winrm "os:windows" -x USER -P PWD  
-a cloud.public_ipv4 "dir C:\chef"
```

```
3.88.178.251 Volume in drive C has no label.  
3.88.178.251 Volume Serial Number is D4E5-7A7A  
3.88.178.251 Directory of C:\chef  
3.88.178.251 02/11/2020 07:01 PM <DIR> .  
3.88.178.251 02/11/2020 07:01 PM <DIR> ..  
3.88.178.251 02/11/2020 07:01 PM <DIR> backup  
3.88.178.251 02/11/2020 05:42 PM <DIR> cache  
3.88.178.251 02/10/2020 09:40 PM 36 chef_guid  
3.88.178.251 02/10/2020 09:39 PM 1,706 client.pem  
3.88.178.251 02/10/2020 09:39 PM 349 client.rb  
3.88.178.251 02/10/2020 09:39 PM 17 first-boot.json  
3.88.178.251 02/01/2016 06:41 AM <DIR> ohai
```

We can find similar information on our windows nodes. This will be found in the 'C:\chef' directory on Windows.

```
knife winrm "os:windows" -x USER -P PWD -a cloud.public_ipv4  
"dir C:\chef"
```

Execute on: **workstation**

Execute from: **anywhere**

Demo: View chef-client config file

```
$ knife winrm "os:windows" -x USER -P PWD  
-a cloud.public_ipv4 "more C:\chef\client.rb"
```

```
3.88.178.251 chef_server_url "https://api.chef.io/organizations/jane3"  
3.88.178.251  
3.88.178.251 validation_client_name "jane3-validator"  
3.88.178.251 file_cache_path "c:/chef/cache"  
3.88.178.251 file_backup_path "c:/chef/backup"  
3.88.178.251 cache_options {(:path => "c:/chef/cache/checksums", :skip_expires => true)}  
3.88.178.251 chef_license "accept"  
3.88.178.251 node_name "iis_web"  
3.88.178.251 log_level :auto  
3.88.178.251 log_location STDOUT
```

Execute on: **workstation**

Execute from: **anywhere**

Exercise



GL: The mychef-client Cookbook

- Generate a *mychef-client* cookbook
- Update the *company_web* policyfile to include the *chef-client* cookbook
- Push the *company_web* policyfile to Chef Infra Server
- Apply the updated Policyfile to our web nodes

We will create the mychef-client cookbook to configure chef-client on each of our nodes to run periodically

GL: Create mychef_client cookbook

```
> cd ~/chef-repo/cookbooks  
  
> chef generate cookbook mychef_client  
  
Generating cookbook mychef_client  
....  
Your cookbook is ready. Type `cd mychef_client` to enter it.  
  
There are several commands you can run to get started locally developing and testing your  
cookbook.  
Type `delivery local --help` to see a full list of local testing commands.  
  
Why not start by writing an InSpec test? Tests for the default recipe are stored at:  
  
test/integration/default/default_test.rb  
  
If you'd prefer to dive right in, the default recipe can be found at:  
  
recipes/default.rb
```

‘cd’ command

Execute on: **workstation**

Execute from: **anywhere**

‘chef’ command

Execute on: **workstation**

Execute from: **~/chef-repo/cookbooks**

GL: Edit mychef_client default recipe

`cookbooks/mychef_client/recipes/default.rb`

```
#  
# Cookbook:: mychef_client  
# Recipe:: default  
#  
# Copyright:: 2020, The Authors, All Rights Reserved.  
  
if platform?('windows')  
  chef_client_scheduled_task 'Run as a scheduled task'  
else  
  chef_client_cron 'Run as a cron job'  
end
```

```
if platform?('windows')  
  chef_client_scheduled_task 'Run as a scheduled task'  
else  
  chef_client_cron 'Run as a cron job'  
end
```

GL: Edit company_web.rb Policyfile

```
~/chef-repo/policyfiles/company_web.rb

...# run_list: chef-client will run these recipes in the order
specified.

run_list 'mychef_client::default', 'company_web::default'

# Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
cookbook 'mychef_client', path: '../cookbooks/mychef_client'
```

Add the **mychef_client** cookbook to the run list and cookbook paths.

You should add the 'mychef_client::default' cookbook to the beginning of the run list because system-level configurations should be applied before any application cookbooks.

In this way, if an application cookbook fails during convergence, the chef-client configuration would still be set up.

GL: Update the Policyfile

```
> cd ~/chef-repo/
> chef update policyfiles/company_web.rb

Building policy company_web
Expanded run list: recipe[mychef_client::default],
recipe[company_web::default]
Caching Cookbooks...
Installing company_web    >= 0.0.0 from path
Installing myiis          >= 0.0.0 from path
Installing apache         >= 0.0.0 from path
Installing mychef_client >= 0.0.0 from path

Lockfile written to /home/chef/chef-repo/policyfiles/company_web.lock.json
Policy revision id:
21f9974ef8cde5eea910f11ab4d0ce5661edf3bc644ba09ac399d79ea31066ab
```

GL: Push the Policyfile

```
$ chef push prod policyfiles/company_web.lock.json  
Uploading policy company_web (21f9974ef8) to policy group prod  
Using apache 0.1.0 (d3f5e6c1)  
Using company_web 0.1.0 (e293727a)  
Using myiis 0.2.1 (cd078444)  
Uploaded mychef_client 0.1.0 (f754b557)
```

GL: Converge the Linux web node

```
$ knife ssh 'name:apache_web' -x chef -P PWD 'sudo chef-client'

ec2-18-2016-64-141.compute-1.amazonaws.com Using policy 'company_web' at
revision 'fd0a5ffelcf4748f69bc66076de563fa2d2214ca16803d69ad89f7dbfc09fa39'
ec2-18-2016-64-141.compute-1.amazonaws.com resolving cookbooks for run list:
*
ec2-18-2016-64-141.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-18-2016-64-141.compute-1.amazonaws.com   - apache (0.1.0)
ec2-18-2016-64-141.compute-1.amazonaws.com   - company_web (0.1.0)
ec2-18-2016-64-141.compute-1.amazonaws.com   - myiis (0.2.1)
ec2-18-2016-64-141.compute-1.amazonaws.com   - mychef_client (0.1.0)
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Verify chef-client is running (Linux)

```
$ knife ssh 'name:apache_web' -x chef -P PWD "sudo cat /etc/cron.d/chef-client"
```

```
ec2-44-205-9-57.compute-1.amazonaws.com # Generated by Chef Infra Client. Changes will be overwritten.  
ec2-44-205-9-57.compute-1.amazonaws.com  
ec2-44-205-9-57.compute-1.amazonaws.com 0,30 * * * * root /bin/sleep 106;  
/opt/chef/bin/chef-client -c /etc/chef/client.rb >> /var/log/chef/client.log 2>&1
```

The output shows that chef-client is running as a service and that it is set to run every 30 minutes, which is the default setting.

Execute on: **workstation**

Execute from: **anywhere**

GL: Converge the iis_web node

```
$ knife winrm 'name:iis_web' -x Administrator -P PWD  
-a cloud.public_ipv4 "chef-client"
```

```
34.195.38.226 Using policy 'company_web' at revision  
'7f50faaaaf9eceb35f72358c38a71b86c7fb7df8f49fee245827eb4613d919827'  
34.195.38.226  
34.195.38.226 resolving cookbooks for run list: ["mychef_client::default@0.1.0  
(f5b035d)", "company_web::default@0.1.0 (clb26cb)"]  
34.195.38.226 Synchronizing Cookbooks:  
34.195.38.226 - apache (0.1.0)  
34.195.38.226 - company_web (0.1.0)  
34.195.38.226 - myiis (0.2.1)  
34.195.38.226 - mychef_client (0.1.0) ...
```

We need to apply this updated policy to our iis_web node as well.

Execute on: **workstation**

Execute from: **anywhere**

GL: Verify chef-client is running

```
$ knife winrm 'name:iis_web' -x Administrator -P PWD -a cloud.public_ipv4 'schtasks /Query | findstr /i "chef-client"'
```

```
3.88.178.251 chef-client 2/13/2020 5:52:00 PM Ready
```

Exercise



GL: The mychef-client Cookbook

- ✓ Generate a *mychef-client* wrapper cookbook
- ✓ Update the *company_web* policyfile to include the chef-client wrapper cookbook
- ✓ Push the *company_web* policyfile to Chef Infra Server
- ✓ Apply the updated Policyfile to our web nodes

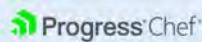
We will create the mychef-client cookbook to configure chef-client on each of our nodes to run periodically



Questions?

Q&A

What questions can we help you answer?



© 2022 Progress Software Corporation and/or its subsidiaries. All rights reserved.

14-20

What questions can we help you answer?

In general or about specifically about node attributes or the 'knife ssh' command.



Progress Chef®

© 2023 Progress Software Corporation. All rights reserved. All other trademarks and/or service marks belong to their respective holders.

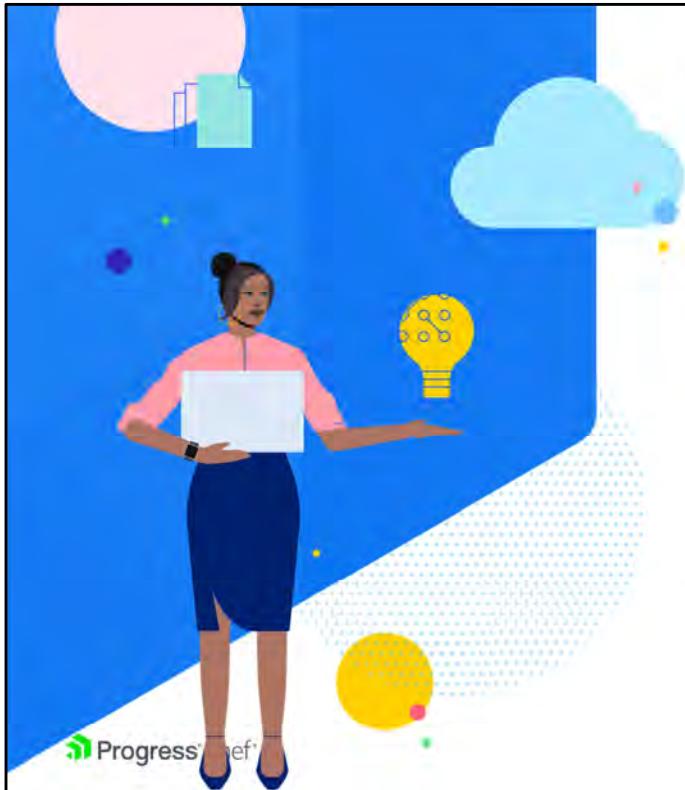
14-21



Using Policy Groups to Reflect Environments

Separating your nodes with policy_group





Objectives

After completing this module, you should be able to

- Deploy a node to an environment via policy_group
- Update the load balancer's search query
- Test your load balancer to confirm that policy_group is separating your node from a group of nodes.

© 2022 Progress Software Corporation and/or its affiliated companies. All rights reserved.

15- 2

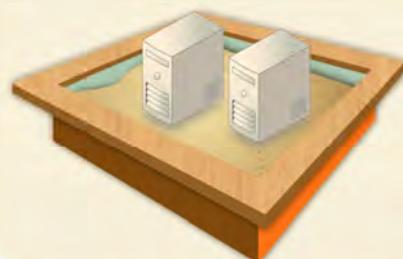
In this section, you will learn how to deploy a node to a policy_group environment and update a search query.

Keeping your infrastructure current

Changing Needs
Changing Software
Growing Organization
Increased Website Popularity

Production

Acceptance



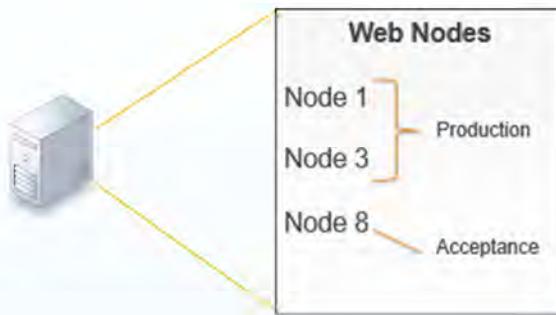
So, we have updated our load balancer's myhaproxy cookbook to dynamically search for and update nodes. Everything is as it should be. But our system is like a living, breathing thing that must grow and be updated to fit our changing needs. We need to find a way to update and test new tools, features 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 policy_group environments are useful.

Policy_group environments

Environments can define different functions of nodes that live in the same infrastructure



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.

Assigning a node to an environment

```
knife node policy set iis_web prod company_web
```



Assigning a node to an environment is as simple as specifying a policy_group in the `knife node policy set...` command.

In this example we assigned the `iis_web` node to the `prod` (production) environment. In this module, you will move your `iis_web` node to a new environment called `acceptance` and see the results.

Assigning a node to an environment

```
knife node policy set iis_web prod company_web
```



We will leave our load balancer and our `apache_web` node in the `prod` environment so the load balancer will serve up only nodes in the `prod` environment.

Reminder: The first time you specify a policy group, that policy group name will be instantiated in Chef Infra Server. Then you can reuse it for other nodes.

Exercise



Group Lab: Using policy_group

- Test your current load balancer's behavior
- Assign the **iis_web** node to acceptance
- Update the load balancer's search criteria to **exclude** nodes in 'acceptance'
- Converge the load balancer node
- Test your load balancer

Let's create an acceptance policy_group environment for our nodes

GL: Test the load balancer

The screenshots illustrate the load balancing process. The first browser window shows the initial state with the Windows host information. After a refresh, the second browser window shows the same host information, indicating a successful switch. The third browser window shows the information for a different host, demonstrating how the load balancer distributes traffic between multiple servers.

Chef Welcomes You!

PLATFORM: windows

HOSTNAME: WIN-8694LT97S51

MEMORY: 8388208kB

CPU Mhz: 2400

Chef Welcomes You!

PLATFORM: windows

HOSTNAME: WIN-8694LT97S51

MEMORY: 8388208kB

CPU Mhz: 2400

Chef Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-29-209

MEMORY: 604192kB

CPU Mhz: 1795.672

Progress Chef®

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.

15- 8

First, point a web browser to the URL of your load balancer/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, just like in a previous module.

Reminder: The haproxy shared cache could take a few second to a couple minutes before it refreshes, so you may not see the "Chef Welcomes You!" page update immediately. You may need to wait 20 or more seconds between refreshes.

GL: Test the load balancer

The image shows three separate browser windows side-by-side, each displaying a "Chef Welcomes You!" page with system information. The top window shows a CentOS host (platform: centos, hostname: ip-172-31-29-209, memory: 604192kB, CPU MHz: 1795.672). The middle-left window shows a Windows host (platform: windows, hostname: WIN-8694LT97S51, memory: 8388208kB, CPU MHz: 2400). The middle-right window shows another CentOS host (platform: centos, hostname: ip-172-31-29-209, memory: 604192kB, CPU MHz: 1795.672). A green arrow points from the top window down to the middle-right window, indicating a transition or comparison between the two CentOS hosts.

Chef Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-29-209

MEMORY: 604192kB

CPU Mhz: 1795.672

Chef Welcomes You!

PLATFORM: windows

HOSTNAME: WIN-8694LT97S51

MEMORY: 8388208kB

CPU Mhz: 2400

Chef Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-29-209

MEMORY: 604192kB

CPU Mhz: 1795.672

Progress Chef™

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.

15 - 9

First, point a web browser to the URL of your load balancer/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, just like in a previous module.

GL: Push the policyfile to the acceptance environment

```
> chef push acceptance policyfiles/company_web.lock.json
```

```
Uploading policy company_web (7680213630) to policy group acceptance
Using apache 0.1.0 (d3f5e6c1)
Using company_web 0.1.0 (fbfd975e)
Using mychef_client 0.1.0 (3411d86f)
Using myiis 0.2.1 (cd078444)
```

New
policy_group

Note that we can push the same policy file to multiple policy groups. In this case we have pushed **company_web.lock.json** to **prod** and also to **acceptance**

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Show the policies on Chef Infra Server

```
> chef show-policy
```

```
company_web
=====
* acceptance: 1e97a11553
* prod:      1e97a11553

myhaproxy
=====
* acceptance: *NOT_APPLIED*
* prod:      e633695adc
```

Here we can see that the **company_web** policy has been uploaded to Chef Infra Server and is in the **acceptance** policy_group.

Execute on: **workstation**
Execute from: **anywhere**

GL: Assign the iis_web node to acceptance

```
> knife node policy set iis_web acceptance company_web
```

```
Successfully set the policy on node iis_web
```

node name

policy_group

policy_name

Here we are setting the iis_web node to the acceptance policy_group.

Execute on: **workstation**
Execute from: **anywhere**

GL: View information about your node

```
> knife node show iis_web
```

```
Node Name: iis_web
Policy Name: company_web
Policy Group: acceptance
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 3.88.178.251
Run List: recipe[mychef_client::default], recipe[company_web::default]
Recipes: mychef_client::default, company_web::default, chef-client::default, chef-client::task, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```

You can see more information about a particular node with the command 'knife node show iis_web'. This will display a summary of the node information that the Chef Infra Server stores.

Execute on: **workstation**

Execute from: **anywhere**

View Windows node information (iis_web)

- Click on Nodes tab and select iis_web
- View the updated policy group

Node	Platform	FQDN	IP Address	Uptime	Last Check-in	Environment
iis_web	centos	ip-172-31-18-98.ec2.internal	172.31.18.98	1 day	--	prod
win001	Windows	WIN-DQFQCUFH0D.ec2.internal	172.31.24.134	2 days	--	prod

View Windows node information (iis_web)

- Click on Nodes tab and select iis_web
- View the updated policy group

Chef Infra Servers > Organizations > Nodes > iis_web

NODE INFORMATION		METADATA	
Environment	prod	Chef Server	cheftraining
Policy Group	acceptance	Chef Organization	student01
Policy Name	company_web		

GL: Update the search to consider policy_group

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
...
web_nodes = search('node',"policy_name:company_web AND policy_group:#{{node.policy_group}}")
server_array = []

web_nodes.each do |one_node|
  one_server = "#{one_node['cloud']['public_hostname']} #{one_node['cloud']['public_ipv4']}:80 maxconn 32"
  server_array.push(one_server)
end
haproxy_backend 'server_backend' do
  server server_array
end
haproxy_service 'haproxy' do
  action [ :enable, :start ]
end
```

Note: We are now using double quotes in the search string.

The double quotes are to interpolate the node.policy_group variable.

Replace the old search criteria with the following:

```
web_nodes = search('node',"policy_name:company_web AND
policy_group:#{{node.policy_group}}")
```

GL: Update the search to consider policy_group

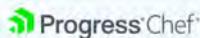
```
~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

...
web_nodes = search('node', "policy_name:company_web AND policy_group:#{node.policy_group}")
servers = []
web_nodes.each do |web_node|
  server = "#{web_node['cloud']['public_hostname']} #{web_node['cloud']['public_ipv4']}:80"
  maxconn 32"
  servers.push(server)
end

haproxy_backend 'servers' do
  server servers
end

haproxy_service 'haproxy' do
  subscribes :reload, 'template[/etc/haproxy/haproxy.cfg]', :delayed
end
```

Note: We are forcing the haproxy service to reload because we've updated the web server pool.

© 2022 Progress Software Corporation and/or its subsidiaries. All rights reserved.15- 17

Instructor Note: This is the first time we've used a notification. These come in the form of "notifies" and "subscribes". Any resource in the resource collection (all resources used during a chef-client run) can send and receive notifications. These allow resources to inform one another when their state changes.

In this example, we are telling the haproxy_service resource to take the action of :reload when the template for the haproxy.cfg file changes on disk. This way, anytime the webserver pool is updated, the haproxy service will be reloaded.

You can have as many notifies or subscribes statements on a resource as you would like, and you can also set timers of :before, :immediately or :after to declare when the actions should take place.

For more information:

https://docs.chef.io/resource_common.html#notifications

https://github.com/sous-chefs/haproxy/blob/master/documentation/haproxy_service.md

GL: Update the myhaproxy version in metadata.rb

```
~/chef-repo/cookbooks/myhaproxy/metadata.rb
```

```
name 'myhaproxy'  
maintainer 'The Authors'  
maintainer_email 'you@example.com'  
license 'All Rights Reserved'  
description 'Installs/Configures myhaproxy'  
version '1.1.0'  
chef_version '>= 16.0'  
depends 'haproxy', '~> 12.2.3'
```

GL: Update the myhaproxy.rb policy

```
> chef update policyfiles/myhaproxy.rb

Building policy myhaproxy
Expanded run list: recipe[myhaproxy::default]
Caching Cookbooks...
Installing myhaproxy >= 0.0.0 from path
Using     haproxy    12.2.3
Using     yum-epel   4.5.0

Lockfile written to C:/Users/Administrator/chef-
repo/policyfiles/myhaproxy.lock.json
Policy revision id:
d04e7e62036b56d53dbc1920d0ad7abd526acfaf3e09a099c751fa2clf9d3b8
```

We just updated `myhaproxy/recipes/default.rb` so we need to update the `myhaproxy.rb` policy.

Execute on: **workstation**

Execute from: **~/chef-repo**

GL: Push the myhaproxy.rb policy

```
> chef push prod policyfiles/myhaproxy.lock.json
```

```
Uploading policy myhaproxy (d04e7e6203) to policy group prod
Using    haproxy 12.2.3 (e7d1c19b)
Using    yum-epel 4.5.0 (dad6a6c0)
Uploaded myhaproxy 1.1.0 (a12936f0)
```

GL: Converge the load balancer node

```
> knife ssh 'name:lb' -x chef -P PWD 'sudo chef-client'
```

```
...
ec2-54-209-220-6.compute-1.amazonaws.com      - update content in file
/etc/haproxy/haproxy.cfg from 91c09e to 8f4138
ec2-54-209-220-6.compute-1.amazonaws.com      - suppressed sensitive resource
ec2-54-209-220-6.compute-1.amazonaws.com      * service[haproxy] action enable (up to
date)
ec2-54-209-220-6.compute-1.amazonaws.com      * service[haproxy] action start (up to
date)
ec2-54-209-220-6.compute-1.amazonaws.com      * haproxy_service[haproxy] action reload
ec2-54-209-220-6.compute-1.amazonaws.com      * service[haproxy] action reload
ec2-54-209-220-6.compute-1.amazonaws.com      - reload service service[haproxy]
ec2-54-209-220-6.compute-1.amazonaws.com      (up to date) ...
ec2-54-209-220-6.compute-1.amazonaws.com Running handlers:
ec2-54-209-220-6.compute-1.amazonaws.com Running handlers complete
ec2-54-209-220-6.compute-1.amazonaws.com Chef Infra Client finished, 2/26 resources
```

Execute on: **workstation**

Execute from: **anywhere**

GL: Only the apache_web node is being proxied

URL of load balancer.

Output from the apache_web server.

Chef Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-29-209

MEMORY: 604192kB

CPU Mhz: 1795.672

Progress Chef

© 2022 Progress Software Corporation and/or its direct and indirect wholly-owned subsidiaries. All rights reserved.

15- 22

Point a web browser to the URL or public IP address of your load balancer. It should display the web page of the apache_web server node that the load balancer is configured to serve. This is because our load balancer and our apache_web server nodes are now in the prod environment policy_group while the iis_web node is in the acceptance environment policy_group.

Exercise



Group Lab: Using policy_group

- ✓ Test your current load balancer's behavior
- ✓ Assign the `iis_web` node to acceptance
- ✓ Update the load balancer's search criteria to **exclude** nodes in 'acceptance'
- ✓ Converge the load balancer node
- ✓ Test your load balancer

Let's create an acceptance policy_group environment for our nodes

Review



Review Questions

1. What is the benefit of constraining cookbooks to a particular environment?
2. What is the key item that defines an environment?
3. What does this bit of code in the load balancer do?

```
web_nodes = search('node',"policy_name:company_web AND policy_group:#{node.policy_group}")
```

1. So you can separate your nodes for specific purposes such as placing a node in an Acceptance environment while testing it.
2. policy_group.
3. It searches for nodes that have the company_web policy_name AND that belong to the same policy_group (environment) that the load balancer is in.



Questions?

Q&A

What questions can we help you answer?



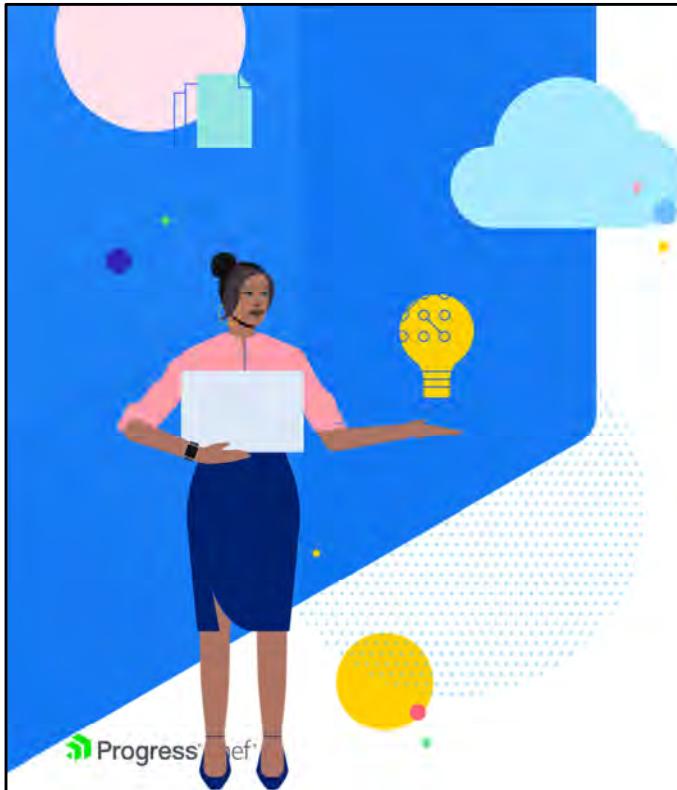


Workstation Installation

Configuring Your Laptop as a Workstation



Now that we have all completed the course, we will walk through installing everything you need to keep going on your journey with Chef.



Objectives

After completing this module, you should be able to

- Install Chef Workstation on your laptop
- Execute commands to ensure everything is installed
- Install a text editor

© 2020 Progress Software Corporation and/or its subsidiaries. All rights reserved.

16- 2

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.

Concept



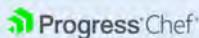
Chef Workstation

Chef Workstation provides everything you need to get started with Chef.

Chef Workstation is a tool chain built on top of the Ruby programming language.

Chef Workstation does not have a graphical-user-interface—it is a collection CLI tools.

<https://docs.chef.io/workstation/>



© 2022 Progress Software Corporation and/or its subsidiaries. All rights reserved.

16- 3

<https://docs.chef.io/workstation/>

Throughout this course we have been using a number of tools found within the Chef Workstation. Chef Workstation contains tools like 'chef' and 'chef-client'. 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.

Chef Workstation 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.

Exercise



GL: Install Chef Workstation and an Editor

- *Install Chef Workstation*
- *Execute commands to ensure everything is installed*
- *Install a text editor (optional)*

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.16- 4

To become a successful Chef developer, you will want to install Chef tools on your local workstation. These tools are available in Chef Workstation.

After the installation is complete, we will verify that the various tools are working.

After that you can optionally download a number of other tools that will help you in your journey using Chef.

Let's get started.

Exercise

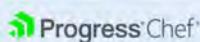


GL: Install Chef Workstation

Click the link below the slide in your participant guide or simply type the URL shown on the slide into a browser.

You'll need to select the correct download for your laptop's system.

<https://downloads.chef.io/chef-workstation/>



© 2022 Progress Software Corporation and/or its subsidiaries. All rights reserved.

16- 5

<https://downloads.chef.io/chef-workstation/>

GL: Installing Chef Workstation

After it downloads, launch the installer.

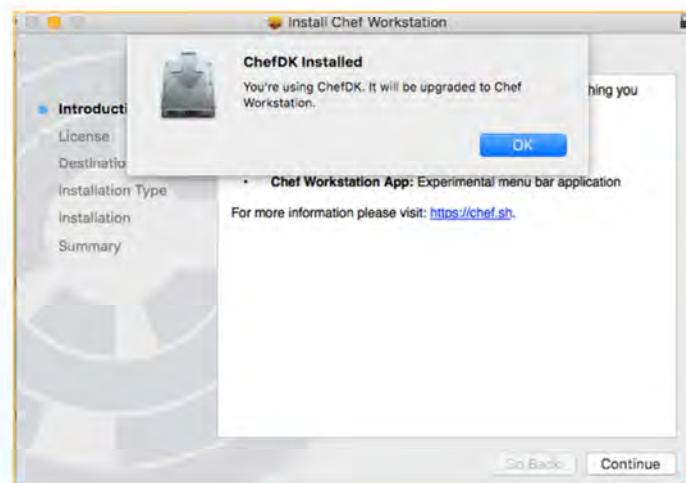
Then double-click the Chef Workstation package.



Follow the Chef Workstation installation wizard's instructions. It could take a few minutes to install.

GL: Installing Chef Workstation

If you get this prompt, allow your old ChefDK to be upgraded to Chef Workstation.



Follow the installation wizard's instructions.

Chef Workstation will be installed into an opscode folder on your laptop.

Exercise



GL: Install Chef Workstation (alternate)

Chocolatey package for Windows

- choco install chef-workstation

Homebrew for Mac

- brew install --cask chef-workstation

Chocolatey: <https://community.chocolatey.org/packages/chef-workstation>

Homebrew: <https://formulae.brew.sh/cask/chef-workstation>

Exercise



GL: Install Chef Workstation and an Editor

- ✓ *Install Chef Workstation*
- *Execute commands to ensure everything is installed*
- *Install a text editor (optional)*

GL: Verify installation of Chef Workstation

```
> chef --version
```

```
Chef Workstation version: 21.7.545
Chef Infra Client version: 17.3.48
Chef InSpec version: 4.38.9
Chef CLI version: 5.3.1
Test Kitchen version: 3.0.0
Cookstyle version: 7.15.2
```

Open a bash session or something like Windows PowerShell and then run this command.

We see some familiar tools like 'chef-client', and test kitchen. These are the ones that we have used on our remote workstation. Some of these tools you have not seen yet.

Lab: Accept all Chef Licenses on your laptop

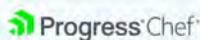
```
$ sudo chef env
+-----+
Chef License Acceptance

Before you can continue, 3 product licenses
must be accepted. View the license at
https://www.chef.io/end-user-license-agreement/

Licenses that need accepting:
* Chef Workstation
* Chef Infra Client
* Chef InSpec

Do you accept the 3 product licenses (yes/no)?
> yes

Persisting 3 product licenses...
✓ 3 product licenses persisted.
```

© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.16-11

Here we are preemptively accepting all Chef licenses.

Accepting Chef licenses is mandatory in all environments such as Chef Workstation, chef-client, Chef Habitat, etc.

(If you installed the latest version of Chef Workstation atop an older version where you already accepted the licenses, the license acceptance may not be apparent in the output.)

Exercise



GL: Install Chef Workstation and an Editor

- ✓ *Install Chef Workstation*
- ✓ *Execute commands to ensure everything is installed*
- *Install a text editor (optional)*



Concept

Text Editors

When working with Chef you spend a large time editing files.

Whatever editor you use should optimize for this workflow.

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.

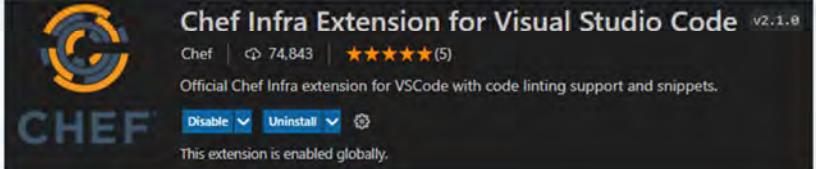
A large number of basic editors that come standard on your operating system are capable of working with chef: notepad; textedit; kedit; etc. However, they are not always optimized for this workflow.

Concept



Visual Studio Code (VSC)

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running and version control. - <https://code.visualstudio.com>



Chef Infra Extension for Visual Studio Code v2.1.0

Chef | 74,843 | ★★★★★(5)

Official Chef Infra extension for VSCode with code linting support and snippets.

Disable Uninstall ⚙

This extension is enabled globally.

 Progress Chef

© 2022 Progress Software Corporation and/or its subsidiaries. All rights reserved.

16-14

You can download VSC at this time, if you don't already have it.

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running and version control. - <https://code.visualstudio.com>

Exercise



GL: Install Chef Workstation and an Editor

- ✓ *Install Chef Workstation*
- ✓ *Execute commands to ensure everything is installed*
- ✓ *Install a text editor (optional)*



Questions?

Q&A

What questions can we answer for you?





Further Resources

Other Places to Talk About, Practice, and Learn Chef



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



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

17- 2

 Progress Chef® Docs Downloads FAQ Contact us Explore Courses Register Sign in

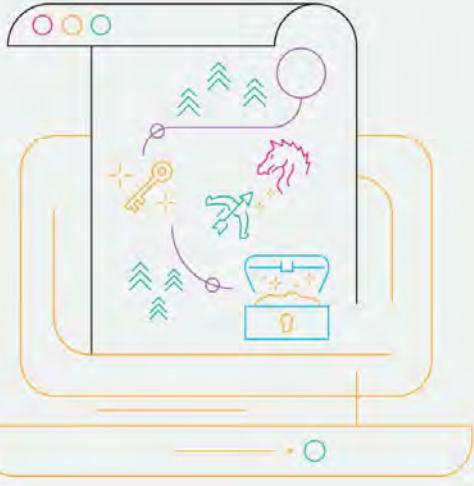
Interactive hands-on learning for those wanting to learn Chef

LEARN CHEF

A new way to learn: Chef, DevOps, and Automation skills.
Expert instruction, on your terms.
Fast, flexible, and free.

[Start Your Journey](#)

learn.chef.io



<https://learn.chef.io/>



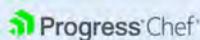
Further Resources

Community Resources

Example Cookbooks, Articles, Podcasts, and More

<https://supermarket.chef.io/>

<https://community.chef.io/>



© 2022 Progress Software Corporation. All rights reserved. All rights reserved.

17- 4

This project contains a living repository of resources curated by the community that showcase some of the better cookbooks to review and use, articles that cover basic and advanced topics, links to podcasts and videos, etc.

<https://supermarket.chef.io/>

<https://community.chef.io/>



Further Resources

Resources you can read

A lot of people in the Chef community have written about Chef.

Here are just a few of those resources.



docs.chef.io

Docs are available to you, 24 hours a day, 7 days a week.

Any question you have, you probably will find the answer for on our Docs site.

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

17- 6

kitchen.ci

More information covering the kitchen.yml file

Ten different drivers covered with examples



© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

17- 7

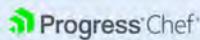


Further Resources

Twitch Channel

<https://www.twitch.tv/chefsoftware>

Live every Tuesday and Thursday. Office Hours, product announcements, book club, demos, everything and anything DevSecOps.



© 2022 Progress Software Corporation and/or its subsidiaries and affiliates. All rights reserved.

17- 8

<https://www.twitch.tv/chefsoftware>

YouTube Channel

- ChefConf Talks
- Training Videos

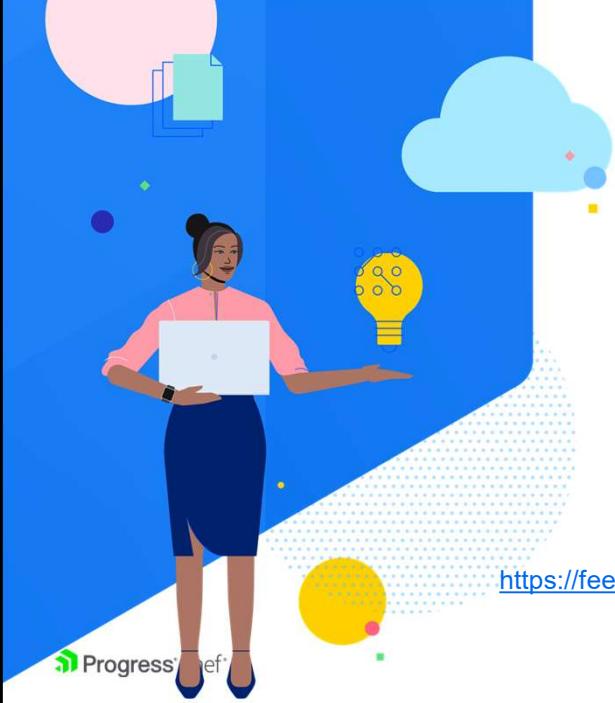
<https://www.youtube.com/user/getchef/playlists>

Progress Chef®

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

17 - 9

We have uploaded a number of videos to the Chef YouTube channel, including training videos and talks from past Chef conferences.



The illustration features a woman with dark hair tied back, wearing a pink long-sleeved top and dark blue pants. She is holding a silver laptop in her hands. To her right is a yellow lightbulb with a network-like pattern inside, symbolizing ideas or innovation. Above the lightbulb is a white cloud with a small red dot. The background is a vibrant blue with abstract shapes like circles and dots in pink, green, and yellow. The Progress Chef logo is visible at the bottom left of the illustration.

Chef product feedback forum

Create your product feature ideas for the Chef engineering teams. As a registered user, you'll be able to vote on your features and the features proposed by others...

<https://feedback.chef.io>

© 2022 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

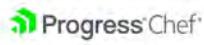
17-10

Help us build the best product. If you have an idea we would love to hear more about it. Or come and vote on other features proposed by others.

<https://feedback.chef.io>



Thank You!



© 2019 Progress Software Corporation. All rights reserved. Progress is a registered trademark of Progress Software Corporation.

17-11

And finally, thank you so much! Hope you enjoyed class!