



CHEFTM

Chef Training Services

Chef Essentials for Linux (CentOS)

Participant Guide

Chef Essentials

Introduction

©2016 Chef Software Inc.

Course v2.1.0



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

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

Instructor Note: Be sure to read Appendix Z for training lab set up notes and additional instructor notes. **Important:** Version 2.x.x of Essentials has a lot of new Day 2 content.

IMPORTANT: This course was tested against ChefDK version 0.17.17. If you use a different version, the exercises and labs may not work properly.

Also, be sure to read the "About Copy/Paste" section of Appendix Z...The participant and instructor guides don't lend themselves to copy/paste very well and don't allow for copying from the PDF'd slides at all. We expect participants to type commands and code rather than relying on copy/paste. (The exceptions are some gist URLs in Day 2 and some other instances where long code has been pasted below some slides in the guides. Such code that is found below slides can be copied.)

Introduce Yourselves

Name

Current job role

Previous job roles/background

Experience with Chef and/or config management

Favorite Text Editor

Expectations

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

You bring with you your own domain expertise and problems. Chef is a framework for solving those problems. Our job is to teach you how to express solutions to your problems with Chef.

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

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

Course Objectives

After completing this course, you should be able to:

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

Agenda

Day 1

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

Day 2

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

Chef

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

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

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

Chef

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

Learning Chef is like learning a language. You will learn the basic concepts very fast but it will take practice until you become comfortable.

A great way to learn Chef is to use Chef

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

Learning Chef is like learning a language. You will learn the basic concepts very fast but it will take practice until you become comfortable.

Chef Fundamentals

Ask Me Anything: It is important that we answer your questions and set you on the path to find more.

Break It: If everything works the first time go back and make some changes. Break it!

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

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

Chef Lab System Architecture

In this course you will use two different architectures:

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

Chef Lab System Architecture

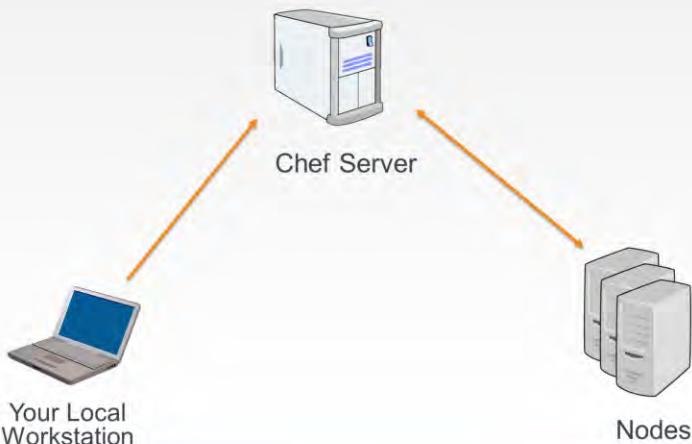
Architecture 1



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

Chef Lab System Architecture

Architecture 2



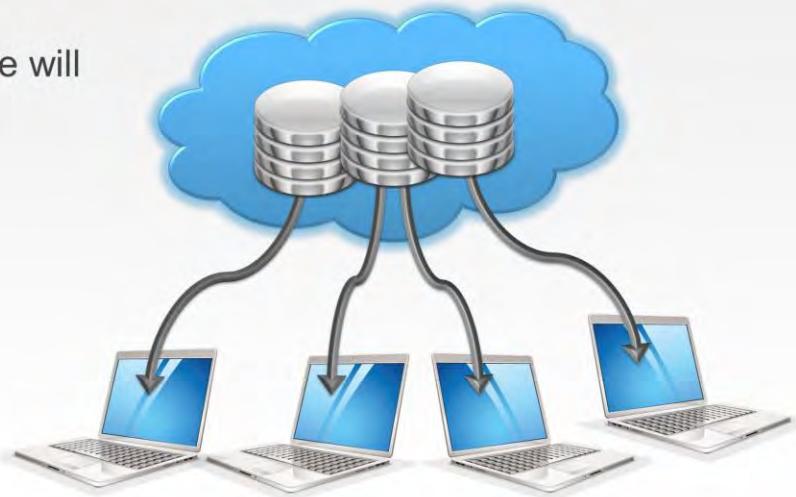
This is the architecture you'll be using later in this course. When using this architecture, the Chef tools will be installed on your laptop and you'll perform your configurations locally before pushing them to the Chef server and ultimately to the nodes you will be managing.

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

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

Getting a Workstation

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



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

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

Hands-on Legend

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

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



Group Lab: Pre-built Workstation

We will provide for you a workstation with all the tools installed.

OBJECTIVE:

- Login to the Remote Workstation



As I mentioned there is a lot work planned for the day. To ensure we focus on the concepts we introduce and not on troubleshooting systems we are providing you a workstation with the necessary tools installed to get started right away.

Instructor Note: At the end of the training it is often a good idea to offer your services to help individuals install necessary software or troubleshoot their systems.

Login to the Workstation



```
> ssh IPADDRESS -l USERNAME
```

```
The authenticity of host '54.209.164.144 (54.209.164.144)' can't  
be established. RSA key fingerprint is  
SHA256:tKoTsPbn6ER9BLThZqntXTxIYem3zV/iTQWvhLrBIBQ. Are you sure  
you want to continue connecting (yes/no)? yes  
chef@54.209.164.144's password: PASSWORD  
chef@ip-172-31-15-97 ~]$
```



We will provide you with the address, username and password of the workstation. With that information you will need to use the SSH tool that you have installed to connect that workstation. On Windows you should use an SSH client like PuTTY to connect to the remote workstation that we assign to you. You'll need to ssh into your assigned workstation in order to issue Chef commands.

This demonstrates how you might connect to the remote machine using your terminal or command-prompt if you have access to the application ssh. This may be different based on your operating system.

Instructor Note: You should assign the participants their Day 1 virtual workstations (AMIs) at this time. The login credentials and password for the virtual workstations is chef/chef.



Group Lab: Pre-built Workstation

We will provide for you a workstation with all the tools installed.

OBJECTIVE:

- ✓ Login to the Remote Workstation



Now that you are connected to that workstation we have taken care of all the necessary work to get started with the training.

Getting a Workstation

The chef user has been granted password-less sudoers access

The following software is installed on the remote workstation:

- Chef DK
- Docker
- kitchen-docker gem



Chef Resources

Chef's Fundamental Building Blocks

©2016 Chef Software Inc.



Objectives

After completing this module, you should be able to:

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

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



GL: Time for Some Fun!

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

Objective:

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

Learning Chef

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

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

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

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

Choose an Editor

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

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

emacs

nano

vi / vim

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

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

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

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

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

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

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

Resources



A resource is a statement of configuration policy.

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

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

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.

Example: Package

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

The package named 'httpd' is installed.

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

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

Instructor Note: The default action for the package resource is create. When you do not specify an action or properties you can define it without the do and end block.

Example: Service

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

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

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

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

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

Example: File

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

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

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

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

Instructor Note: The default action for the file resource is to create the 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/resource_file.html

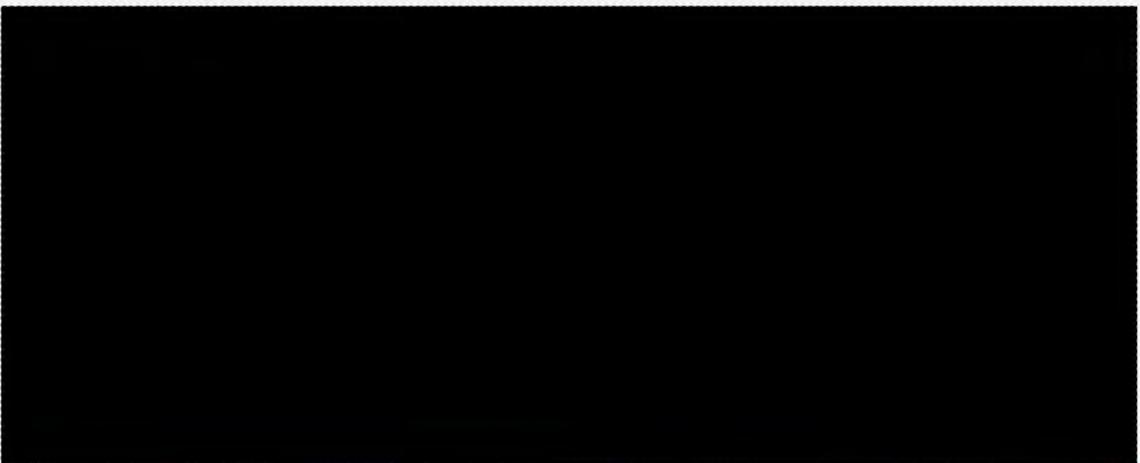
In this example, the file named '/etc/php.ini.default' is deleted.

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.

GL: Use Your Editor to Open the Recipe



```
$ nano moo.rb
```



GL: Update the Moo Recipe

`~/moo.rb`

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

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



GL: Time for Some Fun!

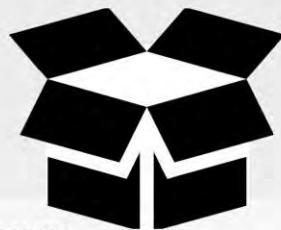
The workstation needs a little personal touch; something that makes it a little more fun.

Objective:

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

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

chef-client



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

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

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

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

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

--local-mode (or -z)



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

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

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

GL: Apply the Setup Recipe



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

```
Starting Chef Client, version 12.13.37
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Installing Cookbook Gems:
Compiling Cookbooks...
[2016-08-22T20:20:45+00:00] WARN: Node ip-172-31-9-151.ec2.internal has an empty run list.
Converging 1 resources
Recipe: @recipe_files:::/home/chef/moo.rb
  * yum_package[cowsay] action install
    - install version 3.03-8.el6 of package cowsay
Running handlers:
Running handlers complete
Chef Client finished, 1/1 resources updated in 01 minutes 25 seconds
```

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

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



GL: Time for Some Fun!

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

Objective:

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

With the package installed it is time to use it.

GL: Run cowsay with a Message



```
$ cowsay will moo for food
```

```
< will moo for food >
-----
 \  ^__^
  \  (oo)\_____
   (__)\       )\/\
      ||----w |
      ||     ||
```



GL: Time for Some Fun!

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

Objective:

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

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



Discussion

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

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

What would happen if the package were to become uninstalled? What would the output be if you applied the recipe again? Was there a situation where the package was already uninstalled and we applied this recipe?



Test and Repair

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

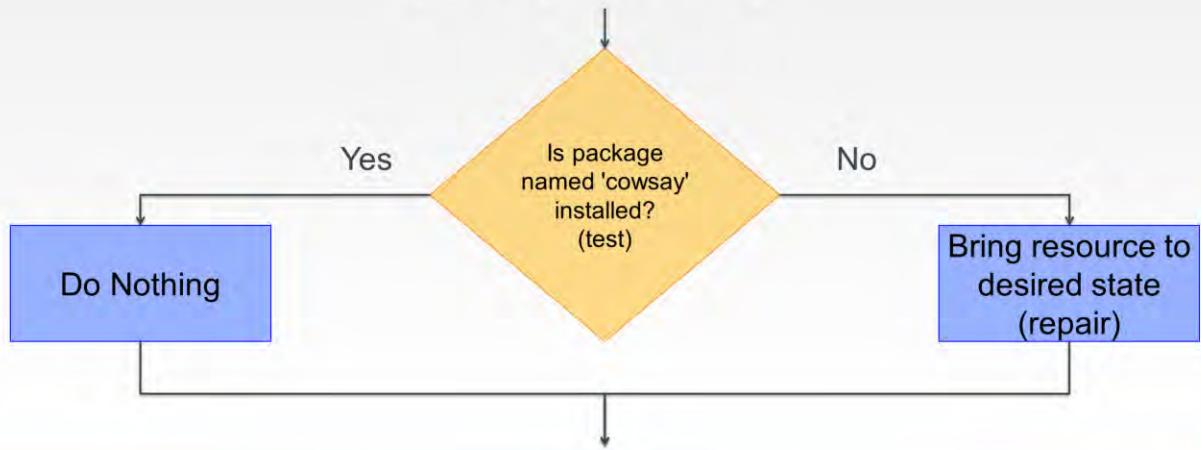
Chef looks at the current state of each resource and takes action only when that resource is out of policy.

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 first tested the system before it takes action.

Test and Repair

```
package 'cowsay'
```





GL: Hello, World?

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

Objective:

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

Great! You installed a package with `chef-client` but we missed a very important step.

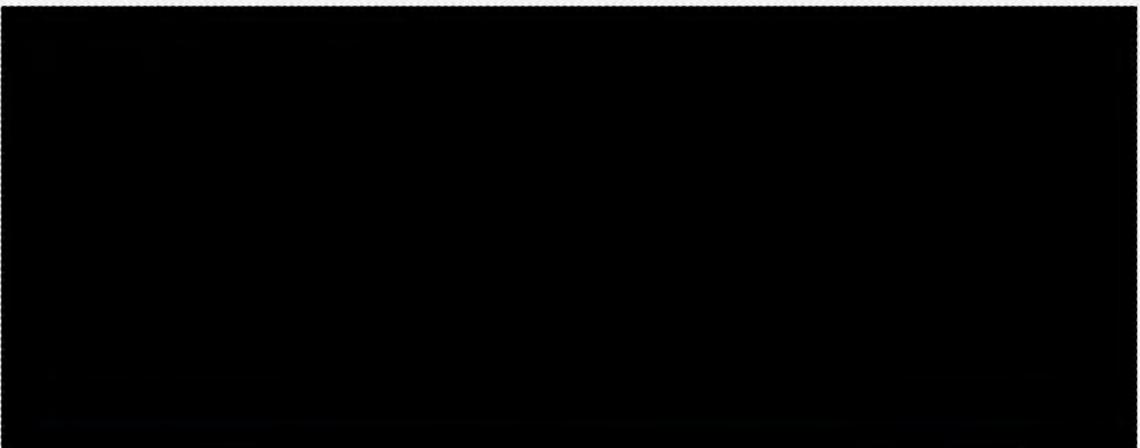
Chef is written in Ruby. Ruby is a programming language and it is required that the first program you write in a programming language is 'Hello World'.

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

GL: Create and Open a Recipe File



```
$ nano hello.rb
```



GL: Create a Recipe File Named hello.rb

~/hello.rb

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

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

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

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

Instructor Note: The default action is to create the file.



GL: Hello, World?

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

Objective:

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

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

GL: Apply the Recipe File



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

```
Starting Chef Client, version 12.13.37
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Compiling Cookbooks...
[2016-02-19T13:08:13+00:00] WARN: Node ip-172-31-12-176.ec2.internal has an empty run list.
Converging 1 resources
Recipe: @recipe_files::/home/chef/hello.rb
* file[hello.txt] action create
  - create new file hello.txt
  - update content in file hello.txt from non to 315f5b
    +++ ./hello.txt20160224-8559-19kqial
      2016-02-24 16:51:04.400844959 +0000
@@ -1 +1,2 @@
+Hello, world!
```

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

Instructor Note: The reason for the sudo prefix in this instance is that it is sometimes easier to build the habit with people.



GL: Hello, World?

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

Objective:

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

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

GL: What Does hello.txt Say?



```
$ cat /hello.txt
```

```
Hello, world!
```

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



GL: Hello, World?

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

Objective:

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

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

Discussion



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

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

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



Test and Repair

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

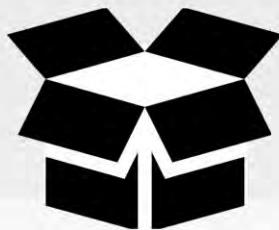
Have we defined a policy for these properties?

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

Instructor Note: The learner is encouraged to change the file permissions, owner, and group here but it is not required. From the resource definition they have not set any of these properties so Chef is relying on the default values provided by the file resource. This prepares them for the next exercise.

Resource Definition

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



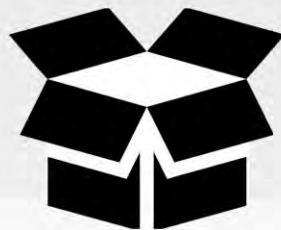
The **TYPE** named **NAME** should be **ACTION'd** with **PROPERTIES**

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.

Resource Definition

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

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



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

Resource Definition

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



The **TYPE** named **NAME** should be **ACTION'd** 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 relative file path to the file we want created. We could have specified a fully-qualified file path to ensure the file was written to the exact same location and not dependent on our current working directory.

Resource Definition

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



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

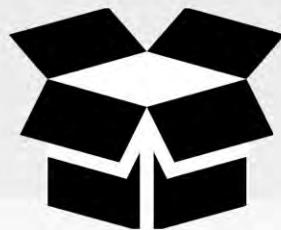
The `do` and `end` keywords here define the beginning of a ruby block. The ruby block and all the contents of it are the second parameters to our resource.

The contents of this block contains 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.

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

Resource Definition

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



The **TYPE** named **NAME** should be **ACTION'd** 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.

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



Lab: The file Resource

- Read** <https://docs.chef.io/resources.html>

- Discover the file resource's:**

- default action.
 - default values for `mode`, `owner`, and `group`.

- Update the file policy in "hello.rb" to:**

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

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

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

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

Instructor Note: Allow 10 minutes to complete this exercise.

Lab: The Updated file Resource

~/hello.rb

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

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

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

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

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

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

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

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

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

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



Lab: The file Resource

✓ **Read** <https://docs.chef.io/resources.html>

✓ **Discover the file resource's:**

- default action.
- default values for `mode`, `owner`, and `group`.

✓ **Update the file policy in "hello.rb" to:**

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

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

Questions



What questions can we answer for you?



Lab: Workstation Setup

- 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.

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Workstation Setup Recipe File

~/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 ...'.

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
    - install version 1.5.3-3.el6 of package tree
  * file[/etc/motd] action create
    - update content in file /etc/motd from e3b0c4 to d100eb
      --- /etc/motd      2010-01-12 13:28:22.000000000 +0000
      +++ /etc/.motd20160224-8754-1xczeyn 2016-02-24 16:57:57.203844958 +0000
      @@ -1 +1,2 @@
      +Property of ...
Running handlers:
Running handlers complete
Chef Client finished, 2/2 resources updated in 17 seconds
```

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



Lab: Workstation Setup

- ✓ 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"



Discussion

What is a resource?

What are some other possible examples of resources?

How did the example resources we wrote describe the desired state of an element of our infrastructure?

What does it mean for a resource to be a statement of configuration policy?

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

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

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
- Test and Repair



Cookbooks

Organizing Recipes

©2016 Chef Software Inc.



Objectives

After completing this module, you should be able to:

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



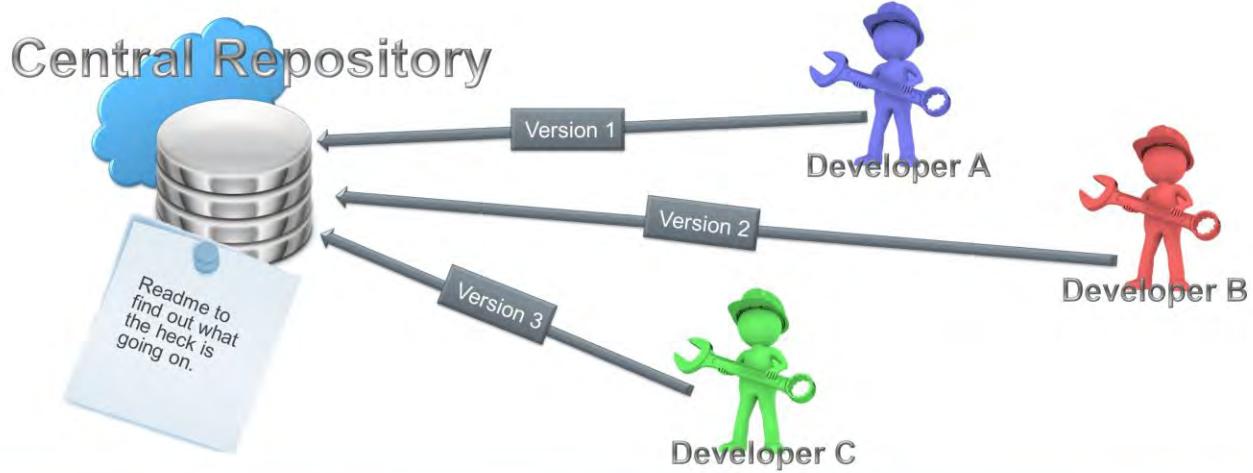
Questions You May Have

1. Thinking about the workstation recipe, could we do something like that for a web server?
2. Is there a way to package up recipes you create with a version number (and maybe a README)?
3. I think `chef` is able to generate something called a cookbook. Shouldn't we start thinking about some version control so we don't lose all our hard work?

Answers:

1. The recipe that you put together to setup the workstation proved useful--useful enough to see if the same could be done with a webserver. It's a package, a file, and a service. Everything you've already completed. Well, almost everything.
2. Now the request to add version control and a README would definitely make it easier to share the recipes that we create. Without version control we'd have no way to build this software collaboratively or recover our work. Without a README no one would know what the recipe even was supposed to do or what it did.
3. And yes, before we start creating more recipes and cookbooks, we should choose a versioning solution.

Collaboration and Version Control



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

A versioning system should include:

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

Each revision should be stored as a new version.

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

Versioning Pros and Cons

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

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

We could use the current date and time down to the minute like in the second example. As a group 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 the rest of this course.



How about we use git? It is automatically installed in this version of the ChefDK (0.17.17).

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.



GL: Create a Cookbook

How are we going to manage this file? Does it need a README?

Objective:

- Use chef to generate a cookbook
- Move the setup recipe into the new cookbook
- Add the new cookbook to version control

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

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

Cookbooks

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

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

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



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

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.

<http://docs.chef.io/cookbooks.html>

Working within Home Directory



```
$ cd ~
```



Before we get started let's return to the home directory.

GL: Create a Cookbooks Directory



```
$ mkdir cookbooks
```



Storing our recipes within a cookbook sounds like a better idea than storing them in a scripts directory. To prepare for the cookbook we are about to create and the other cookbooks we will also be creating today it would be a good idea to store them in shared directory.

Create a directory named cookbooks.

What is 'chef'?

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



A cookbook is a directory with files laid out in a specific, agreed upon structure. 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 Development Kit (Chef DK) comes with a tool named 'chef'. This command-line tool has a number of features.

What can 'chef' do?



```
$ chef --help
```

Usage:

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

Available Commands:

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

What Can 'chef generate' Do?



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

```
Usage: chef generate GENERATOR [options]

Available generators:
  app           Generate an application repo
  cookbook      Generate a single cookbook
  recipe        Generate a new recipe
  attribute     Generate an attributes file
  template      Generate a file template
  file          Generate a cookbook file
  lwrp          Generate a lightweight resource/provider
  repo          Generate a Chef policy repository
  policyfile    Generate a Policyfile for use with the install/push commands
  generator     Copy ChefDK's generator cookbook so you can customize it
```

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'.

GL: Let's Create a Cookbook



```
$ chef generate cookbook cookbooks/workstation
```

```
Generating cookbook workstation
- Ensuring correct cookbook file content
- Committing cookbook files to git
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content
- Adding delivery configuration to feature branch
- Adding build cookbook to feature branch
- Merging delivery content feature branch to master
```

```
Your cookbook is ready. Type `cd cookbooks/workstation` 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.

We have you covered. Call the cookbook *workstation*. When you specify the command we provide the name of the cookbook with the proceeding cookbooks path. This is because we are in the home directory and we want the cookbook to be generated in the cookbooks directory we recently created.

GL: The Cookbook Has a README



```
$ tree cookbooks/workstation
```

```
workstation
├── Berksfile
├── chefignore
├── metadata.rb
└── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
...
6 directories, 8 files
```

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

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



README.md

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

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

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

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

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

GL: The Cookbook Has Some Metadata



```
$ tree cookbooks/workstation
```

```
workstation
├── Berksfile
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
└── spec
    ├── spec_helper.rb
    └── unit
        └── recipes
...
6 directories, 8 files
```

The cookbook also has a metadata file.



metadata.rb

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

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

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

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

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'
description   'Installs/Configures workstation'
long_description 'Installs/Configures workstation'
version       '0.1.0'

# If you upload to Supermarket you should set this so your cookbook
# gets a `View Issues` link
# issues_url 'https://github.com/<insert_org_here>/workstation/issues' if
respond_to?(:issues_url)
```

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.

GL: The Cookbook Has a Folder for Recipes



```
$ tree cookbooks/workstation
```

```
workstation
├── Berksfile
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
└── spec
    ├── spec_helper.rb
    └── unit
        └── recipes
...
6 directories, 8 files
```

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?

GL: The Cookbook Has a Default Recipe



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

```
# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
```

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

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



GL: Create a Cookbook

How are we going to manage this file? Does it need a README?

Objective:

- Use chef to generate a cookbook
- Move the setup recipe into the new cookbook
- Add the new cookbook to version control

Now that we have a cookbooks directory and within it the workstation cookbook it is now time to move the setup recipe into the new cookbook.

GL: Copy the Recipe into the Cookbook



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

From the Home directory, move your setup recipe to the workstation cookbook. This will place the recipe alongside the default recipe already present within the workstation cookbook.

GL: Verify the Cookbook has the Recipe

 \$ tree cookbooks/workstation

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





Group Exercise: Version Control

This is probably a good point to capture the initial state of our cookbook.

Objective:

- ✓ Use chef to generate a cookbook
- ✓ Move the setup recipe into the new cookbook
- Add the new cookbook to version control

Now we have the setup recipe in its new home within the workstation cookbook, it is time to start tracking the changes that we make to it by setting up version control.

GL: Move into the Cookbook Directory



```
$ cd cookbooks/workstation
```

GL: Initialize the Directory as a git Repository



```
$ git init
```

```
Reinitialized existing Git repository in /home/chef/cookbooks/workstation/.git/
```

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 automatically initialized the cookbook as a git repository.

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



```
$ git add .
```

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

This will place all the files into a staging area.



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

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

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



```
$ git status
```

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  .gitignore
    new file:  .kitchen.yml
    new file:  Berksfile
    new file:  README.md
    new file:  chefignore
    new file:  metadata.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.

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



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

```
[master (root-commit) 73b39cb] Initial commit
Committer: ChefDK User <chef@ip-172-31-14-46.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
```

```
git config --global --edit
```

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

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

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

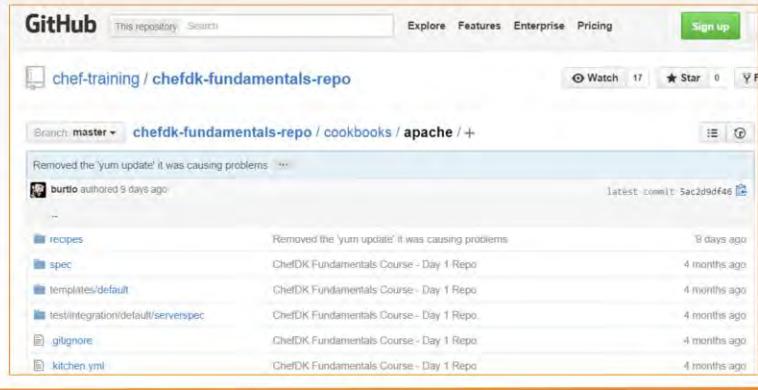
This is like saying we're ready to close the box up.

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

Git Version Control

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

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



The screenshot shows a GitHub repository page for 'chef-training / chefdk-fundamentals-repo'. The 'apache' cookbook's commit history is displayed under the 'Branch: master' dropdown. The commits are as follows:

File	Commit Message	Date
recipes	Removed the 'yum update' it was causing problems	9 days ago
spec	ChefDK Fundamentals Course - Day 1 Repo	4 months ago
templates/default	ChefDK Fundamentals Course - Day 1 Repo	4 months ago
test/integration/default/serverspec	ChefDK Fundamentals Course - Day 1 Repo	4 months ago
.gitignore	ChefDK Fundamentals Course - Day 1 Repo	4 months ago
kitchen.yml	ChefDK Fundamentals Course - Day 1 Repo	4 months ago

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.

GL: Return to the Home Directory



```
$ cd ~
```

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



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: Create a Cookbook



```
$ chef generate cookbook cookbooks/apache
```

```
Generating cookbook apache
- Ensuring correct cookbook file content
- Committing cookbook files to git
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content
- Adding delivery configuration to feature branch
- Adding build cookbook to feature branch
- Merging delivery content feature branch to master
```

```
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.
```

```
Type `delivery local --help` to see a full list.
```

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

Lab: Create a Cookbook



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

```
Compiling Cookbooks...
Recipe: code_generator::recipe
  * directory[cookbooks/apache/spec/unit/recipes] action create (up to date)
  * cookbook_file[cookbooks/apache/spec/spec_helper.rb] action create_if_missing
    (up to date)
  * 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 a43970
      (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
      a43970
```

Lab: Create 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.

Lab: Apply the Server Recipe



```
$ sudo chef-client -z cookbooks/apache/recipes/server.rb
```

```
Converging 3 resources
Recipe: @recipe_files:::/home/chef/cookbooks/apache/recipes/server.rb
  * yum_package[httpd] action install
    - install version 2.2.15-47.el6.centos.3 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 2016-02-24 21:41:45.494844958 +0000
      +++ /var/www/html/.index.html20160224-10036-6y8on7 2016-02-24
      21:41:45.493844958 +0000
    @@ -1 +1,2 @@
    +<h1>Hello, world!</h1>
  * service[httpd] action enable
    - enable service service[httpd]
```

Let's apply the recipe with 'chef-client'. We need to specify the partial path to the recipe file within the apache cookbook's recipe folder. This is quite a bit more text to type and one of those moments where you may find yourself using the shorter (-z) flag over specifying (--local-mode).

Lab: Verify That the Website is Available



```
$ curl localhost
```

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



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`

Congratulations. Setting the Web Server in a new cookbook with the resources you have defined and then applying it to the system demonstrates that you understand the fundamentals of Chef.

GL: Commit Your Work



```
$ cd cookbooks/apache  
$ git add .  
$ git commit -m "Initial commit"
```

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. And commit all the files in the staging area.

A Note About `cookstyle`

Later in this course you will learn about Test Kitchen and how it can be used to test cookbook data across a combination of platforms and test suites.

But what if you want to do a quick syntax test of your recipe?

That's where `cookstyle` can come in handy.

A Note About `cookstyle`

`cookstyle` is a linting tool based on RuboCop that comes with ChefDK.

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

<https://docs.chef.io/release/devkit/cookstyle.html>

<https://docs.chef.io/release/devkit/cookstyle.html>

Example: cookstyle

```
 $ cookstyle  
Inspecting 1 file  
C  
  
Offenses:  
  
server.rb:10:1: C: 1 trailing blank lines detected...  
  
1 file inspected, 1 offense detected
```

In this example, we ran `cookstyle` against the server.rb recipe we created in this module. `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.

Example: cookstyle

```
 $ cookstyle
Inspecting 6 files
..C...

Offenses:

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

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



Discussion

What file would you read first when examining a cookbook?

What other recipes might you include in the apache or workstation cookbook?

Can resources accept multiple actions?

How often would you commit changes with version control?

Answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.



Q&A

What questions can we answer for you?

- Cookbooks
- Versions
- Version control

What questions can we help you answer?

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



chef-client

Applying Recipes from Multiple Cookbooks

©2016 Chef Software Inc.



chef-client

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



How would we apply both the workstation's setup recipe and apache's server 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.

Objectives

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

- Locally apply multiple cookbooks' recipes with chef-client.
- Include a recipe from within another recipe.

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

--local-mode

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

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



chef-client has a number of flags that can be passed to it to configure how it works. Up to this point we have been using the '--local-mode' flag to ensure 'chef-client' does not query the Chef Server it wants to communicate with by default.



--runlist "recipe[COOKBOOK::RECIPE]"
-r "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 '--run-list' 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.

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

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

Applying the following recipes locally:

The 'setup' recipe from the 'workstation' cookbook

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

We are using the abbreviated '-r' to represent the longer flag '--run-list'.

Instructor Note: These commands if executed by a learner at this point will not work. These are being displayed solely as demonstration.

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

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

Applying the following recipes locally:

The 'server' recipe from the 'apache' cookbook

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

We are using the abbreviated '-r' to represent the longer flag '--run-list'.

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

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

Applying the following recipes locally:

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

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

Instructor Note: The command given here includes the backslash '\'. That allows you to specify multiple lines within a terminal. Because of the character limitation of slides it is included to make the command more clear to the learner.

Instructor Note: It is important to note that when specifying a run list, recipes defined within it that are separated with a comma should NOT have a space after the comma or it will create an error.



Applying a Run List

Using a run list will allow us to specify things more 'logically' instead of with paths.

Objective:

- Individually apply the apache cookbook's server recipe and workstation cookbook's setup recipe
- Apply both the apache cookbook's server recipe and workstation cookbook's setup recipe

GL: Return Home First



```
$ cd ~
```

Before you start applying cookbooks through 'chef-client', make sure you are in your home directory. This is important because 'chef-client' looks for cookbooks in the 'cookbooks' directory.

GL: Apply the Cookbook Recipe Locally



```
$ sudo chef-client --local-mode -r "recipe[apache::server]"  
[2016-09-15T14:54:45+00:00] WARN: No config file found or specified on command  
line, using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["apache::server"]  
Synchronizing Cookbooks:  
  - apache  
Compiling Cookbooks...  
Converging 3 resources  
Recipe: apache::server  
  * yum_package[httpd] action install (up to date)  
  * file[/var/www/html/index.html] action create (up to date)  
  * service[httpd] action enable (up to date)
```

From our current working directory, which at the moment is the home directory, the 'chef-client' command will look for a directory named 'cookbooks'. Within that 'cookbooks' directory it will find the cookbook named 'apache'. Within the 'apache' cookbook it will look for the 'server' recipe.

Run this command, from the home directory, and ensure that you see the recipe being applied as it had before except this time the output will display a populated run list and a cookbook that has been synchronized.

Instructor Note: The WARN messages were omitted from this output so you can see the converging resources.

GL: Apply the Cookbook Recipe Locally



```
$ sudo chef-client --local-mode -r "recipe[workstation::setup]"  
[2016-09-15T15:15:26+00:00] WARN: No config file found or specified on command  
line, using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["workstation::setup"]  
Synchronizing Cookbooks:  
  - workstation (0.1.0)  
Compiling Cookbooks...  
Converging 2 resources  
Recipe: workstation::setup  
  * yum_package[tree] action install (up to date)  
  * file[/etc/motd] action create (up to date)
```

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



Applying a Run List

Using a run list will allow us to specify things more 'logically' instead of with paths.

Objective:

- ✓ Individually apply the apache cookbook's server recipe and workstation cookbook's setup recipe
- Apply both the apache cookbook's server recipe and workstation cookbook's setup recipe

Now we see one recipe at a time being applied. Let's try two at a time.

GL: Apply Both Recipes Locally



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

```
[2016-09-15T15:17:27+00:00] WARN: No config file found or specified on  
command line, using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["apache::server", "workstation::setup"]  
Synchronizing Cookbooks:  
  - apache  
  - workstation  
Compiling Cookbooks...  
  
Running handlers:  
...
```

Try applying both recipes from both cookbooks.

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

Instructor Note: It is important to note that when specifying a run list, recipes defined within it that are separated with a comma should NOT have a space after the comma or it will create an error.



Applying a Run List

Using a run list will allow us to specify things more 'logically' instead of with paths.

Objective:

- ✓ Individually apply the apache cookbook's server recipe and workstation cookbook's setup recipe
- ✓ Apply both the apache cookbook's server recipe and workstation cookbook's setup recipe

Applying two at a time demonstrates for us that if we need to apply more than one recipe at a time, with a single command, it can be done.



A Succinct Run List

The cookbook only has one recipe that we care about. Could we set that up as the default?

Objective:

- Load the workstation cookbook's setup recipe in the default recipe
- Apply the workstation cookbook's default recipe
- Commit the changes

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'.

-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.



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

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

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

include_recipe



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>

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

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

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

Demo: Including a Recipe

```
include_recipe 'workstation::setup'
```

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

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

Demo: Including a Recipe

```
include_recipe 'apache::server'
```

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

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

GL: The Default Recipe Includes the Setup Recipe

```
1 ~/cookbooks/workstation/recipes/default.rb
```

```
#  
# Cookbook Name:: workstation  
# Recipe:: default  
#  
# Copyright (c) 2016 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`.



A Succinct Run List

The cookbook only has one recipe that we care about. Could we set that up as the default?

Objective:

- Load the workstation cookbook's setup recipe in the default recipe
- Apply the workstation cookbook's default recipe
- Commit the changes

Now when the default recipe is applied it will include the setup recipe. Lets put that to the test by applying the recipe with 'chef-client'.

GL: Apply the Cookbook's Default Recipe



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

```
WARN: No config file found or specified on command line, using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["workstation"]  
Synchronizing Cookbooks:  
  - workstation (0.1.0)  
Compiling Cookbooks...  
Converging 2 resources  
  
Running handlers:  
Running handlers complete  
Chef Client finished, 0/2 resources updated in 12 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.



A Succinct Run List

The cookbook only has one recipe that we care about. Could we set that up as the default?

Objective:

- ✓ Load the workstation cookbook's setup recipe in the default recipe
- ✓ Apply the workstation cookbook's default recipe
- Commit the changes

We made a change to the cookbook. We were able to verify that it applied correctly. Now would be a good time to commit the changes we have made to version control.

GL: Commit Your Work



```
$ cd cookbooks/workstation  
$ git add .  
$ git commit -m "Update default recipe to include setup  
recipe"
```



A Succinct Run List

The cookbook only has one recipe that we care about. Could we set that up as the default?

Objective:

- ✓ Load the workstation cookbook's setup recipe in the default recipe
- ✓ Apply the workstation cookbook's default recipe
- ✓ Commit the changes

Right now specifying the workstation's setup recipe as the default recipe works. In the future we may find that we need to specify a different one as the default.

GL: Return Home



```
$ cd ~
```



Lab: Update the apache Cookbook

- Update the "apache" cookbook's "default" recipe to:

Include the 'server' recipe from the 'apache' cookbook

- Run chef-client and locally apply the run_list: "recipe[apache]"
- Commit the changes with version control

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

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: The Default Recipe Includes the Apache Recipe

1 ~/cookbooks/apache/recipes/default.rb

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
include_recipe 'apache::server'
```

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

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

Lab: Applying the apache Default Recipe



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

```
[2016-09-15T15:23:18+00:00] WARN: No config file found or specified on command line, using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["apache"]  
Synchronizing Cookbooks:  
  - apache (0.1.0)  
Compiling Cookbooks...  
Converging 3 resources  
  
Running handlers:  
Running handlers complete  
Chef Client finished, 0/3 resources updated in 3.310768509 seconds
```

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

Lab: Commit Your Work

```
$ cd cookbooks/apache  
$ git add .  
$ git commit -m "Update default recipe to include server  
recipe"
```





Lab: Update the apache Cookbook

- ✓ Update the "apache" cookbook's "default" recipe to:

Include the 'server' recipe from the 'apache' cookbook

- ✓ Run chef-client and locally apply the run_list: "recipe[apache]"
- ✓ Commit the changes with version control

Wonderful. Now the apache cookbook's default recipe includes the server recipe. You were able to verify that by applying chef-client with the abbreviated run list. Finally it was a good time to commit the changes that you made.



Discussion

Why would you want to apply more than one recipe at a time?

What are the benefits and drawbacks of using "include_recipe" within a recipe?

Do default values make it easier or harder to learn?

Answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.



Q&A

What questions can we help you answer?

- chef-client
- local mode
- run list
- include_recipe

What questions can we help you answer?

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



Testing Cookbooks

Validating Our Recipes in Virtual Environments

©2016 Chef Software Inc.



Objectives

After completing this module, you should be able to

- Use Test Kitchen to verify your recipes converge on a virtual instance
- Define an InSpec test
- Write and execute tests

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



Can We Test Cookbooks?

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

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?

DISCUSSION



Mandating Testing

What steps would it take to test one of the cookbooks that we have created?

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

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

Instructor Note: This exercise is useful in helping the learners visualize the each step of testing process and how Test Kitchen maps to each of those steps

Steps to Verify Cookbooks

- Create Virtual Machine
- Install Chef Tools
- Copy Cookbooks
- Run/Apply Cookbooks
- Verify Assumptions
- Destroy Virtual Machine

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

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

Instructor Note: The class participant should be able to create a list of similar steps. The names and the detail may vary based on their experience or expertise. Instead of presenting this slide you may find it more engaging to invite the learners to share the list of steps that they created and create a list that represents the voice of the group. If you do, you may find it useful to hide this slide.

Testing Cookbooks

We can start by first mandating that all cookbooks are tested

How often should you test your cookbook?

How often do you think changes will occur?

What happens when the rate of cookbook changes exceed the time interval it takes to verify the cookbook?

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

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

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



Code Testing

An automated way to ensure code accomplishes the intended goal and help the team understand its intent

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

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



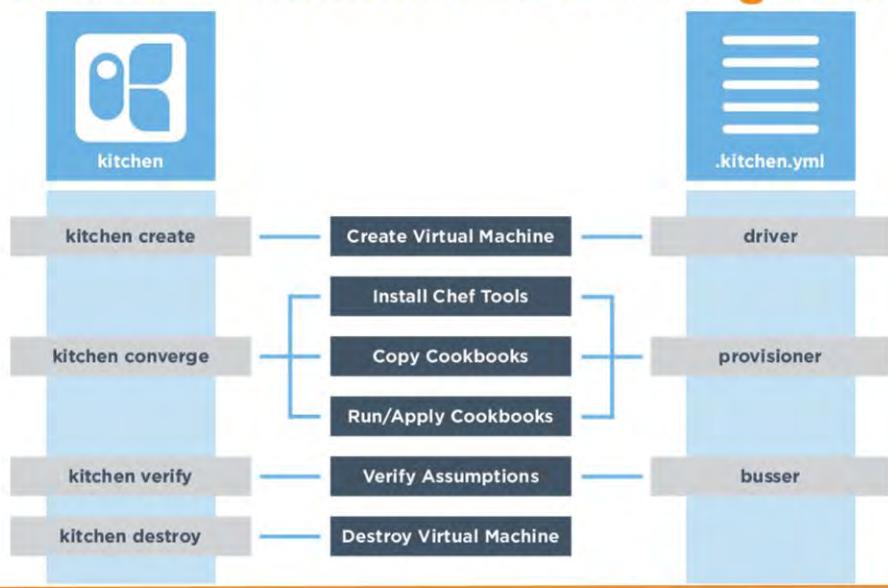
Test Configuration

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

Objective:

- Configure the "workstation" cookbook to test against the centos-6.7 platform
- Test the "workstation" cookbook on a virtual machine

Test Kitchen Commands and Configuration



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 the configuration will be explained in more detail.

Instructor Note: If you created a custom list of steps with your learners use that custom list and overlay the following information over top of it.

What Can 'kitchen' Do?



```
$ kitchen --help
```

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

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

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

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

What Can 'kitchen init' Do?



```
$ kitchen --help init
```

```
UsaGL:  
  kitchen init  
    -D, [--driver=one two three]          # One or more Kitchen Driver gems ...  
                                           # Default: kitchen-vagrant  
    -P, [--provisioner=PROVISIONER]        # The default Kitchen Provisioner to  
use  
                                           # Default: chef_solo  
    [--create-gemfile], [--no-create-gemfile] # Whether or not to create a Gemfi ...  
  
Description:  
  Init will add Test Kitchen support to an existing project for convergence  
  integration testing. A default .kitchen.yml file (which is intended to be  
  customized) is created in the project's root directory and one or more gems will be  
  added to the project's Gemfile.
```

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

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

Do We Have a .kitchen.yml?



```
$ tree cookbooks/workstation -a
```

```
...
├── .kitchen.yml
├── metadata.rb
├── README.md
├── recipes
│   ├── default.rb
│   └── setup.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
│           └── default_spec.rb
└── test
```

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

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

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

What is Inside .kitchen.yml?



```
$ cat cookbooks/workstation/.kitchen.yml
```

```
---
```

```
driver:
```

```
  name: vagrant
```

```
provisioner:
```

```
  name: chef_zero
```

```
verifier:
```

```
  name: inspec
```

```
platforms:
```

```
  - name: ubuntu-16.04
```

```
  - name: centos-7.2
```



.kitchen.yml

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

<http://kitchen.ci/docs/getting-started/creating-cookbook>

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 let's learn more about the various fields in the configuration file.

<http://kitchen.ci/docs/getting-started/creating-cookbook>

Demo: The kitchen Driver

~/cookbooks/workstation/.kitchen.yml

```
---
```

```
driver:
```

```
  name: vagrant
```



```
provisioner:
```

```
  name: chef_zero
```



```
verifier:
```

```
  name: inspec
```



```
platforms:
```

```
  - name: ubuntu-16.04
```

```
  - name: centos-7.2
```

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

Example Drivers:

- docker
- vagrant

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

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

Instructor Note: Testing on this remote workstation requires that we use Docker because Vagrant does not work within a virtual environment. Vagrant is the standard choice when working on your local workstation.

Demo: The kitchen Provisioner

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

verifier:
  name: inspec

platforms:
  - name: ubuntu-16.04
  - name: centos-7.2
```

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 Verifier

~/cookbooks/workstation/.kitchen.yml

```
---
```

```
driver:
```

```
  name: vagrant
```



```
provisioner:
```

```
  name: chef_zero
```



```
verifier:
```

```
  name: inspec
```



```
platforms:
```

```
  - name: ubuntu-16.04
```

```
  - name: centos-7.2
```

This tells Test Kitchen how to verify the converged instances.

The default approach is to use InSpec.

The third key is the verifier, which also has a single key-value pair which is the name of the verifier Kitchen will use when executed. The verifier is responsible for reading and executing the test suite defined for the cookbook. Test Kitchen supports many different verifiers. InSpec is the default and the best approach as you continue to work with developing cookbooks.

Demo: The kitchen Platforms

~/cookbooks/workstation/.kitchen.yml

```
---
```

```
driver:
```

```
  name: vagrant
```

```
provisioner:
```

```
  name: chef_zero
```

```
verifier:
```

```
  name: inspec
```

```
platforms:
```

```
  - name: ubuntu-16.04
```

```
  - name: centos-7.2
```

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

The fourth 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.

Demo: The kitchen Suites

~/cookbooks/workstation/.kitchen.yml

```
...
suites:
- name: default
  run_list:
    - recipe[workstation::default]
  verifier:
    inspec_tests:
      - test/recipes
  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 fifth 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

~/cookbooks/workstation/.kitchen.yml

```
...
suites:
- name: default
  run_list:
    - recipe[workstation::default]
verifier:
  inspec_tests:
    - test/recipes
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.



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.

Example: Kitchen Test Matrix

```
$ kitchen list
```

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

```
suites:                                platforms:  
  - name: default                         - name: ubuntu-12.04  
    run_list:                                - name: centos-6.5  
      - recipe[workstation::default]  
    attributes:
```

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.

Instructor Note: This command will fail if run on the workstations because the vagrant driver is still defined on the remote workstation. This is an example.

Example: Kitchen Test Matrix

```
$ kitchen list
```

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

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

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

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.

Instructor Note: This command will fail if run on the workstations because the vagrant driver is still defined on the remote workstation. This is an example.



Group Exercise: Test Configuration

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

Objective:

- Configure the "workstation" cookbook's .kitchen.yml to use the Docker driver and centos 6.7 platform
- Use kitchen converge to apply the recipe on a virtual machine

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

GL: Move into the Cookbook's Directory



```
$ cd cookbooks/workstation
```

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

GL: Edit the Kitchen Configuration File

~/cookbooks/workstation/.kitchen.yml

```
--  
driver:  
  name: docker  
  
provisioner:  
  name: chef_zero  
  
verifier:  
  name: inspec  
  
platforms:  
  - name: centos-6.7
```



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

Instructor Note: The reason we are using the Docker driver is that it is possible to run this on cloud platforms and perform virtualization within the already existing virtualization.

GL: Edit the Kitchen Configuration File

~/cookbooks/workstation/.kitchen.yml

```
--  
driver:  
  name: docker  
  
provisioner:  
  name: chef_zero  
  
verifier:  
  name: inspec  
  
platforms:  
  - name: centos-6.7
```



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

GL: Look at the Test Matrix



```
$ kitchen list
```

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

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



Converging a Cookbook

Before I add features it really would be nice to test these cookbooks against the environments that resemble production.

Objective:

- ✓ Configure the "workstation" cookbook's .kitchen.yml to use the Docker driver and centos-6.7 platform
- ❑ Use kitchen converge to apply the recipe on a virtual machine

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



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.

Running `kitchen create default-centos-67` would create the one instance that uses the test suite on the platform we want.

Typing in that name would be tiring if you had a lot of instances. A shortcut can be used to target the same system `kitchen create default` or `kitchen create centos` or even `kitchen create 67`. This is an example of using the Regular Expression (REGEXP) to specify an instance.

When you want to target all of the instances you can run `kitchen create`. This will create all instances. Seeing as how there is only one instance this will work well.

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

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



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

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

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

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

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
```

```
----> Starting Kitchen (v1.11.0)
----> Creating <default-centos-67>...
      Sending build context to Docker daemon 193 kB
      (skipping)
----> Finished creating <default-centos-67> (1m18.32s).
----> Converging <default-centos-67>...
$$$$$ Running legacy converge for 'Docker' Driver
      (skipping)
Synchronizing Cookbooks:
  - workstation (0.1.0)
Compiling Cookbooks...
Converging 2 resources
Running handlers:
```

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

The `workstation` cookbook should successfully apply the default recipe. If an error occurs, let's stop and troubleshoot the issues.

Instructor Note: It can take about four minutes for this task to complete on the system. During this time you could demo Test Kitchen on your local workstation using Vagrant and Virtual Box.



Lab: Converge the Recipe for Apache

We want to validate that our run-list installs correctly.

- Within the "apache" cookbook use kitchen converge for the default suite on the centos 6.7 platform.

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

Instructor Note: Allow 8 minutes to complete this exercise.

Lab: Configuring Test Kitchen for Apache

~/cookbooks/apache/.kitchen.yml

```
---
```

```
driver:
```

```
  name: docker
```

```
provisioner:
```

```
  name: chef_zero
```

```
verifier:
```

```
  name: inspec
```

```
platforms:
```

```
  - name: centos-6.7
```

Lab: Return Home and Move into the Cookbook



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

Change into the apache cookbook folder.

Lab: Converge the Cookbook



```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.11.0)
-----> Creating <default-centos-67>...
      Sending build context to Docker daemon 194 kB
      Sending build context to Docker daemon
      (skipping)
      Installing Chef
          installing with rpm...
          warning: /tmp/install.sh.23/chef-12.13.37-1.el6.x86_64.rpm: Header V4
DSA/SHA1 Signature, key ID 83ef826a: NOKEY
      (skipping)
      Synchronizing Cookbooks:
          - apache (0.1.0)
      Compiling Cookbooks...
```

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



Lab: Converge the Recipe for Apache

We want to validate that our run-list installs correctly.

- ✓ Within the "apache" cookbook use kitchen converge for the default suite on the centos 6.7 platform.

DISCUSSION



Test Kitchen

What is being tested when kitchen converges a recipe without error?

What is NOT being tested when kitchen converges the recipe without error?

Kitchen converge will create the instance if it is not already created. It will install Chef. Then it applies the recipe to the system examining each of the resources and asking them to take action to place the system into the desired state. What is being tested when kitchen converges a recipe without error?

What is NOT being tested when kitchen converges the recipe without error?

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.



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 what it was unable to check to see if the system was configured correctly -- is our instance serving up our custom home page.



The First Test

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

Objective:

- In a few minutes we'll write and execute a test that asserts that the tree package is installed when the "workstation" cookbook's default recipe is applied.

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.

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



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

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

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



Kitchen Destroy

kitchen
create

kitchen
converge

kitchen
verify

kitchen
destroy

```
$ kitchen destroy [INSTANCE|REGEXP|all]
```

Destroys one or more instances.

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.



Kitchen Test



```
$ 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 immediately destroyed. This ensures a clean instance to perform all of the steps: create; converge; and verify. 'kitchen test' completes the entire execution by destroying the instance at the end.

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



InSpec

InSpec tests your servers' actual state by executing command locally, via SSH, via WinRM, via Docker API and so on.

<http://inspec.io/>

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

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 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.

<http://inspec.io/>

Example: Is the 'tree' Package Installed?

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

I expect the package tree should be installed.

<http://inspec.io/docs/reference/resources/#id118>

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

<http://inspec.io/docs/reference/resources/#id118>

GL: Example Tests

```
~/cookbooks/workstation/test/recipes/default_test.rb
```

```
unless os.windows?  
  describe user('root') do  
    it { should exist }  
    skip 'This is an example...test.'  
  end  
end  
  
describe port(80) do  
  it { should_not be_listening }  
  skip 'This is an example... test.'  
end
```

When not on windows a user named `root` should exist.

The port 80 should not be listening for incoming connections.

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

First, we need to create a test file. The structure of the path is a convention defined by InSpec's verifier and will automatically be loaded when we run `kitchen verify`. Fortunately for us the test file has already been created when we used 'chef' to generate the workstation cookbook.

By default the test has two example expectations. The first expectation states that if the operating system is not Windows then a user, named 'root' should exist. The second expectation states that that port 80 should not be listening.

GL: Describing the Resources

```
~/cookbooks/workstation/test/recipes/default_test.rb
```

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end
```

A user named 'root'

The port 80

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

Each example starts first with a resource that we are describing. InSpec provides a number of helper methods that allow us to address particular resource.

The 'describe' method takes two parameters - the first is the resource that is being examined.

The second parameter is the block between the **do** and **end**. Within that block we can define more describe blocks that allow us to specify the state of the resource defined in the parent describe method.

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

GL: Describing the State of the Resources

```
~/cookbooks/workstation/test/recipes/default_test.rb
```

```
unless os.windows?  
  describe user('root') do  
    it { should exist }  
    skip 'This is an example...test.'  
  end  
  
  describe port(80) do  
    it { should_not be_listening }  
    skip 'This is an example... test.'  
  end
```

The user named 'root' should exist.

The port 80 should not be listening.

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

Within each describe block you may define zero or more characteristics about the resource defined in the describe.

In the first example we are stating that 'it should exist'. This refers to the the user named 'root'.

In the second example we are stating that 'it should not be listening'. This refers to the port 80 in the parent describe.

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

GL: The `skip` Reminds Us to Remove These Tests

```
~/cookbooks/workstation/test/recipes/default_test.rb
```

```
unless os.windows?  
  describe user('root') do  
    it { should exist }  
    skip 'This is an example...test.'  
  end  
end  
  
describe port(80) do  
  it { should_not be_listening }  
  skip 'This is an example... test.'  
end
```

skip will show a message in the test results.

These skips will remind you that the following expectations are only examples.

A describe can also contain a 'skip' method with a message. The 'skip' method is useful because it allows you to write notes for yourself that will appear in the output results when you execute the tests. These notes are often used to describe expectations that have not been written or in this case to remind you that the following tests were written by a generator and should be removed.

GL: Adding a New Test

~/cookbooks/workstation/test/recipes/default_test.rb

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end

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

Delete the 'skip' lines from the first two tests or delete these two tests. We can make an assumption that on CentOS there will be a root user. We can also assume that if we were to converge only the workstation cookbook that nothing should be running on port 80.

Add a new tests that states that package 'tree' should be installed on the system.



Where do Tests Live?

`workstation/test/recipes/default_test.rb`

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

This is configurable in the kitchen configuration file (.kitchen.yml) in the suites section.

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.

GL: Return Home and Move into the Cookbook



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

GL: Running the Specification



```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.11.1) ...
-----> Verifying <default-centos-67>...
      Use `~/home/chef/cookbooks/workstation/test/recipes/default` for testing
Target: ssh://kitchen@localhost:32768

    ✓ User root should exist
    ✓ Port 80 should not be listening
    ✓ System Package tree should be installed

Summary: 3 successful, 0 failures
      Finished verifying <default-centos-67> (0m1.39s).
-----> Kitchen is finished. (0m2.76s)
```

If you had not deleted the tests below, you should see 3 successful expectations met when you execute 'kitchen verify'. If you deleted the tests, you should see 1 successful expectation.

unless os.windows?

```
describe user('root') do
  it { should exist }
  skip 'This is an example...test.'
end
end
```

```
describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end
```



GL: Commit Your Work

```
$ cd ~/cookbooks/workstation  
$ git add .  
$ git status  
$ git commit -m "Add first test for default test suite"
```

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



More Tests

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

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

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

InSpec

Testing a File



InSpec can help us assert different characteristics about files on the file system. Like if it is a file, directory, socket or symlink.

InSpec can also help us assert the file's mode, owner or group or if the file is readable, writeable, or executable. InSpec is even able to verify the data contained within the file.

<http://inspec.io/docs/reference/resources/file>



In our workstation's setup recipe we also created a file. InSpec provides us resource helpers to test files and their contents as well.

<http://inspec.io/docs/reference/resources/file>

Example: The File Contains Data

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

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

<http://inspec.io/docs/reference/resources/file>

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

<http://inspec.io/docs/reference/resources/file>

Example: The File Contains Specific Content

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

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

<http://inspec.io/docs/reference/resources/file>

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

Instructor Note: InSpec uses 'its' here because it is describing a characteristic of the file object. This is common for 'file' and 'command' to retrieve a value from a particular method on the InSpec test object that is being created.

Example: The File is Owned by a Particular User

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

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

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



Lab: More Tests

- Add tests that validate the file resource
(<http://inspec.io/docs/reference/resources/file>)
- Run kitchen verify to validate the test meets the expectations that you defined
- Commit your changes

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

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

Instructor Note: The learner is not required to test all of the 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.

<http://inspec.io/docs/reference/resources/file>

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/recipes/default_test.rb

```
describe port(80) do
  it { should_not be_listening }
end

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

describe file('/etc/motd') do
  its('content') { should match(/Property of/) }
end
```

Let's review the lab.

Here we are stating: I expect the file named '/etc/motd' has some content and somewhere within that content is the value 'Property of'.

Lab: Our Assertion in a spec File

```
~/cookbooks/workstation/test/recipes/default_test.rb
```

```
describe port(80) do
  it { should_not be_listening }
end

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

describe file('/etc/motd') do
  its('content') { should match(/Property of/) }
  it { should be_owned_by 'root' }
end
```

We could have also written an additional expectation that: I expect the file named '/etc/motd' is owned by the 'root' user.

GL: Return to the Cookbook Directory



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

Change into the workstation cookbook directory.

Lab: Running the Specification



```
$ kitchen verify
```

```
----> Starting Kitchen (v1.11.1)
----> Verifying <default-centos-67>...
      Use `~/home/chef/cookbooks/workstation/test/recipes/default` for testing

Target: ssh://kitchen@localhost:32768

✓ User root should exist
✓ Port 80 should not be listening
✓ System Package tree should be installed
✓ File /etc/motd should be owned by "root"; File /etc/motd con...

Summary: 5 successful, 0 failures, 0 skipped
Finished verifying <default-centos-67> (0m1.43s).
```



Lab: Commit Your Work

```
$ cd ~/cookbooks/workstation  
$ git add .  
$ git status  
$ git commit -m "Add additional tests for default  
test suite"
```

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



Lab: More Tests

- ✓ Add tests that validate the file resource
- ✓ Run kitchen verify to validate the test meets the expectations that you defined
- ✓ Commit your changes



Testing

What questions can we help you answer?

What questions can we help you answer?



Testing Our Webserver

I would love to know that the webserver is installed and running correctly.

Objective:

- Discuss and decide what should be tested with the apache cookbook

Now let's turn our focus towards testing the apache cookbook.



Testing

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

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

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

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

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

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

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



Lab: Testing Apache

- Update the test file for the "apache" cookbook's default recipe
- Add tests that validate a working web server
<http://inspec.io/docs/reference/resources/port/>
<http://inspec.io/docs/reference/resources/command/>
- Run kitchen verify
- Commit your changes

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

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

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

<http://inspec.io/docs/reference/resources/port/>
<http://inspec.io/docs/reference/resources/command/>

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Return Home and 'cd cookbooks/apache'



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

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

Lab: What Does the Webserver Say?

```
~/cookbooks/apache/test/recipes/default_test.rb

unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example test, replace with your own test.'
  end
end

describe port(80) do
  it { should be_listening }
  skip 'This is an example test, replace with your own test.'
end
```

The default expectation defined in the 'default_test.rb' file needs to be updated as we are explicitly starting a web server to run on port 80. We can again remove the 'skip' in both of these examples and then update the port to state that it 'should be listening' instead of 'it should not be listening'.

Lab: What Does the Webserver Say?

```
~/cookbooks/apache/test/recipes/default_test.rb
```

```
describe port(80) do
  it { should be_listening }
  skip 'This is an example test, replace with your own test.'
end

describe command('curl localhost') do
  its('stdout') { should match('Hello, world') }
end
```

Besides listening on the port we can verify that the content being returned by our website is correct by running a command like 'curl'. Here we are saying: I expect the command 'curl localhost' standard out to contain within it the content 'Hello, world'.

Lab: Remove the server test file



```
$ rm test/recipes/server.rb
```



When we generated the server recipe it automatically generated for us a server test file. We do not happen to need this test file as we have defined all the expectations in the `default_test.rb` file.

Lab: Verifying the Expectations



```
$ kitchen verify
```

```
----> Starting Kitchen (v1.11.1)
----> Verifying <default-centos-67>...
      Use `/home/chef/cookbooks/apache/test/recipes/default` for testing

Target: ssh://kitchen@localhost:32769

✓ User root should exist
✓ Port 80 should be listening
✓ Command curl localhost stdout should match "Hello, world"

Summary: 3 successful, 0 failures, 0 skipped
Finished verifying <default-centos-67> (0m0.87s).
```

When we execute `kitchen verify` we see that the 3 expectations have been met.

The user named 'root' exists.

The port 80 is listening for incoming connections.

The command 'curl localhost' standard out contains 'Hello, world'.



Lab: Commit Your Work

```
$ cd ~/cookbooks/apache  
$ git add .  
$ git status  
$ git commit -m "Add tests for default test suite"
```

Again, let's commit the work.



Lab: Testing Apache

- ✓ Create a test file for the "apache" cookbook's default recipe
- ✓ Add tests that validate a working web server
- ✓ Run kitchen verify
- ✓ Commit your changes



Discussion

Why do you have to run kitchen within the directory of the cookbook?

Where would you define additional platforms?

Why would you define a new test suite?

What are the limitations of using Test Kitchen to validate recipes?

Answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

Discussion



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, InSpec.



Details About the System

Finding and Displaying Information About Our System

©2016 Chef Software Inc.



Objectives

After completing this module, you should be able to

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



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.



Details About the Node

Displaying system details in the MOTD definitely sounds useful.

Objective:

- Update the MOTD file contents, in the "workstation" cookbook, to include node details

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

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

Some Useful System Data

- IP Address
- hostname
- memory
- CPU - MHz

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

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

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

Instructor Note: The next series of steps often seem like an obvious mistake to the learners. It may be helpful to have the learners watch as you perform these steps and comment on the process.

GL: Discover the IP Address



```
$ hostname -I
```

```
172.31.8.68 172.17.42.1
```

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

`hostname -I`

GL: Discover the Host Name



```
$ hostname
```

```
banana-stand
```

GL: Discovering the Memory



```
$ cat /proc/meminfo
```

MemTotal:	502272 kB
MemFree:	118384 kB
Buffers:	141156 kB
Cached:	165616 kB
SwapCached:	0 kB
Active:	303892 kB
Inactive:	25412 kB
Active(anon):	22548 kB
Inactive(anon):	136 kB
Active(file):	281344 kB
Inactive(file):	25276 kB
Unevictable:	0 kB
Mlocked:	0 kB

GL: Discover the CPU - MHz



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

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

GL: Adding the CPU

~/cookbooks/workstation/recipes/setup.rb

```
file '/etc/motd' do
  content 'Property of ...
  IPADDRESS: 104.236.192.102
  HOSTNAME : banana-stand
  MEMORY    : 502272 kB
  CPU       : 2399.998 MHz
  '
  mode '0644'
  owner 'root'
  group 'root'
end
```

We can now add the ipaddress, hostname, memory, and cpu to the file resource's content attribute.



GL: Introducing a Change

By creating a change we have introduced risk.

Lets run our cookbook tests before we apply the updated recipe.

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

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

Instructor Note: This is important to reinforce the workflow of working with recipes. Even though this seems like a small change it important to verify every change.

GL: Change into Our Cookbook



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

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

GL: Run Our Tests



```
$ kitchen test
```

```
----> Starting Kitchen (v1.11.1)
----> Cleaning up any prior instances of <default-centos-67>
----> Destroying <default-centos-67>...
      Error response from daemon: Container
6b45fed... is not running
      Finished destroying <default-centos-67> (0m0.08s).
----> Testing <default-centos-67>
----> Creating <default-centos-67>...
      Sending build context to Docker daemon 195.1 kB
      Sending build context to Docker daemon
      Step 0 : FROM centos:centos6
      ---> 273aleca2d3a
      Step 1 : ENV container docker
      ---> Using cache
```

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

GL: Return Home and Apply workstation Cookbook



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

```
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["workstation"]  
Synchronizing Cookbooks:  
  - workstation (0.1.0)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
Converging 2 resources  
Recipe: workstation::setup  
* yum_package[tree] action install (up to date)  
* file[/etc/motd] action create  
  - update content in file /etc/motd from d100eb to 63e97f
```

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

1. So we need to return home to the parent directory of all our cookbooks.
2. Then use `chef-client` to locally apply the run list defined as: the workstation cookbook's default recipe.

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



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

```
Property of ...
```

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



Capturing System Data

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

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

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

Now that we've defined these values, let's reflect:

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

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

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



Hard Coded Values

The values that we have derived at this moment may not be the correct values when we deploy this recipe again even on the same system!

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

Data In Real Time

How could we capture this data in real-time?



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

Capturing the data in real-time on each system is definitely possible. One way would be to execute each of these commands, parse the results, and then insert the dynamic values within the file resource's content attribute. We could also figure out a way to run system commands within our recipes. Before we start down this path, we'd like to introduce you to Ohai.

Instructor Note: There are many ways within Ruby to escape out and run a system command. Someone new to Chef and Ruby may reach for those and this section is important in showing that most of the work has already been done.

Ohai!

Ohai is a tool that already captures all the data that we similarly demonstrated finding.



<http://docs.chef.io/ohai.html>

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

<http://docs.chef.io/ohai.html>

Ohai!



```
$ ohai
```

```
{  
  "kernel": {  
    "name": "Linux",  
    "release": "2.6.32-431.1.2.0.1.el6.x86_64",  
    "version": "#1 SMP Fri Dec 13 13:06:13 UTC 2013",  
    "machine": "x86_64",  
    "os": "GNU/Linux",  
    "modules": {  
      "veth": {  
        "size": "5040",  
        "refcount": "0"  
      },  
      "ipt_addrtype": {  
        "size": "5040",  
        "refcount": "0"  
      }  
    }  
  }  
}
```

All About The System



Ohai queries the operating system with a number of commands, similar to the ones demonstrated.

The data is presented in JSON (JavaScript Object Notation).

<http://docs.chef.io/ohai.html>

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

<http://docs.chef.io/ohai.html>



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.

<http://docs.chef.io/ohai.html>

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

<http://docs.chef.io/ohai.html>



The Node Object

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

<http://docs.chef.io/nodes.html#attributes>

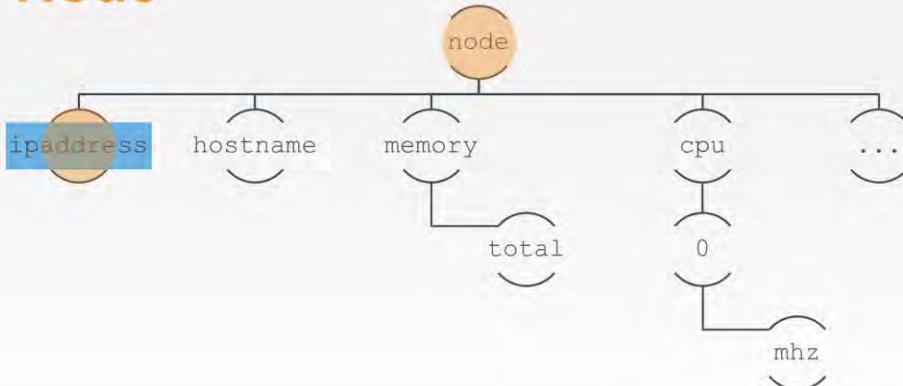
The node object is a representation of our system. It stores all these attributes found about the system. It is available within all the recipes that we write to assist us with solving the similar problems we outlined at the start.

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

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

<http://docs.chef.io/nodes.html#attributes>

The Node



IPADDRESS: 104.236.192.102

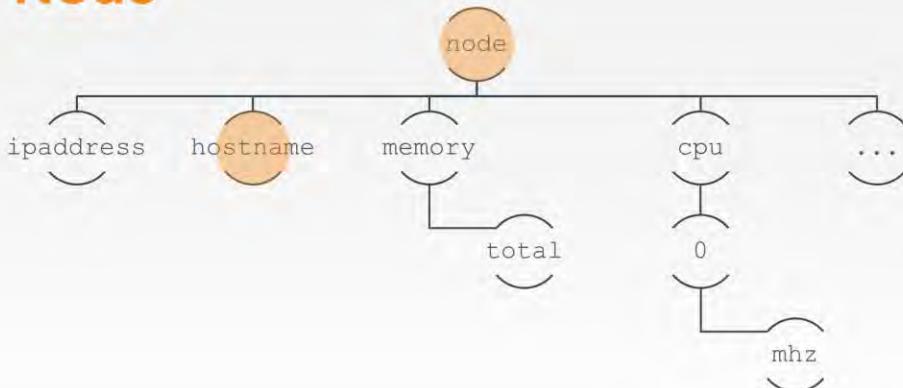
```
"IPADDRESS: #{node['ipaddress']}
```

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

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

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

The Node

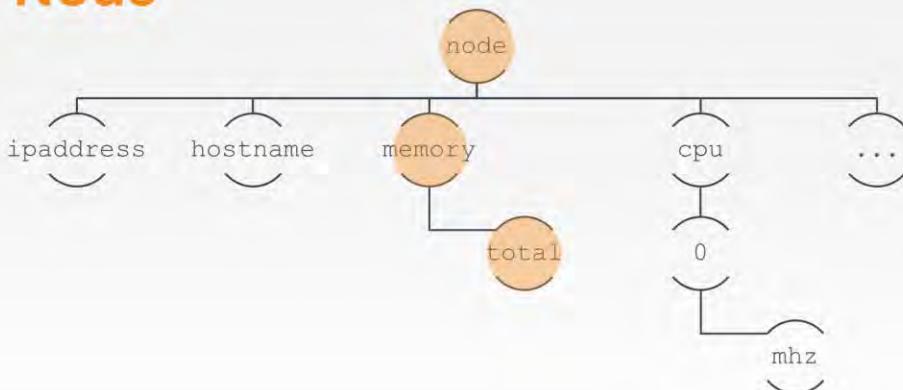


HOSTNAME: banana-stand

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

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

The Node



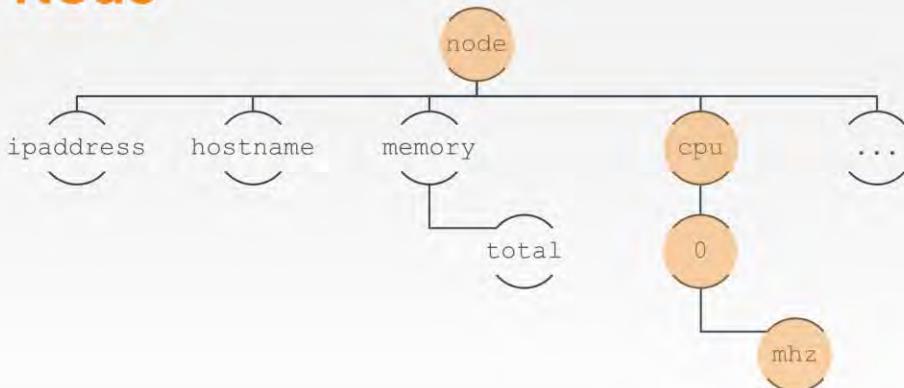
MEMORY: 502272kB

```
"Memory: #{node['memory']['total']}
```

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

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

The Node



CPU: 2399.998MHz

```
"CPU: #{node['cpu']['0']['mhz']}
```

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



String Interpolation

```
I have 4 apples
```

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

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

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

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



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.

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



String Interpolation

```
I have 4 apples
```

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

GL: Using the Node's Attributes

~/cookbooks/workstation/recipes/setup.rb

```
# ... PACKAGE RESOURCES ...
file '/etc/motd' do
  content "Property of ...
  IPADDRESS: #{node['ipaddress']}
  HOSTNAME : #{node['hostname']}
  MEMORY   : #{node['memory']['total']}
  CPU       : #{node['cpu'][0]['mhz']}
"
  mode '0644'
  owner 'root'
  group 'root'
end
```

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

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



GL: Verify the Changes

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

Again we have created a change.

Move into the workstation cookbook's directory. Verify the changes we made to the workstation cookbook's default recipe with kitchen. Return to the home directory. Use 'chef-client' to locally apply the workstation cookbook's default recipe.

Instructor Note: Allow 8 minutes to complete this exercise.

Lab: Test the Workstation's Default Recipe



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

```
----> Testing <default-centos-67>  
----> Creating <default-centos-67>...  
    Sending build context to Docker daemon 195.1 kB  
    Sending build context to Docker daemon  
Step 0 : FROM centos:centos6  
    --> 273aleca2d3a  
Step 1 : ENV container docker  
    --> Using cache  
    --> 9beebd0e6677  
Step 2 : RUN yum clean all  
    --> Using cache
```

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

Lab: Apply the Workstation's Default Recipe



```
$ cd ~  
$ sudo chef-client --local-mode -r "recipe[workstation]"  
  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["workstation"]  
Synchronizing Cookbooks:  
  - workstation (0.1.0)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
Converging 2 resources  
Recipe: workstation::setup  
  * yum_package[tree] action install (up to date)  
  * file[/etc/motd] action create (up to date)
```

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



GL: Verify the Changes

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



Changes Mean a New Version

Let's bump the version number and check in the code to source control.

Objective:

- Update the version of the "workstation" cookbook
- Commit the changes to the "workstation" cookbook to version control

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



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

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

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

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



Semantic Versions

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

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

<http://semver.org>

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

Major versions are often large rewrites or large changes that have the potential to not be backwards compatible with previous versions. This might mean adding support for a new platform or a fundamental change to what the cookbook accomplishes. Minor versions represent smaller changes that are still compatible with previous versions. This could be new features that extend the existing functionality without breaking any of the existing features. And finally Patch versions describe changes like bug fixes or minor adjustments to the existing documentation.

<http://semver.org>

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?

GL: Update the Cookbook Version

~/cookbooks/workstation/metadata.rb

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

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

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

GL: Commit Your Work



```
$ cd ~/cookbooks/workstation  
$ git add .  
$ git status  
$ git commit -m "Release version 0.2.0"
```



Lab: Node Details in the Webserver

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

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

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

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Apache Recipe

~/cookbooks/apache/recipes/server.rb

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

...

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

Lab: Test the Apache Cookbook's Default Recipe



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

```
----> Starting Kitchen (v1.11.1)  
----> Cleaning up any prior instances of <default-centos-67>  
----> Destroying <default-centos-67>...  
      Error response from daemon: Container  
2118ec3c6ae4833733a4350fe9880342cf7d9fff5a88d1c39eda660ecd3271c4 is not running  
      Finished destroying <default-centos-67> (0m0.10s).  
----> Testing <default-centos-67>  
----> Creating <default-centos-67>...  
      Sending build context to Docker daemon 195.6 kB  
      Sending build context to Docker daemon  
      Step 0 : FROM centos:centos6  
      ---> 273aleca2d3a
```

Lab: Run chef-client to Apply the Apache Cookbook



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

```
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["apache"]  
Synchronizing Cookbooks:  
  - apache (0.1.0)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
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 158cae
```

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

Lab: Update the Cookbook Version

```
~/cookbooks/apache/metadata.rb
```

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

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

Lab: Commit Your Work



```
$ cd ~/cookbooks/apache  
$ git add .  
$ git status  
$ git commit -m "Release version 0.2.0"
```



Lab: Node Details in the Webserver

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

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

Congratulations. You have successfully demonstrated the entire development workflow.



Discussion

What is the major difference between a single-quoted string and a double-quoted string?

How are the details about the system available within a recipe?

How does the version number help convey information about the state of the cookbook?

Answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

Discussion



Q&A

What questions can we help you answer?

- Ohai
- Node Object
- Node Attributes
- String Interpolation
- Semantic Versions

With that we have added all of the requested features.

What questions can we help you answer?

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



Desired State and Data

Extracting the Content for Clarity

©2016 Chef Software Inc.



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



Cleaner Recipes

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

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

Apache Recipe

~/cookbooks/apache/recipes/server.rb

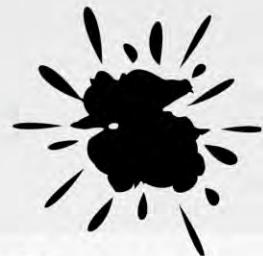
```
package 'httpd'

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

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

What if new changes are given to us for the web 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.

Double Quotes Close Double Quotes



Double quoted strings are terminated by double quotes.

```
"<h1 style="color: red;">Hello, World!</h1>"
```



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

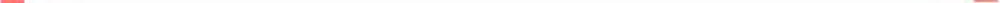
Double-quoted strings are terminated by double-quotes so if any of the text that we paste into this content field has double quotes it is going to have to be escaped.



Backslash

We can use double-quotes as long as we prefix them with a backslash.

```
"<h1 style=\"color: red;\">"Hello, World!</h1>"
```



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



Backslash

Backslashes are reserved characters. So to use them you need to use a backslash.

```
"Root Path: \\\"
```



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

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

Backslash

Backslashes are reserved characters. So to use them you need to use a backslash.

```
"Root Path: \\"
```



Unexpected Formatting

```
file '/etc/motd' do
  content 'This is the first line of the file.
           This is the second line. If I try and line it up...
           Don't even think about pasting ASCII ART in here!
           '
end
```

This is the first line of the file.

This is the second line. If I try and line
it up...

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

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

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

Instructor Note: The Message of the Day file is not a white-space important file. Other configuration files that could be managed with Chef may be white-space important.



Copy Paste

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

This could sound like a bug waiting to happen.

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



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 you wouldn't need to escape any common characters.

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



GL: Cleaner Recipes

Adding the node attributes to our recipes did make it harder to read.

Objective:

- Decide which resource will help us address this issue

GL: Let's Check the Docs...



Use the file resource to manage files directly on a node.

Use the **cookbook_file** resource to copy a file from a cookbook's `/files` directory. Use the **template** resource to create a file based on a template in a cookbook's `/templates` directory. And use the **remote_file** resource to transfer a file to a node from a remote location.

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

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

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

Lets start with the `remote_file` resource.

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



remote_file

Use the **remote_file** resource to transfer a file from a remote location using file specificity. This resource is similar to the file resource.

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

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

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

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



cookbook_file

Use the **cookbook_file** resource to transfer files from a sub-directory of COOKBOOK_NAME/files/ to a specified path located on a host that is running the chef-client.

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

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

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

Demo: cookbook_file's Source Match Up

```
$ tree cookbooks/apache/files/default  
files/default  
└── index.html  
0 directories, 1 file
```

```
cookbook_file '/var/www/index.html' do  
  source 'index.html'  
end
```

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

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



Template

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

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/resource_template.html

Let's explore templates.

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

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

Demo: Template File's Source Matches Up

```
$ tree cookbooks/apache/templates/default
templates/default
└── index.html.erb
0 directories, 1 file

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

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.



Template

To use a template, two things must happen:

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

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

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

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



GL: Cleaner Apache Recipe

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

Objective:

- 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

So our objective is clear. We need to use a template resource and create a template 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:
  app           Generate an application repo
  cookbook      Generate a single cookbook
  recipe        Generate a new recipe
  attribute     Generate an attributes file
  template      Generate a file template
  file          Generate a cookbook file
  lwrp          Generate a lightweight resource/provider
  repo          Generate a Chef policy repository
  policyfile    Generate a Policyfile for use with the install/push commands
                 (experimental)
```

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

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

Let's use help to review the command again. And let's ask for help about the 'generate' subcommand.

GL: What Can chef generate template Do?



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

```
Usage: chef generate template [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
      ...
      -a, --generator-arg KEY=VALUE    Use to set arbitrary attribute KEY to
      VALUE in the
      -I, --license LICENSE            all_rights, apache2, 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
```

Finally 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.

GL: Use chef to Generate a Template



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

```
Recipe: code_generator::template  
* directory[cookbooks/apache/templates/default] action create  
- create new directory cookbooks/apache/templates/default  
* 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)
```

GL: Lets Look at the Template File



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

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

1 directory, 1 file
```



Cleaner Recipes

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

Objective:

- ✓ 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

ERB

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

Ruby code can be embedded using expressions and statements.

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

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/resource_template.html#using-templates

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 %>
```

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.

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

Text Within an ERB Template

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

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

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

Text Within an ERB Template

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

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

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

RECIPE



The Angry Squid

<% =

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

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

GL: Move Our Source to the Template

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

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

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

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

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.



Cleaner Recipes

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

Objective:

- ✓ 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

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

GL: Remove the Existing Content Attribute

~/cookbooks/apache/recipes/server.rb

```
package 'httpd'

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

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

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

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

GL: Change File Resource to a Template Resource

~/cookbooks/apache/recipes/server.rb

```
package 'httpd'

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

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

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

So it's time to change the file resource to a template resource so that it can use the template file that we have defined. You could just change `file` to `template` in this case.

What to Specify as the Source?

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

```
package 'httpd'

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

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

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

GL: Viewing the Partial Path to the Template



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

```
cookbooks/apache/templates
```

```
└── default
```

```
    └── index.html.erb
```

```
1 directories, 1 file
```

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

And we see the filepath index.html.erb.

Instructor Note: The default folder denotes that we want to use this file for all platforms.

GL: Update the Source Attribute

~/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
```

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



Cleaner Recipes

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

Objective:

- ✓ 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



Lab: Update the Version

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

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

Instructor Note: Allow 8 minutes to complete this exercise.

Lab: Test the Cookbook



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

```
----> Starting Kitchen (v1.11.1)  
----> Cleaning up any prior instances of <default-centos-67>  
----> Destroying <default-centos-67>...  
      Finished destroying <default-centos-67> (0m0.00s).  
----> Testing <default-centos-67>  
----> Creating <default-centos-67>...  
      Sending build context to Docker daemon 197.1 kB  
      Sending build context to Docker daemon  
      Step 0 : FROM centos:centos6  
      ---> 273a1eca2d3a
```

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

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

Lab: Change Directories and Apply the Cookbook



```
$ cd ~  
$ sudo chef-client --local-mode -r "recipe[apache]"  
  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["apache"]  
Synchronizing Cookbooks:  
  - apache (0.1.0)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
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 158cae to 43191a
```

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

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

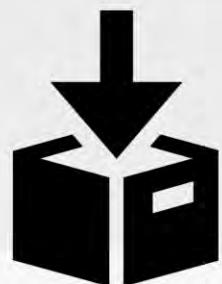
Lab: Update the Cookbook's Patch Number

~/cookbooks/apache/metadata.rb

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

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

Lab: Commit the Changes



```
$ cd ~/cookbooks/apache  
$ git add .  
$ git status  
$ git commit -m "Update default recipe to use  
template"
```



Lab: Update the Version

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



Lab: Use the Template

For the "workstation" cookbook:

- Use chef generate to create a template named "motd.erb".
- Copy the source attribute from the file named '/etc/motd' into the template file "motd.erb"
- Remove a resource: The file named '/etc/motd'
- Add a resource: The template named '/etc/motd' is created with the source 'motd.erb'
- Use kitchen to test it and chef-client to locally apply the default recipe.

It's time to do that again--this time for the workstation cookbook.

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

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

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Return Home and Generate the Template



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

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

Lab: Copy the Existing Source into the Template

~/cookbooks/workstation/templates/motd.erb

Property of ...

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

Lab: Update the motd.erb to Use ERB

~/cookbooks/workstation/templates/motd.erb

Property of ...

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

Replace all the string interpolation with ERB tags.

Lab: Remove the file Resource

~/cookbooks/workstation/recipes/setup.rb

```
# ... PACKAGE RESOURCES ...

file '/etc/motd' do
  content "Property of ...
  IPADDRESS: #{node['ipaddress']}
  HOSTNAME : #{node['hostname']}
  MEMORY    : #{node['memory']['total']}
  CPU       : #{node['cpu'][0]['mhz']}
"
  mode '0644'
  owner 'root'
  group 'root'
end
```

Remove the file resource from the setup recipe.

Lab: Replace it with the Template Resource

~/cookbooks/workstation/recipes/setup.rb

```
# ... PACKAGE RESOURCES ...

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

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

Lab: Test the Cookbook



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

```
----> Starting Kitchen (v1.11.1)  
----> Cleaning up any prior instances of <default-centos-67>  
----> Destroying <default-centos-67>...  
      Finished destroying <default-centos-67> (0m0.00s).  
----> Testing <default-centos-67>  
----> Creating <default-centos-67>...  
      Sending build context to Docker daemon 197.1 kB  
      Sending build context to Docker daemon  
      Step 0 : FROM centos:centos6  
      ---> 273a1eca2d3a
```

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

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

Lab: Change Directories and Apply the Cookbook



```
$ cd ~  
$ sudo chef-client --local-mode -r "recipe[workstation]"  
  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["workstation"]  
Synchronizing Cookbooks:  
  - workstation (0.1.0)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
Converging 2 resources  
Recipe: workstation::setup  
  * yum_package[tree] action install (up to date)  
  * template[/etc/motd] action create  
    - update content in file /etc/motd from e8c4cc to fb0ae3
```

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

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



Lab: Use the Template

For the "workstation" cookbook:

- ✓ Use chef generate to create a template named "motd.erb".
- ✓ Copy the source attribute from the file named '/etc/motd' into the template file "motd.erb"
- ✓ Remove a resource: The file named '/etc/motd'
- ✓ Add a resource: The template named '/etc/motd' is created with the source 'motd.erb'
- ✓ Use kitchen to test it and chef-client to locally apply the default recipe.



Lab: Update the Version

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

With everything working it is time to update the patch version and commit the changes.

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: Update the Cookbook's Patch Number

~/cookbooks/workstation/metadata.rb

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

Update the patch version number for the workstation cookbook.

Lab: Commit the Changes



```
$ cd ~/cookbooks/workstation  
$ git add .  
$ git status  
$ git commit -m "Update default recipe to use  
template"
```



Lab: Update the Version

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

Updating the version and committing your work is an essential part of the workflow.

Discussion



Discussion

What is the benefit of using a template over defining the content within a recipe? What are the drawbacks?

What do each of the ERB tags accomplish?

Answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

Discussion



Q&A

What questions can we help you answer?

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

What questions can we help you answer?

Generally or specifically about resources, templates, and ERB.



Workstation Installation

Configuring Your Laptop as a Workstation

©2016 Chef Software Inc.



Objectives

After completing this module, you should be able to

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

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

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



Installing the ChefDK

Installing the tools on your system

Objective:

- Install the ChefDK
- Open a Terminal / Command Prompt
- Execute a series of commands to ensure everything is installed
- Download a repository of cookbooks
- Install a text editor (optional)

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

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

After that you can optionally download a number of other tools that will help you in your journey using Chef. For example a text editor.

Let's get started.



ChefDK

The ChefDK contains tools like chef-apply, chef-client, and kitchen.

You can find the ChefDK to download at the website downloads.chef.io. Be sure to download ChefDK version 0.17.17

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

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

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

Instructor Note: Prior to attending this course they may have received correspondence that informed them to setup the ChefDK on their systems. It is possible that they did not and this slides acts as a good reminder to ensure that they have the necessary tools before continuing on to the next section. **IMPORTANT:** This course was tested against ChefDK version 0.17.17. If you use a different version, the exercises and labs may not work properly.



GL: Download the ChefDK

ChefDK is a tool chain built on top of the Ruby programming language.

The ChefDK installer does not install any particular graphical-user-interface—installs CLI instead

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

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

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

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

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



GL: Installing ChefDK

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

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

Instructor Note: They should install the ChefDK locally here.

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

GL: Installing ChefDK



Follow the ChefDK installation wizard's instructions. It could take a few minutes to install ChefDK.

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

GL: ChefDK PowerShell

For Windows users, ChefDK installs a Chef version of PowerShell on your desktop that ensures everything is properly set up for the shell.

You should run all the commands in the rest of the labs from within this ChefDK shell.



Mac users' native terminals will be properly set up by ChefDK.



Lab: Run All These Commands

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

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

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

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

git

Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005, and has since become the most widely adopted version control system for software development.



<http://git-scm.com/downloads>

We used git on the remote workstations. Chef and the Chef community uses git to manage the source code that we write. It is not required that you install git or use git when working with source code. However, we strongly recommend you use a version control tool and if you have not selected one you can use the copy of git that is included in the latest ChefDK.

<http://git-scm.com/downloads>

Instructor Note: It is not necessary that the learner install git. In some situations they may not be even allowed to install it. This again may be a good point to remind them that working with Chef and the Chef community they will likely come in contact with git again and so it is useful to have this tool to be able to participate in reading the source code and contributing fixes, changes, and features.

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.

ATOM Editor

Atom is modern, approachable, and hackable to the core. We can't wait to see what you build with it.



<https://atom.io>

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

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

<https://atom.io>

Visual Studio Code



Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs.

<https://code.visualstudio.com>

Visual Studio Code is another editor in a similar class as Atom. It is cross-platform and provides comparable features.

<https://code.visualstudio.com>



The Chef Server

A Hub for Configuration Data

©2016 Chef Software Inc.



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

Objectives

After completing this module, you should be able to

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

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



More Web Servers?

More easily manage multiple nodes

Objective:

- Create a Hosted Chef Account
- Upload your cookbooks to the Hosted Chef Server
- Add a new node as a managed node

Managing an Additional System

To manage another system, you would need to:

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

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

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

Instructor Note: This exercise is to show the value of using a Chef Server with regard to managing multiple systems. It can be done with the group, with individuals, or done in pairs.

Managing Additional Systems

Installing the Chef tools, transferring the apache cookbook, and applying the run list is not terribly expensive.

- Chef provides a one-line curl install.
- You could use **git** to clone the repository from a common **git** repository.
- Applying the run list.

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

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

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

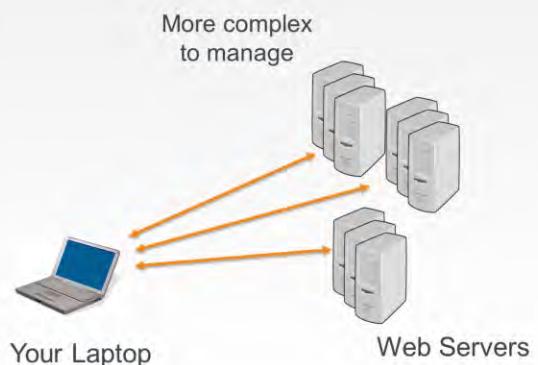
Then applying the run list requires the execution of a command on that system.

Managing Additional Systems

Now



Future



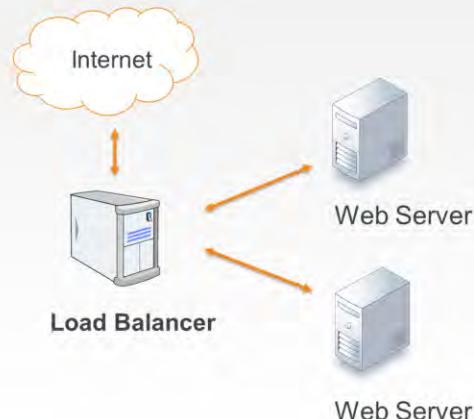
So the overall time required to setup a new instance is not a massive time investment. This manual process will definitely take its toll when requirements demand you manage more than a few additional nodes.

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

Managing User Traffic

A load balancer can forward incoming user web requests to other nodes.



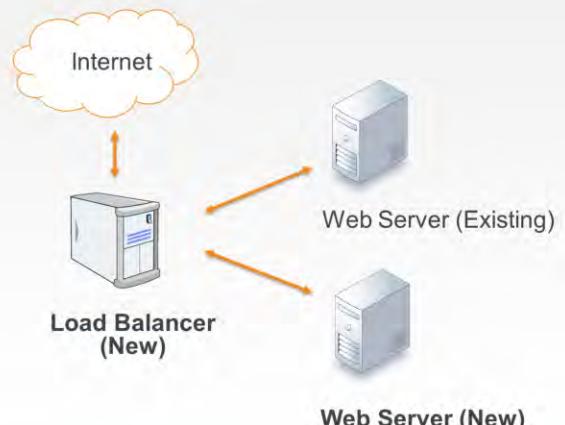
Let's change topics for a moment to managing user web traffic.

In addition to the complexities of configuring and managing multi-server infrastructure, such as web servers, you also need to develop a way to route incoming traffic to each of those web servers and other nodes. There are many ways that you can route the traffic from one node to a group of similar nodes. This can be done with services by some of the major cloud providers or it can be done with another instance running as a load balancer. A load balancer allows us to receive incoming requests and forward those requests to other nodes. A load balancer allows us to receive incoming requests and forward those requests to other nodes.

Instructor Note: The preceding three slides covered the complexity of configuring and managing multi-server infrastructure. This slide and page is now talking about managing user **web traffic**, via a load balancer.

Managing User Traffic

Today you will set up a new load balancer that will direct web requests to similarly-configured nodes.



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

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

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

Steps to Set up Load Balancer and Web Servers

Web Server

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

Load Balancer

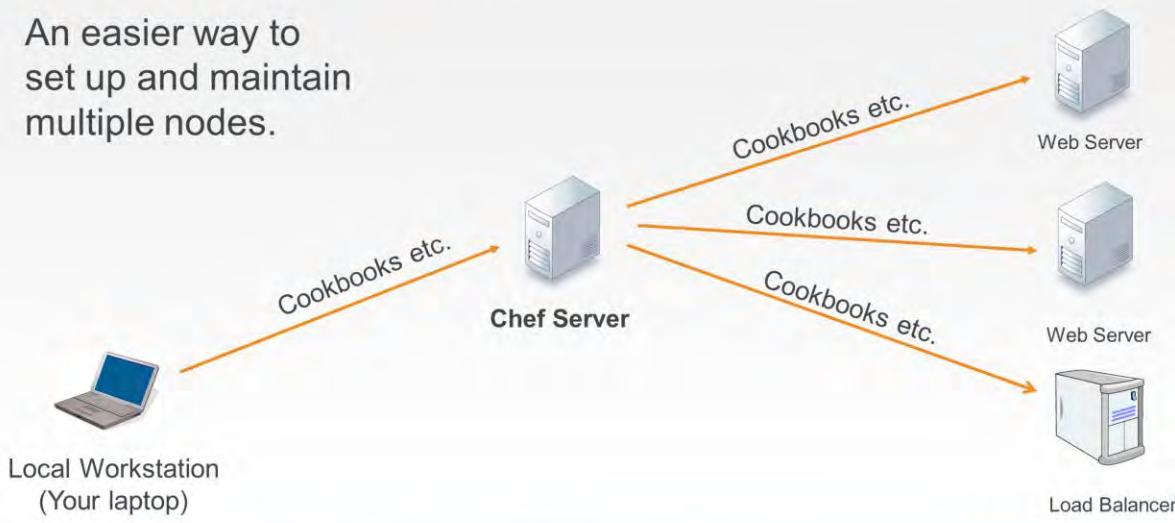
1. Create the haproxy (load balancer) cookbook
2. Provision the instance
3. Install Chef
4. Copy the haproxy cookbook
5. Apply the cookbook

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

Instructor Note: The left side is setting up a new web server that is like the one they created yesterday. The right is new work that the learner will be accomplishing today. Note that the haproxy is actually a load balancer.

The Chef Server

An easier way to set up and maintain multiple nodes.



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

The Chef Server is designed to help us manage multiple nodes in this situation. The Chef Server acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by 'chef-client'. Nodes, such as web servers, load balancers, etc., use 'chef-client' to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server). In a production environment, the 'chef-client' runs in an automated mode—it polls the Chef Server for updates at set intervals and then applies any configuration changes. This scalable approach distributes the configuration effort throughout the organization.

Flavors of Chef Server

**Open Source
Chef Server**

Chef Server
(Support +
Premium
Features)

**Multi-tenant
Hosted Chef Server**

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



GL: Hosted Chef

More easily manage multiple nodes

Objective:

- Create a Hosted Chef Account
- Upload your cookbooks to the Hosted Chef Server
- Add a new node as a managed node

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

GL: Signing Up for a Hosted Chef Account

Steps

1. Navigate to <https://manage.chef.io/signup>
2. Fill out the form as indicated in this image using your name and a valid email address and then click **Get Started**.

The screenshot shows the 'Start your free trial of Hosted Chef' page. It features fields for Full Name (Jane Smith), Company (Chef), Email (janesmith@chef.io), and Username (janesmith). A checkbox for agreeing to the Terms of Service and Master License and Services Agreement is checked. A large orange 'Get Started' button is at the bottom, with a cursor pointing to it.



To get started with Hosted Chef Server, visit <https://manage.chef.io/signup> and sign up for a Hosted Chef Account.

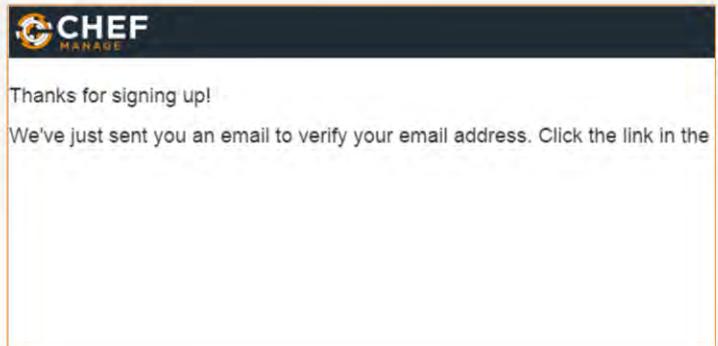
Instructor Note: Learners that already have an account can login instead of creating an account. (Using the 'Already have an account? Click here to sign in link'.)

The learner's account may be tied to an production organization. The learner can create a new organization with their existing account. If there are still concerns they can create a new login with a unique email address.

GL: Signing Up for a Hosted Chef Account

Steps

- When prompted, open the email just sent to you and click the link in the email to finish the creation of your account.



GL: Signing Up for a Hosted Chef Account

Steps

4. Enter a password when prompted and then click **Create User**.

You should write down your password in case you forget it.

The screenshot shows a web application interface for 'CHEF MANAGE'. At the top, there's a dark header bar with the 'CHEF' logo and the word 'MANAGE'. Below this, the main content area has a white background. A large orange banner at the top says 'Email Verification Successful'. Below the banner, there's a message: 'Thank you for verifying your email address! Please enter the password you'd like to use below and submit the form to complete the creation of your account.' Underneath the message is a password input field labeled 'Password' with several dots in it. To the right of the input field is a small icon of a clipboard with a pencil. At the bottom of the form is an orange button labeled 'Create User' with a white hand cursor icon pointing to it.

GL: Signing Up for a Hosted Chef Account

Steps

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



GL: Signing Up for a Hosted Chef Account

Steps

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

The screenshot shows a modal dialog titled "Create Organization". It has two text input fields: "Full Name (example: Chef, Inc.)" containing "Janesmith Org" and "Short Name (example: chef)" containing "janesorg". At the bottom right are two buttons: "Cancel" and "Create Organization", with the "Create Organization" button being highlighted by a mouse cursor.

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

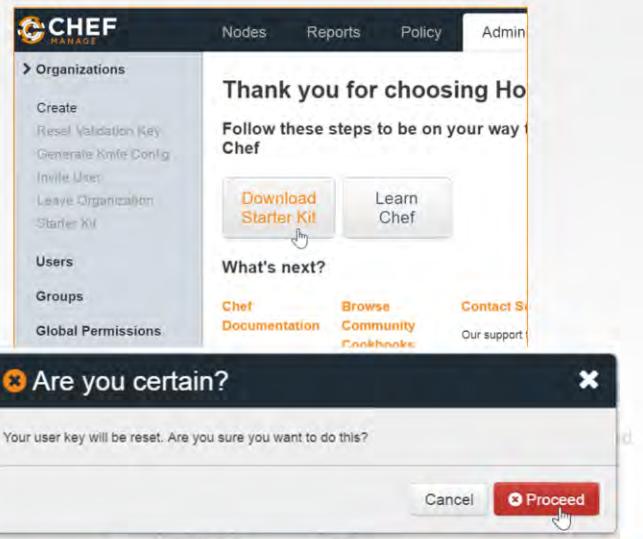
All of the cookbooks, instances and other configuration details that you manage with Chef will be stored on the Chef Server for this particular organization. No other organization will have access to it.

GL: Signing Up for a Hosted Chef Account

Steps

- From the resulting page, click **Download Starter Kit** and then click **Proceed** when prompted.

A chef-starter zip file should download to your laptop.



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

Instructor Note: By far the easiest way to get setup with Managed Chef Server is download the Starter Kit. The most important pieces of the starter kit are found in a hidden directory (".`chef`") which contains the organization key, user key, and organization configuration file.

GL: Signing Up for a Hosted Chef Account

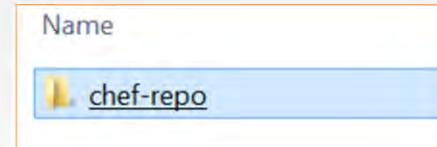
Steps

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

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

Mac: /home/username/chef-repo

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



Instructor Note: The reason that spaces are not suggested is that Ruby tools have a hard time with file paths that contain spaces.



GL: Download a Repository

A repository containing a similar copy of the work you did previously in this course can be downloaded from here:

<https://github.com/chef-training/chef-essentials-repo>

The cookbooks that were created during the first modules can be found here:
<https://github.com/chef-training/chef-essentials-repo>

These are not the exact cookbooks that you created but ones that have been completed with additional comments and details added.

Instructor Note: A learner may want to use the exact copy of the cookbooks that they developed. You may need to coordinate with the learners on using git or other methods to retrieve those cookbooks from those remote workstations.

GL: Download the Repository

The cookbooks created after the first day of completing the Chef DK Fundamentals — Edit

45 commits 47 branches 0 releases 1 contributor

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

Latest commit f7cce17 on Mar 1 burtlo Merge branch '07-03-workstation_cookbook_template_resource' into master

cookbooks Update workstation cookbook version to 0.2.1
README.md Adding README to describe how to use this REPO
hello.rb Update hello to include attributes and action
README.md

Clone with HTTPS Use Git or checkout with SVN using the web URL
<https://github.com/chef-training/chef-essentials> Use SSH

[Open in Desktop](#) [Download ZIP](#)

Chef Essentials Linux - Repository



©2016 Chef Software Inc.

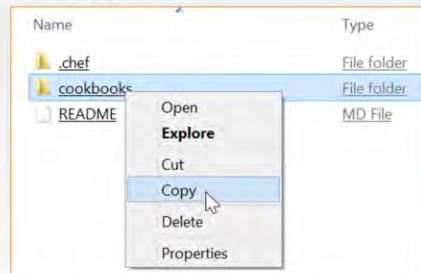
9-21

You may clone the repository or download the zip file. Both of those links can be found by clicking the "Clone or Download" button.

GL: Paste the cookbooks Folder

Steps

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



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

Important: If you had an existing chef-repo prior to class that you want to preserve, save a copy of your old cookbooks folder before pasting the new one into your chef-repo.

Instructor Note: The Starter Kit only contained a starter cookbook that has no value to the learner.

GL: Navigate to the chef-repo



```
$ cd ~/chef-repo
```

The starter kit contains the configuration to reach the Chef Server and your credentials to validate the communication between your workstation and the Chef Server.

To verify the connection with the Chef Server you will need to run commands within the repository you downloaded.

Open a terminal or command prompt and navigate to the chef-repo directory.

knife

knife is a command-line tool that provides an interface between a local chef-repo and the Chef Server.



knife is a command-line tool that allows us to request and send information to the Chef Server.

knife helps users manage nodes, cookbooks, roles, environments, and more. knife does this through a series of sub-commands.

GL: knife --help



```
$ knife --help
```

```
Available subcommands: (for details, knife SUB-COMMAND --help)

** BOOTSTRAP COMMANDS **
knife bootstrap FQDN (options)
knife bootstrap windows ssh FQDN (options)
knife bootstrap windows winrm FQDN (options)

** CLIENT COMMANDS **
knife client bulk delete REGEX (options)
knife client create CLIENT (options)
knife client delete CLIENT (options)
knife client edit CLIENT (options)
```

You can look at all the commands with 'knife –help'.

This will display all the sub-commands available. In your case you want to verify that the client list contains a single entry so you need to look for help for the specific command 'knife client --help'.

GL: knife client --help



```
$ knife client --help
```

```
Available client subcommands: (for details, knife SUB-COMMAND --help)

** CLIENT COMMANDS **

knife client bulk delete REGEX (options)
knife client create CLIENT (options)
knife client delete CLIENT (options)
knife client edit CLIENT (options)
knife client list (options)
knife client reregister CLIENT (options)
knife client show CLIENT (options)
```

This will give us an even smaller subset of the commands related specifically to asking the Chef Server about client information. A general command is the list command which will output all the clients that the Chef Server currently maintains.

GL: knife client list



```
$ knife client list
```

```
ORGNAME-validator
```

For your Chef Server account there should be a single client that is the organization name-validator. This is a special key that has access to the Chef Server. The important thing is that the result does not contain an error with the configuration or authenticating with the Chef Server.

If you receive an error ensure that you: typed the command correctly; executed the command within the chef repository directory; are connected to the internet and not blocking ssl connections from your own system's proxy servers or virtual private networks; and have a .chef directory, within the chef repository, which contains the knife configuration file (knife.rb), personal key, and organizational key.



Hosted Chef

More easily manage multiple nodes

Objective:

- ✓ Create a Hosted Chef Account
- Upload your cookbooks to the Hosted Chef Server
- Add a new node as a managed node

With all that complete, you are now able to communicate with the Chef Server. At this point we will refer to the system in front of you, with the chef repository, the configuration, and the keys installed as your workstation.

When working with Chef with a Chef Server, the workstation is the location where you will compose your cookbook code. When that code is complete, you will then upload it to the Chef Server.

GL: knife cookbook --help



```
$ knife cookbook --help
```

```
** COOKBOOK COMMANDS **  
knife cookbook bulk delete REGEX (options)  
knife cookbook create COOKBOOK (options)  
knife cookbook delete COOKBOOK VERSION (options)  
knife cookbook download COOKBOOK [VERSION] (options)  
knife cookbook list (options)  
knife cookbook metadata COOKBOOK (options)  
knife cookbook metadata from FILE (options)  
knife cookbook show COOKBOOK [VERSION] [PART] [FILENAME] (options)  
knife cookbook test [COOKBOOKS...] (options)  
knife cookbook upload [COOKBOOKS...] (options)
```

Similar to asking the Chef Server about the list of available clients, you can also ask for information about cookbooks. You can find all the commands related to the cookbooks subcommand by running `knife cookbook --help` .

Similar to the list of clients, you can examine a list of cookbooks.

GL: knife cookbook list



```
$ knife cookbook list
```



Running this command will return the cookbooks currently uploaded to the Chef Server. The empty response should come as no surprise.

You want to change that. So you are going to upload each of your cookbooks to the Chef Server.

GL: Change to the cookbooks/apache Directory



```
$ cd cookbooks/apache
```

To upload a cookbook to the Chef Server you need to be within the directory of the cookbook. Let us start with the apache cookbook. Change directory into the apache cookbook directory which is within the cookbooks directory.

Berkshelf

Berkshelf is a cookbook management tool that allows us to upload your cookbooks and all of its dependencies to the Chef Server.



<http://berkshelf.com>

To upload the cookbook you will need to use another tool called Berkshelf.

Berkshelf is a cookbook management tool that allows us to upload your cookbooks and all of its dependencies to the Chef Server. In this instance, your current cookbooks have no dependencies, but in the future when they do, Berkshelf will assist you in ensuring those are all uploaded.

<http://berkshelf.com>

GL: Run berks --help



```
$ berks --help
```

Commands:

```
berks apply ENVIRONMENT      # Apply version locks from Berksfile.lock to a ...
berks contingent COOKBOOK    # List all cookbooks that depend on the given c...
berks cookbook NAME [PATH]   # Create a skeleton for a new cookbook
berks help [COMMAND]         # Describe available commands or one specific c...
berks info [COOKBOOK]        # Display name, author, copyright, and dependen...
berks init [PATH]            # Initialize Berkshelf in the given directory
berks install                # Install the cookbooks specified in the Berksfile
berks list                   # List cookbooks and their dependencies specifi...
berks outdated [COOKBOOKS]   # List dependencies that have new versions avai...
berks package [PATH]          # Vendor and archive the dependencies of a Berk...
berks search NAME            # Search the remote source for cookbooks matchi...
berks shelf SUBCOMMAND       # Interact with the cookbook store
berks show [COOKBOOK]         # Display the path to a cookbook on disk
```

Berkshelf is a command-line tool that you can ask to see available the commands.

To see full command descriptions you can include the berks sub commands. For example: `berks apply ENVIRONMENT --help`

GL: Run berks install



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using apache (0.2.1) from source at .
```

Berkshelf is used on a per-cookbook basis. As dependencies are often per cookbook you'll need to change into the directory of the cookbook.

You should install any dependencies that your cookbook might have. Again, in this instance there are no dependencies external to this cookbook but Berkshelf ensures that this is the case when it runs the 'berks install' command.

You'll see that it finds the current cookbook within your current directory, it contacts the Supermarket for any external dependencies, and then ...

GL: See the Berksfile.lock



```
$ ls -al (or ls -Force if using Powershell)
```

```
drwxr-xr-x 7 chef chef 4096 Aug 27 18:44 .
drwxr-xr-x 4 chef chef 4096 Aug 27 16:17 ..
drwxr-xr-x 8 chef chef 4096 Aug 27 16:07 .git
-rw-r--r-- 1 chef chef 126 Aug 27 15:46 .gitignore
drwxr-xr-x 3 chef chef 4096 Aug 27 18:45 .kitchen
-rw-r--r-- 1 chef chef 183 Aug 27 18:44 .kitchen.yml
-rw-r--r-- 1 chef chef 47 Aug 27 15:46 Berksfile
-rw----- 1 chef chef 77 Aug 27 18:45 Berksfile.lock
-rw-r--r-- 1 chef chef 54 Aug 27 15:46 README.md
-rw-r--r-- 1 chef chef 974 Aug 27 15:46 cheffignore
-rw-r--r-- 1 chef chef 198 Aug 27 15:46 metadata.rb
drwxr-xr-x 2 chef chef 4096 Aug 27 16:34 recipes
```

...it completes by writing a Berksfile.lock to the file system.

The Berksfile.lock is a receipt of all the cookbooks and dependencies found at the exact moment that you ran 'berks install'.

GL: See the Contents of the Berksfile.lock



```
$ cat Berksfile.lock
```

```
DEPENDENCIES
```

```
apache
  path: .
  metadata: true
```

```
GRAPH
```

```
apache (0.2.1)
```

This lock file is useful to ensure that in the future you use the same dependencies when working with the cookbook.

GL: Upload the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded apache (0.2.1) to: 'https://api.opscode.com:443/organizations/ORG'
```

With the dependencies accounted for, it is time to upload the to the Chef Server. This is another sub-command that Berkshelf provides called 'upload'. Run the command to upload the apache cookbook to the Chef Server.

GL: Display Cookbooks within Your Org



```
$ knife cookbook list
```

apache	0.2.1
--------	-------

When that is complete you can return to the cookbook command that allows you to display the cookbooks within your organization by running this command. This will show you that the Chef Server has the apache cookbook that you have uploaded.

Note: You can also use the -a flag to see all versions of a cookbook. For example, if you had multiple versions of a cookbook:

```
$ knife cookbook list -a
```

apache	0.3.1 0.3.0 0.2.1
--------	-------------------



Lab: Upload Cookbooks

- Upload your remaining cookbooks
- Verify that all cookbooks are uploaded

As a lab, upload the remaining cookbooks within the cookbooks directory. After you have done that verify that the cookbooks have been uploaded.

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: cd and Run knife cookbook list



```
$ cd ~/chef-repo/cookbooks/workstation  
$ knife cookbook list
```

```
apache      0.2.1
```

The one remaining cookbook is the workstation cookbook. Berkshelf is a cookbook management tool that examines the contents and dependencies of a single cookbook.

1. Change into the cookbooks directory.
2. Verify that the cookbook is not currently uploaded.

Lab: Install the Cookbook Dependencies



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'workstation' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using workstation (0.2.1) from source at .
```

Lab: Upload the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded workstation (0.2.1) to:  
'https://api.opscode.com:443/organizations/ORG'
```

Run "berks upload" to upload the cookbook and all its dependencies to the Chef Server.

Lab: Is the workstation Cookbook Uploaded?



```
$ knife cookbook list
```

```
apache      0.2.1
workstation 0.2.1
```



Lab: Upload Cookbooks

- ✓ Upload your remaining cookbooks
- ✓ Verify that all cookbooks are uploaded

Great. All of the cookbooks are on the Chef Server and you were able to verify they are there.



Hosted Chef

More easily manage multiple nodes

Objective:

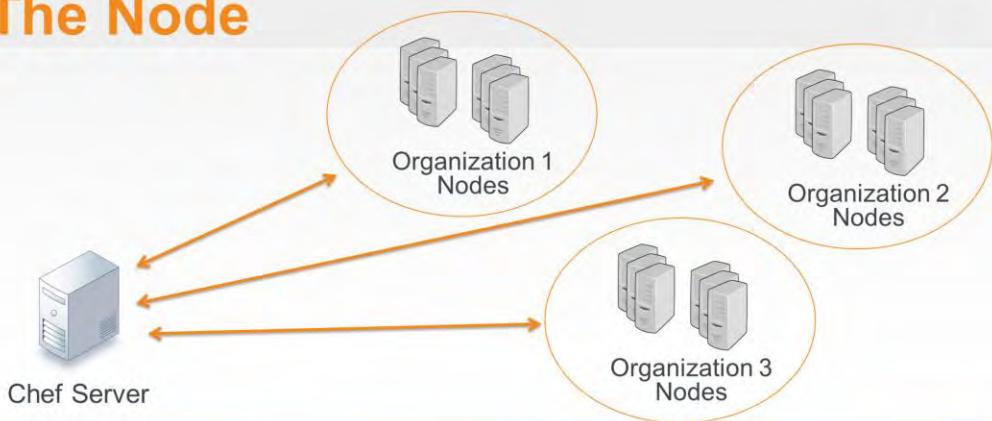
- ✓ Create a Hosted Chef Account
- ✓ Upload your cookbooks to the Hosted Chef Server
- ❑ Add a new node as a managed node

You have one remaining objective and that is to add an instance as a node within your organization. You will use a new node as the node to bootstrap.

CONCEPT



The Node



As you know by now, a node is a server that Chef is managing. A node could be a web server, an application server, a database server, a load balancer, and so on.

A node can only join one organization. To be a node means that it has Chef installed, has configuration files in place, and when you run the chef-client application with no parameters it will successfully contact the Chef Server and ask it for the run list that it should apply and the cookbooks required to execute that run list.

When a node is part of the organization you manage that information on the Chef Server as well. A Chef Server can manage multiple organizations. Managing that information in a Chef Server allows us to use for inventory, querying and searching.



GL: Bootstrap Your Node

In this lab you will use a new instance and bootstrap it as a managed node.

You'll need the FQDN of that instance to perform this lab.

GL: Change to the chef-repo



```
$ cd ~/chef-repo
```



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)
```

GL: Run 'knife node list'



```
$ knife node list
```





Bootstrapping a Node

Often, the node you are bootstrapping may not have Chef installed. It may also not have details of where the Chef 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.

<https://learn.chef.io/skills/beyond-essentials-1>

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 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 Server.

<https://learn.chef.io/skills/beyond-essentials-1>

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; this option is used internally by Opscode
  --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 Server; Running chef-client to apply a default run list.

GL: Bootstrap Your Node

```
$ knife bootstrap FQDN -x USER -P PWD --sudo -N node1
Creating new client for node1
Creating new node for node1
C   Fully Qualified Domain Name
e   .a
ec2-54-175-46-24.compute-1.amazonaws.com
ec2-54-175-46-24.compute-1.amazonaws.com resolving cookbooks for run list: []
ec2-54-175-46-24.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-175-46-24.compute-1.amazonaws.com [2016-09-16T16:51:21+00:00] WARN: Node node1 has an empty run list.
ec2-54-175-46-24.compute-1.amazonaws.com Converging 0 resources
ec2-54-175-46-24.compute-1.amazonaws.com
ec2-54-175-46-24.compute-1.amazonaws.com Running handlers:
```

To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-x' flag and the password '-P' flag. Include the '--sudo' flag because you are installing software and writing configuration to directories traditionally owned by the root user. Name the node with the '-N' flag. This is optional but makes it easier for us to communicate. When we ask you to look at the details of node 1 or login to node 1, it will be easier to remember than the fully-qualified domain name.

You should use the FQDN of yesterday's virtual workstation in this exercise.

Here's an example of this bootstrap command.

```
'knife bootstrap ec2-54-175-45-177.compute-1.amazonaws.com -x chef -P chef --sudo -N node1'
```

When executing the command, the output will tell us what it installed and ran.

GL: Run 'knife node list' Again



```
$ knife node list
```

```
node1
```

When bootstrapping is done, you can see that your organization knows about the new node by again running the command "knife node list". You now see that you have a new node, node1, uploaded to the Chef Server.

GL: View More Information About Your Node



```
$ knife node show node1
```

```
Node Name:    node1
Environment:  _default
FQDN:        ip-172-31-8-68.ec2.internal
IP:          54.175.46.24
Run List:
Roles:
Recipes:
Platform:    centos 6.8
Tags:
```

You can see more information about a particular node with the command 'knife node show node1'. This will display a summary of the node information that the Chef Server stores.

GL: Add a Recipe to a Run List

 \$ knife node run_list add node1 "recipe[apache]"

```
node1:  
  run_list: recipe[apache]
```

node1 does not have a list of recipes that it applies to the system by default. You can make Chef Server tell node1 to apply a specific run-list the next time node 1 runs 'chef-client'.

You can do that through the 'knife node run_list add' command. In this example, you are adding to node1's run-list the apache cookbook's default recipe.

GL: ssh to node1 and Converge Recipe



```
$ ssh chef@IPADDRESS  
$ sudo chef-client
```

```
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["apache"]  
Synchronizing Cookbooks:  
  - apache (0.2.1)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
...
```



Hosted Chef

More easily manage multiple nodes

Objective:

- ✓ Create a Hosted Chef Account
- ✓ Upload your cookbooks to the Hosted Chef Server
- ✓ Add a new node as a managed node

Discussion



Discussion

What is the benefit of storing cookbooks in a central repository?

What is the primary tool for communicating with the Chef Server?

How did you add a node to your organization?

Answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.



Q&A

What questions can you help you answer?

- Chef Server
- Managed Chef
- Berkshelf
- Bootstrapping Nodes

With all of the objectives complete you are finished with this section. What question can you answer for you?



Templates, Variables & Search

Creating Dynamic Content

©2016 Chef Software Inc.



Now we're going to switch focus to talk about templates, variables and we'll also introduce search.

Note: There are bit.ly URLs in this and some of the following modules. If you open your participant guide with Acrobat Reader (or via Preview on a Mac), you could copy those bit.ly URLs from the speaker notes section below the slides and then paste the URLs into browser. You may also be able to simply click the URLs from the speaker notes section below each slide.

Objectives

After completing this module, you should be able to:

- Set a run list while bootstrapping a node.
- Identify EC2 specific node attributes.
- Execute a search using knife and within a recipe.

After completing this module you'll be able to run a search across all nodes in your infrastructure. You'll also be able to set a run list while bootstrapping a node



Manage Multiple Nodes

Our site has just gotten super busy so we now need to manage multiple nodes.

So, time has moved on, our site has got super busy so we need use multiple web server nodes to handle the traffic

Exercise



GL: Bootstrap a New Node

In this exercise you will:

- Bootstrap another node named node2.
- Also set the run list at boot time.

GL: Lets Bootstrap a New Node



```
$ knife bootstrap FQDN -x USER -P PWD --sudo -N node2 -r 'recipe[apache]'
```

```
...
54.84.233.7      +      <h2>ipaddress: 172.31.29.219</h2>
54.84.233.7      +      <h2>hostname: ip-172-31-29-219</h2>
54.84.233.7      +</body>
54.84.233.7      +</html>
54.84.233.7    * service[httpd] action enable
54.84.233.7      - enable service service[httpd]
54.84.233.7    * service[httpd] action start
54.84.233.7      - start service service[httpd]
54.84.233.7
54.84.233.7 Running handlers:
54.84.233.7 Running handlers complete
54.84.233.7 Chef Client finished, 4/4 resources updated in 24.447046971 seconds
```

So we're going to bootstrap the new node and name it node2.

Potential gotcha to look out for:-

If someone copies this bootstrap command from a PDF (or a slide) and pastes to command line, then it might not pick up the correct '-' character for the run list flag, resulting in an empty run list. See the difference in the '-' between this (first is incorrect!):-

```
knife bootstrap 52.90.0.212 -x chef -P chef --sudo -N node3 -r 'recipe[haproxy]'
```

And this

```
knife bootstrap 52.90.0.212 -x chef -P chef --sudo -N node3 -r 'recipe[haproxy]'
```

GL: Lets Bootstrap a New Node



```
$ knife bootstrap FQDN -x USER -P PWD --sudo -N node2 -r 'recipe[apache]'
```

```
...
54.84.233.7      +      <h2>ipaddress: 172.31.29.219</h2>
54.84.233.7      +      <h2>hostname: ip-172-31-29-219</h2>
54.84.233.7      +</body>
54.84.233.7      +</html>
54.84.233.7      * service[httpd] action enable
54.84.233.7          - enable service service[httpd]
54.84.233.7      * service[httpd] action start
54.84.233.7          - start service service[httpd]
54.84.233.7
54.84.233.7 Running handlers:
54.84.233.7 Running handlers complete
54.84.233.7 Chef Client finished, 4/4 resources updated in 24.447046971 seconds
```

The one difference between bootstrapping this node, and what we did last time is now we're setting the run list at boot time by passing the '-r' flag

DISCUSSION



Test the Webservers

To test these web servers we need to find their FQDN or public IP Address

Obviously we know the name & IP address of our node in this instance as we've just bootstrapped it. However this node could have been launched automatically, and in such circumstances if the node name was not specified, then the FQDN will be used, and this is often autogenerated (like in AWS for example)

So you lets say we need to find the name & IP address of the nodes.

GL: List All of Our Nodes



```
$ knife node list
```

```
node1  
node2
```

We could use the ‘knife node list’ command to find all nodes on your system

GL: View More Information About Nodes



```
$ knife node show node1
```

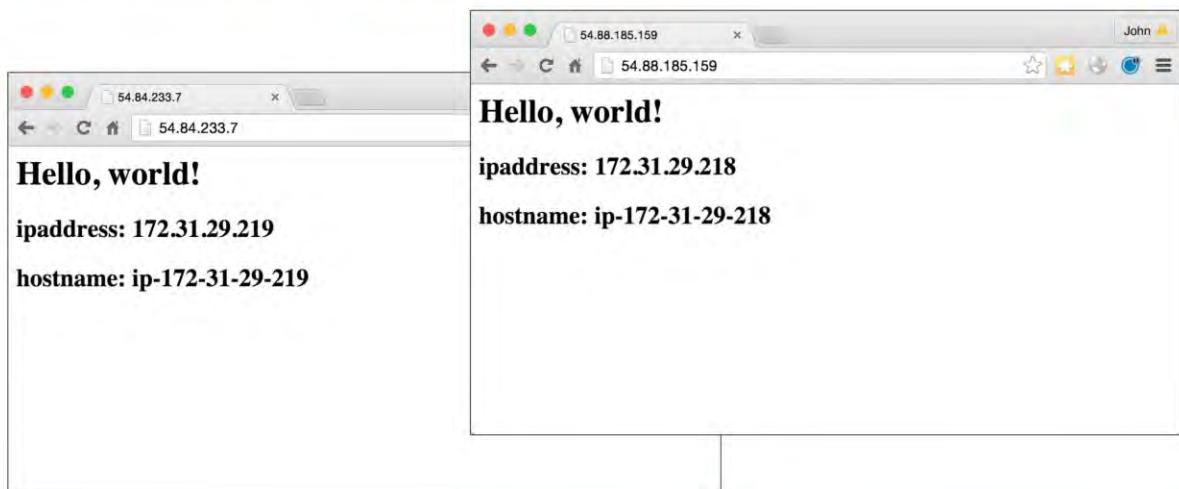
```
Node Name: node1
Environment: _default
FQDN: ip-172-31-60-9.ec2.internal
IP: 54.167.232.148
Run List: recipe[apache]
Roles:
Recipes: apache, apache::default,
apache::server
Platform: centos 6.8
Tags: :
```

```
$ knife node show node2
```

```
Node Name: node2
Environment: _default
FQDN: ip-172-31-1-215.ec2.internal
IP: 54.205.59.139
Run List: recipe[apache]
Roles:
Recipes: apache, apache::default,
apache::server
Platform: centos 6.8
Tags:
```

Once we've found the nodes, we could use 'knife node show' to find some details on them.

GL: Testing Our Websites



Point a web browser to the URL or public IP address of your webservers.



How Do We See Detail Across All Nodes?

`knife node show` only works with individual nodes so how do I see this info for multiple nodes?

So we need a way to search the Chef Server and return details of **ALL** our webserver nodes at once.



How Do We See Detail Across All Nodes?

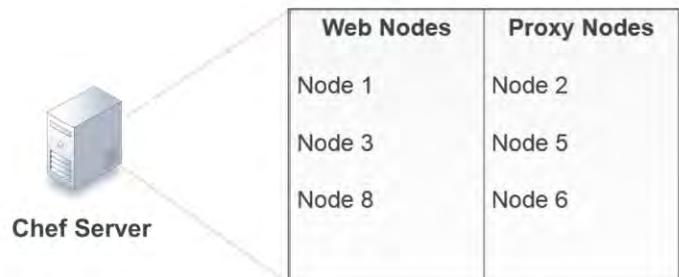
`knife node show` only works with individual nodes so how do I see this info for multiple nodes?

Search allows you to retrieve information across multiple nodes.

So we need a way to search the Chef Server and return details of **ALL** our webserver nodes at once

The Chef Server and Search

Chef Server maintains a searchable index of all nodes within our infrastructure.



The Chef Server maintains a representation of all the nodes within an infrastructure and provides a way for us to discover these systems through Search.

Search is a service discovery tool that allows us to query the Chef Server across a few indexes. One such index is on our nodes.

What is search?

Use search to query data indexed on the chef server

```
$ knife search INDEX SEARCH_QUERY
```

The search runs on the server and is invoked from within a recipe or using knife



INDEX can be 'client', 'environment', 'node', 'role', (or the name of a data bag)

SEARCH_QUERY is of the format "attribute:value"

Querying *** : *** returns everything

The slide shows the syntax for invoking a search from the command line. The knife command can be used to instigate a search on the Chef Server, and results are returned to the workstation. The format is

"knife search index (what type of object you looking for) search_query (which of these you want)"

The index could be 'node', 'client', 'environment', 'role'. Right now we're looking for nodes.

The search_criteria is which nodes do we want. We could just use "*** : ***" to find all nodes, or you could narrow your search.

<http://wiki.apache.org/solr/SolrQuerySyntax>

GL: View Information for All Nodes



```
$ knife search node "*:*"
```

```
2 items found

Node Name: node1
Environment: _default
FQDN: ip-172-31-29-218.ec2.internal
IP: 54.88.185.159
Run List: recipe[apache]
Roles:
Recipes: apache::default, apache::server
Platform: centos 6.7
Tags:

Node Name: node2
Environment: _default
FQDN: ip-172-31-29-219.ec2.internal
IP: 54.84.233.7
Run List: recipe[apache]
Roles:
Recipes: apache::default, apache::server
Platform: centos 6.7
```

So run the command `knife search node "*:*`" – you should see all nodes returned.

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 web servers) 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.

GL: Narrow the Search

```
 $ knife search node "*:*" -a ipaddress
```

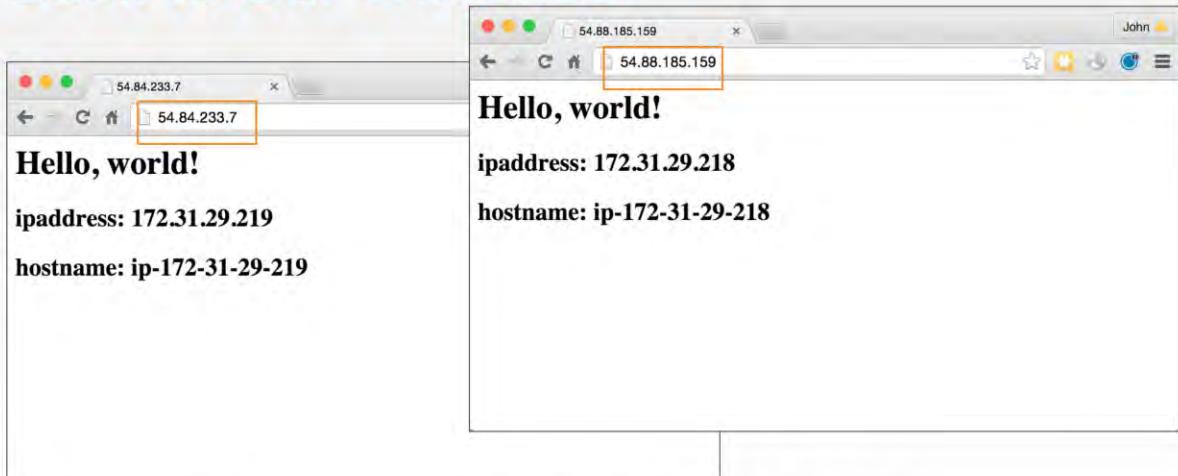
```
2 items found

node1:
  ipaddress : 172.31.13.2

node2:
  ipaddress : 172.31.6.173
```

The '-a' flag allows you to specify a particular attribute from those nodes. So if you append '-a ipaddress' you'll get the IP Address returned

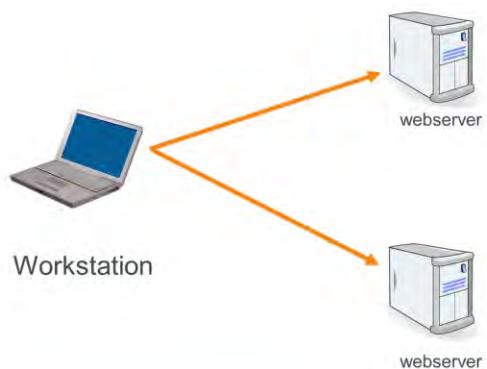
Back to Our Websites



We have to browse to each site individually, ugh.

Two Webservers, One Browser

So we have scaled out our webservers, but we still need to hit each webserver individually.

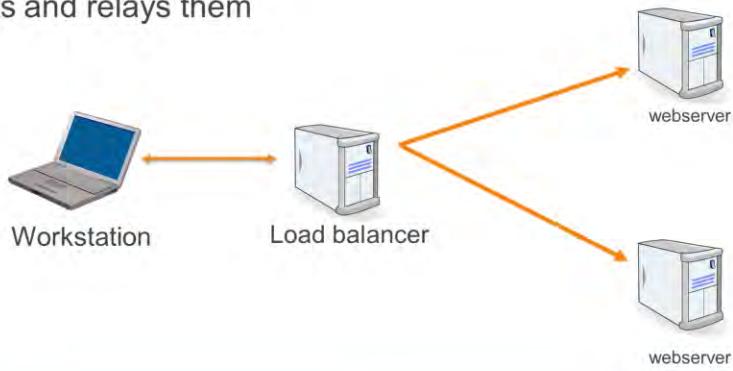


Having to browse to both our webservers individually is less than ideal

Load Balancer

Adding a load balancer will allow us to better grow our infrastructure.

The LB receives requests and relays them to the web servers.



So 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 want to use the load balancer to balance the traffic load between one or more web servers.

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 node running apache and to future nodes.



GL: Scaling up

Our site has just got super busy with multiple web servers – so we now need a load balancer.

Objective:

- Create a load balancer cookbook
- Upload cookbook to Chef Server
- Bootstrap a new node that runs the load balancer cookbook

So we want to instantiate a load balancer. To do this there are three steps we need to complete:-

- Create a load balancer cookbook
- Upload cookbook to Chef Server
- Bootstrap a new node that runs the load balancer cookbook

GL: Generate HAProxy Cookbook

```
 $ cd ~/chef-repo  
$ chef generate cookbook cookbooks/haproxy  
  
Compiling Cookbooks...  
Recipe: code_generator::cookbook  
  * directory[C:/Users/YOU/chef-repo/cookbooks/haproxy] action create  
    - create new directory C:/Users/YOU/chef-repo/cookbooks/haproxy  
  * template[C:/Users/YOU/chef-repo/cookbooks/haproxy/metadata.rb] action create_if_missing  
    - create new file C:/Users/YOU/chef-repo/cookbooks/haproxy/metadata.rb  
    - update content in file C:/Users/YOU/chef-repo/cookbooks/haproxy/metadata.rb from none  
      to 899276  
      (diff output suppressed by config)  
  * template[C:/Users/YOU/chef-repo/cookbooks/haproxy/README.md] action create_if_missing
```

Change to your chef-repo directory and then 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.

GL: Edit haproxy cookbook's default recipe

chef-repo/cookbooks/haproxy/recipes/default.rb

```
#  
# Cookbook Name:: haproxy  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
package 'haproxy'  
  
template '/etc/haproxy/haproxy.cfg' do  
  source 'haproxy.cfg.erb'  
end  
  
service 'haproxy' do  
  action [:start, :enable]  
end
```

So please type in the contents of the default recipe.

We're going to use HAProxy as our load balancer. You'll notice that this is VERY similar to the Apache default recipe. It uses the package/template/service trio of resources to install the package, create the configuration file and start the service. These three resources are the most important in config management.

Instructor Note:

<http://bit.ly/1pQmPar>

GL: Edit haproxy cookbook's default recipe

chef-repo/cookbooks/haproxy/recipes/default.rb

```
#  
# Cookbook Name:: haproxy  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
package 'haproxy'  
  
template '/etc/haproxy/haproxy.cfg' do  
  source 'haproxy.cfg.erb'  
end  
  
service 'haproxy' do  
  action [:start, :enable]  
end
```

We're using a 'template' resource to create a configuration file for HAProxy. This configuration will contain details of each web server it needs to route traffic to.



Configure HAProxy

We need to configure HAProxy to route traffic to our web server.

Have a look at how HAProxy is configured

<http://bit.ly/1Xoai9R>

You can see the format of the HAProxy configuration file (`/etc/haproxy/haproxy.cfg`) at this link shown. Most of this is static text, except for the very last line.

<http://bit.ly/1Xoai9R>

GL: Generate the Template



```
$ chef generate template cookbooks/haproxy haproxy.cfg
```

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

Use '**chef generate template**' to create a template in the haproxy cookbook - the file we want to create is named haproxy.cfg

GL: Configuring haproxy.cfg.erb

cookbooks/haproxy/templates/haproxy.cfg.erb

```
...
frontend main *:80
  acl url_static      path_beg     -i /static /images /javascript /stylesheets
  acl url_static      path_end     -i .jpg .gif .png .css .js

  use_backend static      if url_static
  default_backend          app

backend static
  balance roundrobin
  server   static 127.0.0.1:4331 check

backend app
  balance roundrobin
  server app <><IP ADDRESS>>:80 weight 1 maxconn 100 check
```

Copy the bit.ly URL from below the slide in your participant guide:

<http://bit.ly/1Xoai9R>

Copy/paste from the gist URL shown at <http://bit.ly/1Xoai9R>

and then we'll modify the green highlighted line in a moment. You'll see there is a placeholder for the IP address – we'll populate that shortly.

Top Tip: Click the 'Raw' button, the ctrl-a & ctrl-c to copy all the text. If you try select the text from within the gist window you could pick up some unwanted characters.



Deriving the Public IP Address in AWS

Let's stop for a moment and discuss how we can add a webserver's public IP Addresses to haproxy.cfg using the `cloud` attribute.

HAProxy Configuration Should Look Like This

```
...  
backend app  
    balance roundrobin  
    server app0 <><node1 IP ADDRESS>>:80 weight 1 maxconn 100 check  
    server app1 <><node2 IP ADDRESS>>:80 weight 1 maxconn 100 check
```

We need to add the webserver's IP Addresses to haproxy.cfg

Discussion: We need to populate haproxy.cfg with the IP Addresses of the webservers – i.e. we need to configure the load balancing pool. This requires having one line per webserver.

HAProxy Configuration Should Look Like This

```
...  
backend app  
    balance roundrobin  
    server app0 <>node1 IP ADDRESS>>:80 weight 1 maxconn 100 check  
    server app1 <>node2 IP ADDRESS>>:80 weight 1 maxconn 100 check
```

We need to add the webserver's IP Addresses to haproxy.cfg

Doing it manually seems wrong

Search!

We need to populate haproxy.cfg with the IP Addresses of the webservers – i.e. we need to configure the load balancing pool. This requires having one line per webserver.

But this needs to be updated dynamically, as the list of webservers could be changing dynamically

So lets use the search facility

HAProxy Configuration Should Look Like This

```
...  
backend app  
    balance roundrobin  
    server app0 <><node1 IP ADDRESS>>:80 weight 1 maxconn 100 check  
    server app1 <><node2 IP ADDRESS>>:80 weight 1 maxconn 100 check
```

Values from

```
knife search node "recipes:apache\\:\\default" -a ipaddress
```

We'll want to pull the IP addresses of our webservers directly from the Chef Server.
i.e. we want to find the IP address of all nodes whose run list contains the recipe
'apache::default'

If we run this search from the command line we need to escape the colon (:) as it's a
special character in the search

Houston, We Have a Problem!



```
$ knife search node "recipes:apache\\:\\default" -a ipaddress
```

```
2 items found
```

```
node1:
```

```
  ipaddress: 172.31.29.218
```

```
node2:
```

```
  ipaddress: 172.31.29.219
```

These IP Addresses are not accessible from the outside network

However the problem we're seeing is the 'ipaddress' node attribute returns the node's non-routable IP Address (IPs in the range 172.16.0.0 - 172.31.255.255 cannot be accessed from outside their local own network).

These VMs are running in EC2, so we need their public facing IP address.



Amazon EC2 Instances

We can't use the `ipaddress` attribute within our recipes – they're on the private (internal) network and we need external access.

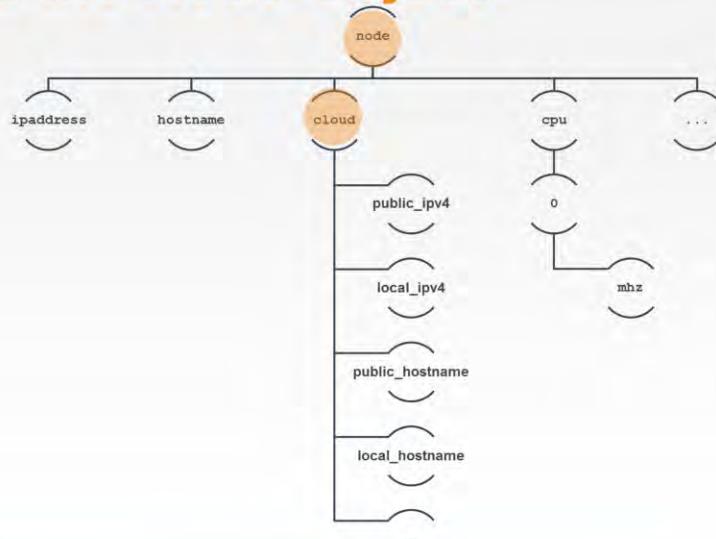
In a previous section we looked at the node object.

Nodes in EC2 have some specific networking attributes in their node object – some private & some public.

The reason you may need to ask the node for a different set of attributes is that we are using Amazon as a cloud provider for our instances. These instances are displaying the internal IP address when we ask for the `ipaddress` attribute.

Ohai collects attributes from the current cloud provider and makes them available in an attribute named 'cloud'. We can look at the `cloud` attribute on our first node and see that it returns for us information about the node.

EC2 and the Node Object



You will remember a diagram like this from before when we talked about the node object. If the node is running in EC2, then Ohai populates a special attribute called "cloud". This attribute is actually a nested hash that contains the information we need.

Instructor Note: These ec2 specific attributes only appear if the file '/etc/chef/ohai/hints/ec2.json' exists and contains {}. This file can be created manually, but is placed automatically if you use the `knife ec2` plugin (i.e. `knife ec2 server create ...`) or using Chef Provisioning.

Individual Node's EC2 Information



```
$ knife node show node1 -a cloud
```

```
node1:
  cloud:
    local_hostname: ip-172-31-29-218.ec2.internal
    local_ipv4: 172.31.29.218
    private_ips: 172.31.29.218
    provider: ec2
    public_hostname: ec2-54-88-185-159.compute-1.amazonaws.com
    public_ips: 54.88.185.159
    public_ipv4: 54.88.185.159
```

Individual Node's Public IP Address



```
$ knife node show node1 -a cloud.public_ipv4
```

```
node1:  
  cloud.public_ipv4: 54.88.185.159
```

If you use 'knife node show' to display the 'cloud' attribute for node1, you will see the local, private, and public connection information.

We need the public_ipv4 address of node1 in our HAProxy config file. So you can get that information using the syntax '`-a cloud.public_ipv4`'

View EC2 Information for All Nodes



```
$ knife search node "*:*" -a cloud
```

```
2 items found

node1:
  cloud:
    local_hostname: ip-172-31-29-218.ec2.internal
    local_ipv4: 172.31.29.218
    private_ips: 172.31.29.218
    provider: ec2
    public_hostname: ec2-54-88-185-159.compute-1.amazonaws.com
    public_ips: 54.88.185.159
    public_ipv4: 54.88.185.159

node2:
  cloud:
    local_hostname: ip-172-31-29-219.ec2.internal
    local_ipv4: 172.31.29.219
  ...
```

Again, you can see more information about a particular node with the command 'knife node show node1', however if you use 'knife search' you can see information about a collection of nodes.

View Public IP for All Nodes

 \$ knife search node "*:*" -a cloud.public_ipv4

```
2 items found

node1:
  cloud.public_ipv4: 54.88.185.159

node2:
  cloud.public_ipv4: 54.84.233.7
```

The information we want is the 'cloud.public_ipv4' attribute.

HAProxy Configuration Should Look Like This

```
...  
backend app  
    balance roundrobin  
    server app0 <>node1 IP ADDRESS>>:80 weight 1 maxconn 100 check  
    server app1 <>node2 IP ADDRESS>>:80 weight 1 maxconn 100 check
```

Values from

```
knife search node "recipes:apache\:\:default" -a cloud.public_ipv4
```

So the search we invoke should return the 'cloud.public_ipv4' attribute for all nodes that run the recipe [apache>::default].

GL: Edit haproxy Cookbook's Default Recipe

chef-repo/cookbooks/haproxy/recipes/default.rb

```
...
package 'haproxy'

allwebservers = search('node', 'recipes:apache\:\:default')

template '/etc/haproxy/haproxy.cfg' do
  source 'haproxy.cfg.erb'
  variables(
    :webservers => allwebservers
  )
  notifies :restart, 'service[haproxy]'
end

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

Copy the bit.ly URL from below the slide in your participant guide:

<http://bit.ly/1pQo0qd>

So let's continue our group lab exercise. Up until now we've invoked a search from the command line using knife. However the real power of Chef comes when you embed a search within a recipe.

Copy/paste from the gist URL shown <http://bit.ly/1pQo0qd> and then we'll modify the file.

Tip: Click the 'Raw' button, then ctrl-a & ctrl-c to copy all the text from <http://bit.ly/1pQo0qd>. If you try select the text from within the gist window you could pick up some unwanted characters.

GL: Edit haproxy Cookbook's Default Recipe

chef-repo/cookbooks/haproxy/recipes/default.rb

```
...
package 'haproxy'

allwebservers = search('node', 'recipes:apache\:\:default')

template '/etc/haproxy/haproxy.cfg' do
  source 'haproxy.cfg.erb'
  variables(
    :webservers => allwebservers
  )
  notifies :restart, 'service[haproxy]'
end

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

Invoke a search in the recipe
and save the results in the
variable 'allwebservers'

So the first thing we want to do is update our recipe to include a search. This search returns a list of all web server nodes into a variable called “**allwebservers**” - for all you Ruby folks, this returns an ruby array. Notice the search syntax in a recipe is different than at command line.

GL: Edit haproxy Cookbook's Default Recipe

chef-repo/cookbooks/haproxy/recipes/default.rb

```
...
package 'haproxy'

allwebservers = search('node', 'recipes:apache\:\:default')

template '/etc/haproxy/haproxy.cfg' do
  source 'haproxy.cfg.erb'
  variables(
    :webservers => allwebservers
  )
  notifies :restart, 'service[haproxy]'
end

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

Pass the variable 'allwebservers'
into the template

We then pass this array directly into the template.

GL: Configuring haproxy.cfg.erb

cookbooks/haproxy/templates/haproxy.cfg.erb

```
...  
use_backend static          if url_static  
default_backend             app  
  
backend static  
    balance     roundrobin  
    server      static 127.0.0.1:4331 check  
  
backend app  
    balance     roundrobin  
    server app<<IP ADDRESS>>:80 weight 1 maxconn 100 check  
    <% @webservers.each_with_index do |web, n| -%>  
        server <%= "app#{n}" %> <%= web['cloud']['public_ipv4'] %>:80 weight 1 maxconn 100 check  
    <% end -%>
```

Remove line in template with hardcoded IP placeholder

Copy the bit.ly URL from below the slide in your participant guide:

<http://bit.ly/1pQnwQZ>

In the template file we want to remove the last line and replace it with the three lines shown on the slide.

File URL: <http://bit.ly/1pQnwQZ>

Instructor note: You should understand what's going on here, and talk the class through it in detail.

GL: Configuring haproxy.cfg.erb

cookbooks/haproxy/templates/haproxy.cfg.erb

```
...
use_backend static           if url_static
default_backend              app

backend static
  balance     roundrobin
  server     static 127.0.0.1:4331 check

backend app
  balance     roundrobin
<% @webservers.each_with_index do |web, n| -%>
  server <%= "app#{n}" %> <%= web['cloud']['public_ipv4'] %>:80 weight 1 maxconn 100 check
<% end -%>
```

Iterate over the 'webservers' and add a line for each

So here we're taking the array of web servers that's passed in from the recipe, and looping over each one in a '.each_with_index' ruby statement (i.e. essentially a 'for' loop).

Looking at the first line, for each node we create two local variables in each loop of the .each statement – 'web' and 'n'. 'n' is simply a sequential number that starts at '0' and increments by 1 each time it goes through the loop, and 'web' contains the actual node data.

Then on the second line a new line gets printed in the 'haproxy.cfg' file during each iteration of the .each loop. The '<%= "app#{n}" %>' prints the sequence number ('0', '1', '2', etc) and '<%= web['cloud']['public_ipv4'] %>' prints the public IP address of that node.

We'll see the resulting file a little later

Instructor Note

<http://bit.ly/1pQnwQZ>

You should understand what's going on here, and talk the class through it in detail



Lab: Scaling up

Our site has just got super busy with multiple web servers – so we now need a load balancer.

- ✓ Create a load balancer cookbook
- Upload cookbook to Chef Server
- Bootstrap a new node that runs the load balancer cookbook

So now we have created the HAProxy cookbook.

Next we need to upload it to the Chef Server. As a lab exercise, upload the cookbook to the Chef Server

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: Upload the Cookbook



```
$ cd ~/chef-repo/cookbooks/haproxy
```

Lets review that lab.

You change into the directory for the 'haproxy' cookbook.

Lab: Upload the Cookbook



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'haproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using haproxy (0.1.0) from source at .
```

We use the Berkshelf to upload our cookbooks. Run the command "berks install".

Lab: Upload the Cookbook



```
$ berks upload
```

```
Uploaded haproxy (0.1.0) to: 'https://api.opscode.com:443/organizations/ORGNAME'
```

Lab: Verify the Cookbook Upload



```
$ knife cookbook list
```

apache	0.2.1
haproxy	0.1.0
workstation	0.2.1



Lab: Scaling up

Our site has just got super busy with multiple web servers – so we now need a load balancer.

Objective:

- Create a load balancer cookbook
- Upload cookbook to Chef Server
- Bootstrap a new node that runs the load balancer cookbook

Lab: Bootstrap a Load Balancer Node



```
$ knife bootstrap FQDN -x USER -P PWD --sudo -N node3 -r 'recipe[haproxy]'
```

```
...
ec2-54-162-31-253.compute-1.amazonaws.com      +    server app0 54.205.59.139:80 weight 1
maxconn 100 check
ec2-54-162-31-253.compute-1.amazonaws.com      +    server app1 54.167.232.148:80 weight 1
maxconn 100 check
ec2-54-162-31-253.compute-1.amazonaws.com      * service[haproxy] action start
ec2-54-162-31-253.compute-1.amazonaws.com      - start service service[haproxy]
ec2-54-162-31-253.compute-1.amazonaws.com      * service[haproxy] action enable
ec2-54-162-31-253.compute-1.amazonaws.com      - enable service service[haproxy]
ec2-54-162-31-253.compute-1.amazonaws.com      * service[haproxy] action restart
ec2-54-162-31-253.compute-1.amazonaws.com      - restart service service[haproxy]
ec2-54-162-31-253.compute-1.amazonaws.com
ec2-54-162-31-253.compute-1.amazonaws.com Running handlers:
ec2-54-162-31-253.compute-1.amazonaws.com Running handlers complete
ec2-54-162-31-253.compute-1.amazonaws.com Chef Client finished, 5/5 resources updated in 57
```

So we're going to bootstrap the new load balancer node and name it node3. In this case we're also setting the haproxy default recipe at the same time.

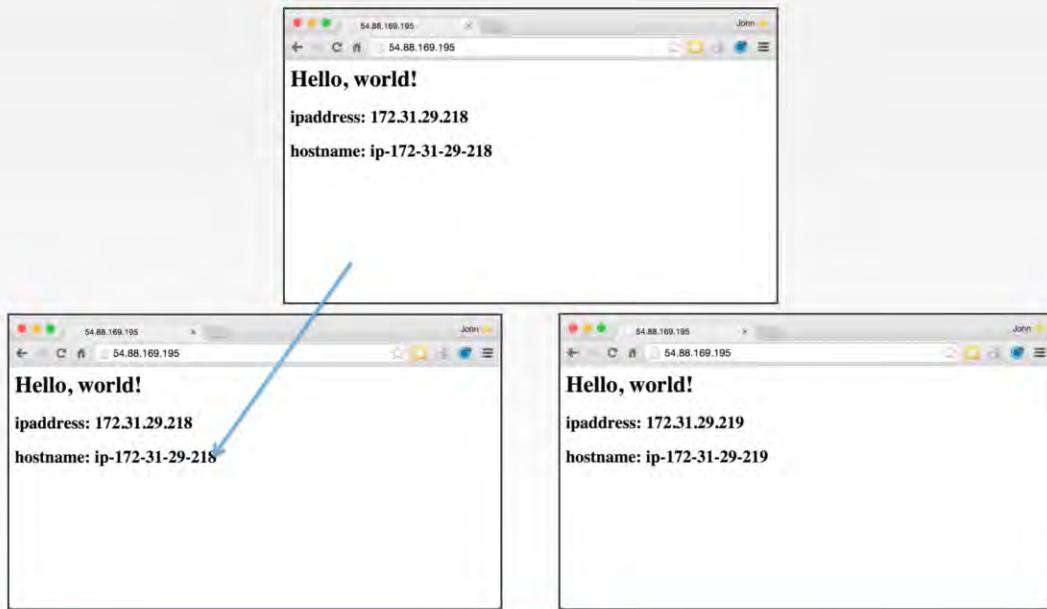
You'll need the FQDN of one of you remaining unused nodes.

Lab: Validate the New Node



```
$ knife node show node3
```

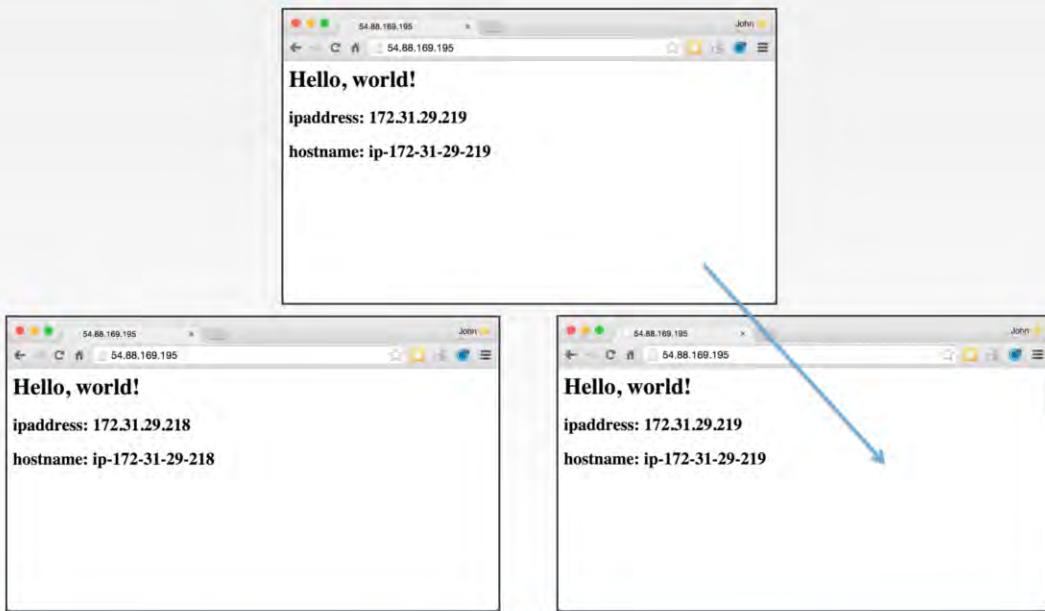
```
Node Name:  node3
Environment: _default
FQDN:      ip-172-31-59-247.ec2.internal
IP:        54.174.229.169
Run List:   recipe[haproxy]
Roles:
Recipes:    haproxy, haproxy::default
Platform:   centos 6.8
Tags:
```



Point a web browser to the FQDN or public IP address of your load balancer. It should display the web pages of the web server nodes that the load balancer is configured to serve.

Refresh your browser a few times to see the round-robin effect.

Nothing should change externally. You may see some differences in the logs as the proxy configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.



Refresh your browser a few times to see the round-robin effect.

DISCUSSION



Discussion

How might Search work in the context of a webserver & database?

Where else might you use dynamic content in files?

Answer these questions.

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

Discussion



Q&A

What questions can we help you answer?

- Search
- Passing variables into templates

What questions can we help you answer?

In general or about specifically about Chef Super Market, wrapper cookbooks, node attributes, the 'knife ssh' command.



Cookbook Attributes, Attribute Files and Dependencies

Setting attributes within a cookbook

Your goal in this module is to bootstrap another node, this time a web server, and add it to the proxy members.

Objectives

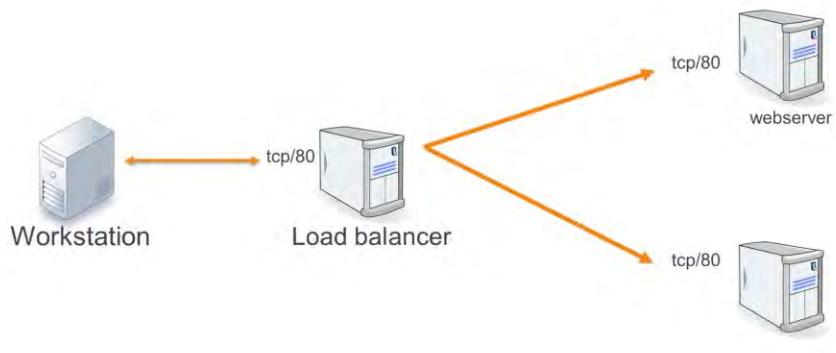
After completing this module, you should be able to

- Explain where cookbook attributes reside
- Configure dependencies between cookbooks
- Use `knife ssh` command

In this module you will learn how to bootstrap, update the run list, and run chef-client on a node. You will also learn how to update a default attribute within a recipe, version and upload a cookbook.

Our Topology

We now have a load balancer & two web servers, all listening on port 80



With a single web server running with our organization, it's now time to talk about the next goal to tackle. We need to setup a proxy server.

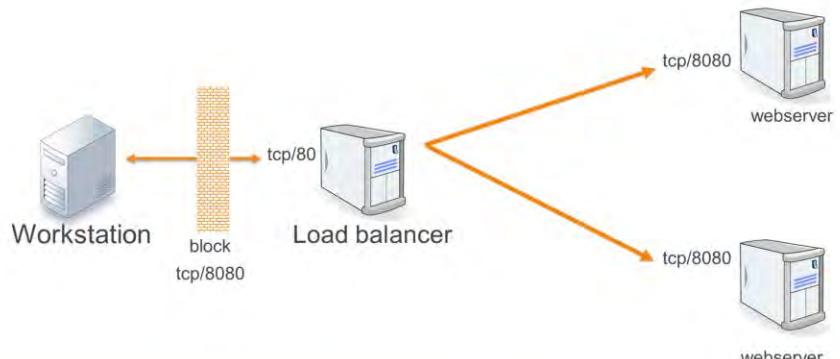
A proxy server is able to receive requests and relay them to other systems. In our case, we specifically want to use the proxy server to balance the entire traffic load between one or more systems.

This means we will need to establish a new node within our organization, install the necessary software to make the node a proxy server, and configure it so that it will relay requests to our existing node running apache and to future nodes.

Our Topology

We now have a load balancer & two web servers, all listening on port 80

IT have mandated that direct access to the webservers should be blocked





Reconfigure Apache

Apache is configured to listen on port 80 in the file
`/etc/httpd/conf/httpd.conf`

We will create a template for this file in the apache cookbook and change the value to 8080 via an attribute.



Attribute Files

The Node Object contains many automatic attributes generate 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.



Best Practices

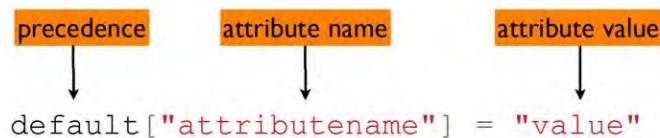
- Well-written cookbooks change behavior based on attributes.
- Ideally, you don't have to modify the contents of a cookbook to use it for your specific use case.
- Look at the attributes directory for things you can override through roles to affect behavior of the cookbook.
- Of course, well written cookbooks have sane defaults, and a README to describe all this.

Setting Attributes in Attribute Files

Cookbook attributes are set in the attributes file

```
./cookbooks/<cookbook>/attributes/default.rb
```

Format is:



We'll look at precedence later.



EXERCISE

GL: Reconfigure Apache

So we want HAProxy to serve content of port tcp/80, and Apache to serve of tcp/8080.

Objective:

- Reconfigure Apache to serve content of port tcp/8080
- Reconfigure HAProxy to contact Apache on port tcp/8080

GL: Bump the Cookbook Version Number

```
~/chef-repo/cookbooks/apache/metadata.rb
```

```
name          'apache'
maintainer    'The Authors'
maintainer_email 'you@example.com'
license        'all_rights'
description    'Installs/Configures apache'
long_description 'Installs/Configures apache'
version        '0.3.0'
```

GL: Generate the attributes File

```
 $ cd ~/chef-repo  
$ chef generate attribute cookbooks/apache default
```

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

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

GL: Set the Port Value as an Attribute

cookbooks/apache/attributes/default.rb

```
default['apache']['port'] = 8080
```

Its good practice to include the name of the cookbook in the attribute name – helps trace where the value is set, although this is not enforced.

GL: Update the apache::server Recipe

cookbooks/apache/recipes/server.rb

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

template '/etc/httpd/conf/httpd.conf' do
  action :create
  source 'httpd.conf.erb'
  notifies :restart, 'service[httpd]'
end

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

GL: Generate the Template file



```
$ chef generate template cookbooks/apache httpd.conf
```

```
output suppressed by config)
MacBook-Pro-3:chef-repo johnfitzpatrick$ chef generate template
cookbooks/apache httpd.conf.erb
Compiling Cookbooks...
Recipe: code_generator::template
  * directory[cookbooks/apache/templates/default] action create (up to date)
  * template[cookbooks/apache/templates/default/httpd.conf.erb] action create
    - create new file cookbooks/apache/templates/default/httpd.conf.erb
    - update content in file cookbooks/apache/templates/default/httpd.conf.erb
from none to e3b0c4
  (diff output suppressed by config)
```

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

GL: Add the 'port' Variable to the Template

cookbooks/apache/templates/default/httpd.conf.erb

```
...  
MaxSpareThreads    75  
ThreadsPerChild    25  
MaxRequestsPerChild 0  
</IfModule>
```

```
Listen 80  
Listen <%= node['apache']['port'] %>
```

```
LoadModule auth_basic_module modules/mod_auth_basic.so  
LoadModule auth_digest_module modules/mod_auth_digest.so  
...
```

If you are feeling hardcore, type it. Or copy the bit.ly URL from below the slide in your participant guide:

<http://bit.ly/1Hdx7E1>

Please see instructions below in your participant guide.

1. Go to this URL... <http://bit.ly/1Hdx7E1> ...click the Raw button, and copy the contents of the file.
2. Paste the copied contents into cookbooks/apache/templates/default/httpd.conf.erb and modify the line (line 27) as shown in this slide.

LAB



Lab: Upload the Cookbook

Objective:

- Upload the apache cookbook to the Chef Server

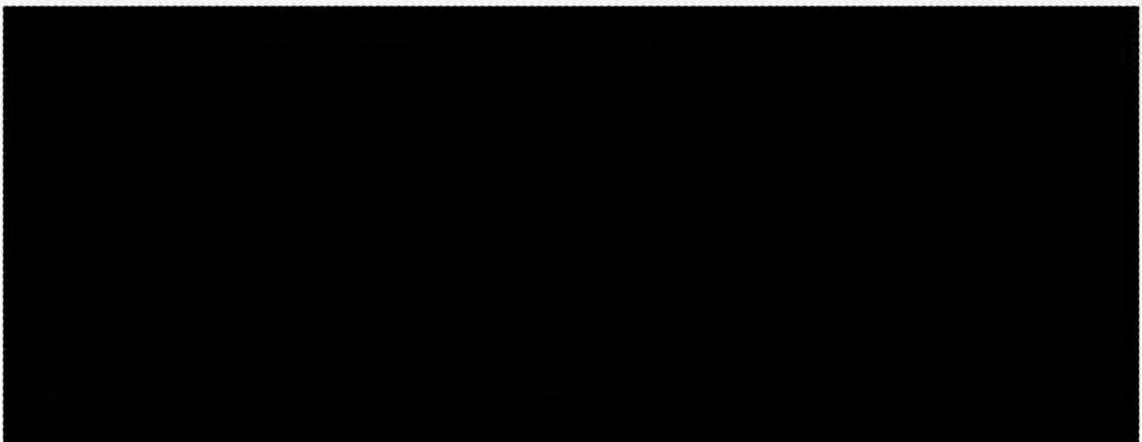
As a lab exercise, upload the cookbook to the Chef Server

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: Upload the Cookbook



```
$ cd cookbooks/apache
```



Lab: Upload the Cookbook



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using apache (0.3.0) from source at .
```

Lab: Upload the Cookbook



```
$ berks upload
```

```
Uploaded apache (0.3.0) to:  
'https://api.opscode.com:443/organizations/ORGNAME'
```

After installing all the necessary dependent cookbooks, we used 'berks upload' to send the cookbook and all its dependencies to the Chef Server. This is again an easier method to manage dependencies instead of manually identifying the dependencies and then uploading each single cookbook at a time.



EXERCISE

GL: Reconfigure Apache

So we want HAProxy to serve content of port tcp/80, and Apache to serve of tcp/8080.

Objective:

- ✓ Reconfigure Apache to serve content of port tcp/8080
- Reconfigure HAProxy to contact Apache on port tcp/8080

GL: Bump the haproxy Cookbook Version

```
cookbooks/haproxy/metadata.rb
```

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

GL: Configuring haproxy.cfg.erb

`cookbooks/haproxy/templates/default/haproxy.cfg.erb`

```
...
backend static
  balance roundrobin
  server static 127.0.0.1:4331 check

backend app
  balance roundrobin
  <% @webservers.each_with_index do |web, n| -%>
    server <%= "app#{n}" %> <%= web['cloud']['public_ipv4'] %>:80 weight 1 maxconn 100 check
    server <%= "app#{n}" %> <%= web['cloud']['public_ipv4'] %>:<%= node['apache']['port'] %>
  weight 1 maxconn 100 check
<% end -%>
```

Add the port attribute as configured
in the apache cookbook

You can copy the bit.ly URL from
below the slide in your participant
guide:

<http://bit.ly/1Xoai9R>

You can copy/paste from here <http://bit.ly/1Xoai9R>

Note. The text in the green highlight is wrapped on the slide but it should be one line.



We've Introduced a Dependency

We now have the situation where HAProxy cookbook relies on an attribute that is configured in version 0.3.0 of the Apache cookbook

We need to manage this dependency – we do this in the haproxy cookbook `metadata.rb` file

GL: Bump the Cookbook Version number

```
cookbooks/haproxy/metadata.rb
```

```
name          'haproxy'  
maintainer    'The Authors'  
maintainer_email 'you@example.com'  
license        'all_rights'  
description    'Installs/Configures haproxy'  
long_description 'Installs/Configures haproxy'  
version        '0.2.0'  
  
depends 'apache', '>= 0.3.0'
```



EXERCISE

GL: Reconfigure Apache

So we want HAProxy to serve content of port tcp/80, and Apache to serve of tcp/8080.

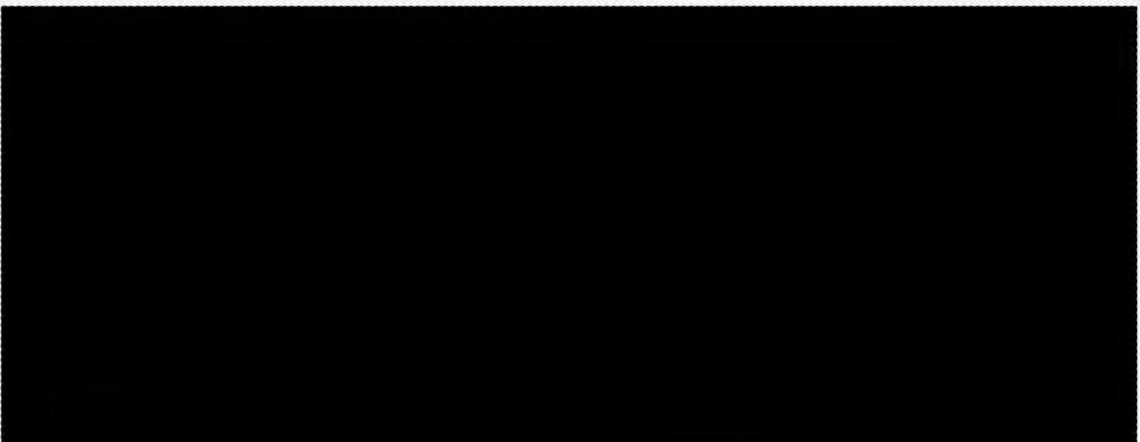
Objective:

- ✓ Reconfigure Apache to serve content of port tcp/8080
- ✓ Reconfigure HAProxy to contact Apache on port tcp/8080
- Upload the Cookbook

GL: Upload the Cookbook



```
cd ~/chef-repo/cookbooks/haproxy
```



Lets review that lab.

You change into the directory for the 'myhaproxy' cookbook.



Configuring Berks

`berks install` retrieves dependencies from the Chef Supermarket by default.

We want to override this to use the local apache cookbook instead of the community one.

GL: Edit Berksfile

```
chef-repo/cookbooks/haproxy/Berksfile
```

```
source "https://supermarket.chef.io"  
metadata  
  
cookbook 'apache', path: '../apache'
```

GL: Upload the Cookbook



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at ../apache
Fetching 'haproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using apache (0.3.0) from source at ../apache
Using haproxy (0.2.0) from source at .
```

Run the command `berks install`.

GL: Upload the Cookbook



```
$ berks upload
```

```
Skipping apache (0.3.0) (frozen)
Uploaded haproxy (0.2.0) to:
'https://api.opscode.com:443/organizations/ORGNAME'
```

After installing all the necessary dependent cookbooks, we used 'berks upload' to send the cookbook and all its dependencies to the Chef Server. This is again an easier method to manage dependencies instead of manually identifying the dependencies and then uploading each single cookbook at a time.

The 'frozen' apache item is normal in this scenario.

DISCUSSION



SSH Woes

Logging into both systems is a pain. We can use another knife tool to allow us to send commands to all of our nodes.

We asked you to login to that remote node and run 'sudo chef-client' to apply the new run list defined for that node. This does in fact work but considering that we may need to execute this command for this node and many future nodes, it seems like a lot of windows and commands that we would need to execute.

GL: Using knife ssh



```
$ knife ssh --help
```

```
knife ssh QUERY COMMAND (options)
  -a, --attribute ATTR      The attribute to use for opening the connection -
  default depends on the context
  -s, --server-url URL    Chef Server URL
  --chef-zero-host HOST    Host to start chef-zero on
  --chef-zero-port PORT    Port (or port range) to start chef-zero on. Port ranges
like 1000,1010 or 8889-9999 will try all given ports until one works.
  -k, --key KEY            API Client Key
  --[no-]color              Use colored output, defaults to false on Windows, true
otherwise
  -C, --concurrency NUM    The number of concurrent connections
  -c, --config CONFIG      The configuration file to use
  --defaults               Accept default values for all questions
```

To make our lives easier, the 'knife' command provides a subcommand named 'ssh' that allows us to execute a command across multiple nodes that match a specified search query.

GL: Run chef-client on All Nodes



```
$ knife ssh "*:*" -x USERNAME -P PASSWORD "sudo chef-client"
```

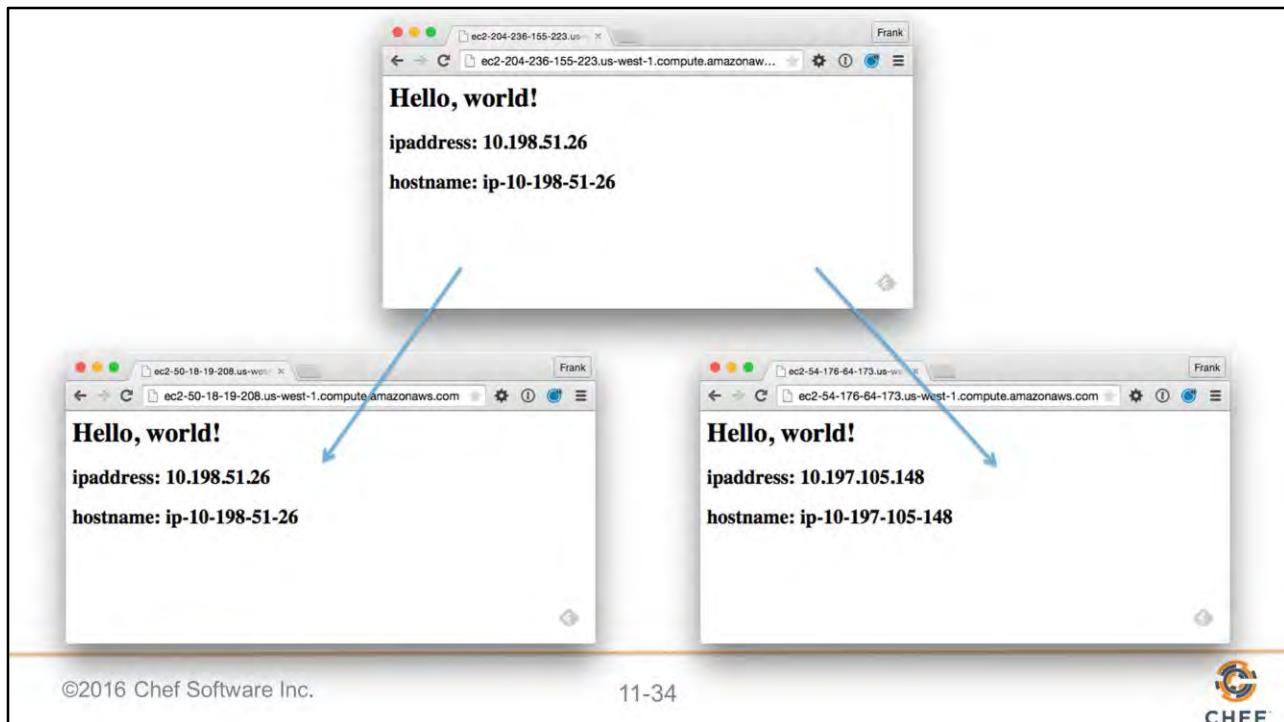
```
ec2-54-88-169-195.compute-1.amazonaws.com Starting Chef Client, version 12.4.4
ec2-54-84-233-7.compute-1.amazonaws.com   Starting Chef Client, version 12.4.4
ec2-54-88-185-159.compute-1.amazonaws.com Starting Chef Client, version 12.4.4
ec2-54-88-169-195.compute-1.amazonaws.com resolving cookbooks for run list: ["haproxy"]
ec2-54-84-233-7.compute-1.amazonaws.com   resolving cookbooks for run list: ["apache"]
ec2-54-88-169-195.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-84-233-7.compute-1.amazonaws.com   Synchronizing Cookbooks:
ec2-54-88-169-195.compute-1.amazonaws.com     - haproxy
ec2-54-84-233-7.compute-1.amazonaws.com     - apache
...
...
```

There are a lot of options for defining the search criteria that we will continue to explore. The most important criteria in this instance is star-colon-star. This means that we want to issue a command to all nodes.

So if you want to execute a "sudo chef-client" run for all of your nodes, you should write out this command.

You would need to provide the user name to log into the system, the password for that system, and then finally the command to execute.

In this way, you could easily ask your nodes to update from your current workstation as long as they all have the same login credentials. For more security, you should likely use SSH keys and forego specifying a username and password.



Point a web browser to the FQDN or public IP address of your load balancer. It should display the web pages of the web server nodes that the load balancer is configured to serve.

Refresh your browser a few times to see the round-robin effect.

Nothing should change externally. You may see some differences in the logs as the proxy configuration file might change the order of the two entries but the end results is that our proxy server node is still delivering traffic to our two web server nodes.



Discussion

Attributes are like parameters to your cookbook – no hard-coded values in recipes or templates.

Can you imagine in complex topologies, where you could have multiple levels of dependencies between cookbooks – berkself handles all of that out the box?

DISCUSSION



Q&A

What questions can we answer for you?

Before we continue let us stop for a moment answer any questions that anyone might have at this time.



CHEF™

Roles

Giving your Nodes a Role

©2016 Chef Software Inc.





Objectives

After completing this module, you should be able to:

- Assign roles to nodes so you can better describe them and configure them in a similar manner
- Set attribute values within roles

Roles



A role describes a run list of recipes that are executed on the node.

A role may also define new defaults or overrides for existing cookbook attribute values.

Up until this point it has been a mouthful to describe the nodes within our organization. We have two nodes, node1 and node3, that have the apache cookbook's default recipe in their run list. We have one node, node2, that has the myhaproxy cookbook's default recipe in its run list.

The Chef Server allows us to create and manage roles. A role describes a run list of recipes that are executed on the node. A role may also define new defaults or overrides for existing cookbook attribute values. Similar to what we accomplished with the wrapper cookbook.

A node may have zero or roles assigned to it.

Roles



You assign a role to a node in its run list.

This allows you to configure many similar nodes



GL: Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- Give our loadbalancer node a "loadbalancer" Role
- Give our web nodes a "web" Role

In this section you will create a proxy role and assign it to the run list of node2. You will also will create a web role and assign it to the run list of node1 and node3.

This is particularly powerful because we will no longer have to manage each of these identical nodes individually, instead we can make changes to the role that they share and all of the nodes that have this role will update accordingly.

GL: Create the loadbalancer.rb

```
chef-repo/roles/loadbalancer.rb
```

```
name 'loadbalancer'  
description 'load balancer'  
run_list 'recipe[haproxy]'
```

Create a file named `loadbalancer.rb`. This is a ruby file that contains specific methods that allow you to express details about the role. You'll see that the role has a name, a description, and run list.

The name of the role as a practice will share the name of the ruby file unless it cannot for some reason. The name of the role should clearly describe what it attempts accomplish.

The description of the role helps reinforce or clarify the intended purpose of the role. When selecting a role name that is not clear it is important that a helpful description is provided to help ensure everyone on the team understands its purpose.

The run list defines the list of recipes that give the role its purpose. Currently the proxy role defines a single recipe--the `myhaproxy` cookbook's default recipe.

GL: What Can 'knife role' Do?



```
$ cd ~/chef-repo
$ knife role --help

** ROLE COMMANDS **

knife role bulk delete REGEX (options)
knife role create ROLE (options)
knife role delete ROLE (options)
knife role edit ROLE (options)
knife role env_run_list add [ROLE] [ENVIRONMENT] [ENTRY[,ENTRY]] (options)
knife role env_run_list clear [ROLE] [ENVIRONMENT]
knife role env_run_list remove [ROLE] [ENVIRONMENT] [ENTRIES]
knife role env_run_list replace [ROLE] [ENVIRONMENT] [OLD_ENTRY] [NEW_ENTRY]
knife role env_run_list set [ROLE] [ENVIRONMENT] [ENTRIES]
knife role from file FILE [FILE..] (options)
```

GL: Upload It to the Chef Server & Verify

```
 $ knife role from file loadbalancer.rb
```

```
Updated Role loadbalancer!
```

```
$ knife role list
```

```
loadbalancer
```

Now you need to upload it to the Chef Server. This is done through the command 'knife role from file proxy.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file proxy.rb.

GL: View Details of the Role



```
$ knife role show loadbalancer
```

```
chef_type:          role
default_attributes:
description:        load balancer
env_run_lists:
json_class:         Chef::Role
name:               loadbalancer
override_attributes:
run_list:
```

GL: Set the loadbalancer role to node3



```
$ knife node run_list set node3 "role[loadbalancer]"
```

```
node3:  
  run_list: role[loadbalancer]
```

The last step is to redefine the run list for node2. We want the run list to contain only the proxy role.

Previously, we used the command 'knife node run_list add' to append a new item to the existing run list. There is also a command that allows us to remove an item from the run list. There is a command that allows us to set the run list to a value provided. This will replace the existing run list with a new one that we provide.

GL: Verify the Run List



```
$ knife node show node3
```

```
Node Name: node3
Environment: _default
FQDN: ip-172-31-55-82.ec2.internal
IP: 54.162.31.253
Run List: role[loadbalancer]
Roles:
Recipes: haproxy, haproxy::default
Platform: centos 6.8
Tags:
```

After you update the run list, you can verify that the node has the correctly-defined run list by running 'knife node show node2'.

GL: Converge All the load balancer Nodes



```
$ knife ssh "role:loadbalancer" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-162-31-253.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-162-31-253.compute-1.amazonaws.com resolving cookbooks for run list:
["haproxy"]
ec2-54-162-31-253.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-162-31-253.compute-1.amazonaws.com   - apache (0.3.0)
ec2-54-162-31-253.compute-1.amazonaws.com   - haproxy (0.2.0)
ec2-54-162-31-253.compute-1.amazonaws.com Installing Cookbook Gems:
ec2-54-162-31-253.compute-1.amazonaws.com Compiling Cookbooks...
...
ec2-54-162-31-253.compute-1.amazonaws.com Running handlers:
ec2-54-162-31-253.compute-1.amazonaws.com Running handlers complete
ec2-54-162-31-253.compute-1.amazonaws.com Chef Client finished, 0/4 resources
updated in 10 seconds...
```

You can use 'knife ssh' to run 'sudo chef-client' on all the nodes again to ensure that nothing has changed.

In this instance we only interested in having node2 run the command so we can get a little more creative with the search criteria and find nodes with the role proxy. In this case there is only one result.

Within the results, nothing should change. Switching over to the role did not change the fundamental recipes that were applied to the node.



Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- Give our load balancer node a "loadbalancer" Role
- Give our web nodes a "web" Role



GL: Define a Web Role

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- Create a role named 'web' that has the run list 'recipe[apache]'
- Set node1's run list to be "role[web]"
- Set node2's run list to be "role[web]"

In this lab, define a new role named 'web' that has the run list: including the apache cookbook's default recipe.

When you're done defining the role, upload it to the Chef Server, and then set the run list on node1 and node3 to the role that you have defined.

And for good measure, though nothing should have changed, run 'sudo chef-client' on both node1 and node3 to ensure that no functionality has been lost.

GL: Create the web.rb File

`chef-repo/roles/web.rb`

```
name 'web'  
description 'Web Server'  
run_list 'recipe[apache]'
```

First we create a file named `web.rb` in the `roles` directory.

The name of the role is `web`. The description should be `Web Server`. The run list you define should contain the `apache` cookbook's default recipe.

DISCUSSION



Role Attributes

If an attribute is set in a cookbook, and is also set in a role, then the role value wins!

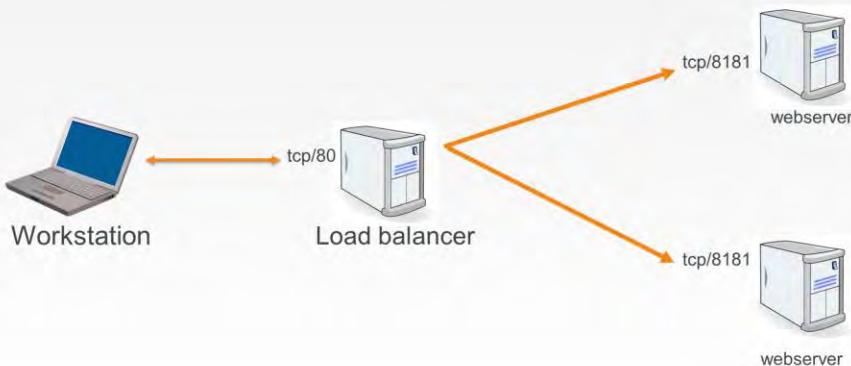


GL: Role Attributes

Objective:

Lets set apache to listen on tcp/8181 instead of tcp/8080 via the role.

GL: Our New Topology



With a single web server running with our organization, it's now time to talk about the next goal to tackle. We need to setup a proxy server.

A proxy server is able to receive requests and relay them to other systems. In our case, we specifically want to use the proxy server to balance the entire traffic load between one or more systems.

This means we will need to establish a new node within our organization, install the necessary software to make the node a proxy server, and configure it so that it will relay requests to our existing node running apache and to future nodes.

```
name 'web' description 'Web Server' run_list 'recipe[apache]' default_attributes({'apache' => {'port' => 8181}})
```

GL: Create the web.rb File

[chef-repo/roles/web.rb](#)

```
name 'web'  
description 'Web Server'  
run_list 'recipe[apache]'  
default_attributes({  
  "apache" => {  
    "port" => 8181  
  }  
})
```

First we create a file named web.rb in the roles directory.

The name of the role is web. The description should be Web Server. The run list you define should contain the apache cookbook's default recipe.

GL: Upload it to the Chef Server & Verify

```
 $ knife role from file web.rb
```

```
Updated Role web!
```

```
$ knife role list
```

```
loadbalancer  
web
```

Now you need to upload it to the Chef Server. This is done through the command 'knife role from file proxy.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file proxy.rb.

GL: Verify Specific Information About the Role



```
$ knife role show web
```

```
chef_type:          role
default_attributes:
  apache:
    port: 8181
description:        Web Server
env_run_lists:
json_class:         Chef::Role
name:               web
override_attributes:
run_list:
```

GL: Set Run List on node1 and node2

```
 $ knife node run_list set node1 "role[web]"
```

```
node1:  
  run_list: role[web]
```

```
$ knife node run_list set node2 "role[web]"
```

```
node2:  
  run_list: role[web]
```

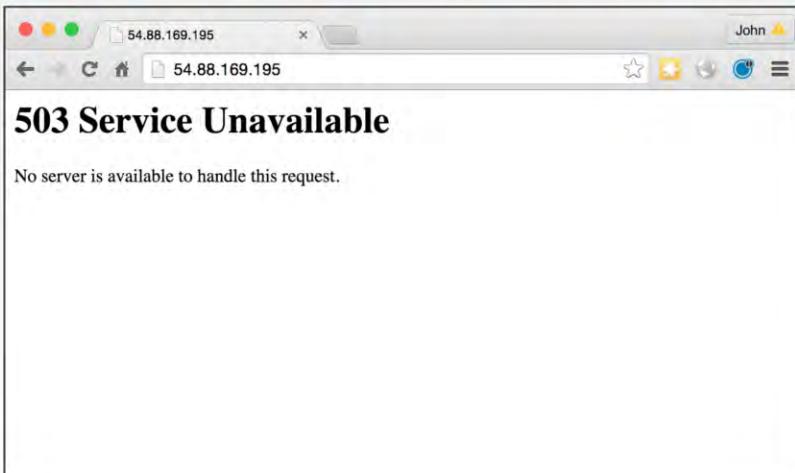
GL: Converge All Web Nodes



```
$ knife ssh "role:web" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-84-233-7.compute-1.amazonaws.com      - restart service service[httpd]
ec2-54-84-233-7.compute-1.amazonaws.com
ec2-54-84-233-7.compute-1.amazonaws.com      Running handlers:
ec2-54-84-233-7.compute-1.amazonaws.com      Running handlers complete
ec2-54-84-233-7.compute-1.amazonaws.com      Chef Client finished, 2/6 resources
updated in 9.758669459 seconds
ec2-54-88-185-159.compute-1.amazonaws.com    * service[httpd] action restart
ec2-54-88-185-159.compute-1.amazonaws.com      - restart service service[httpd]
ec2-54-88-185-159.compute-1.amazonaws.com
ec2-54-88-185-159.compute-1.amazonaws.com      Running handlers:
ec2-54-88-185-159.compute-1.amazonaws.com      Running handlers complete
ec2-54-88-185-159.compute-1.amazonaws.com      Chef Client finished, 2/6 resources
updated in 10.349332394 seconds
```

GL: Test the Load Balancer



We still need to configure our load balancer to pick up new apache port (tcp/8181)

Point a web browser to the URL or public IP address of your proxy server. It should display **'503 Service Unavailable' No server is available to handle this request.'**

So we still need to configure our load balancer to pick up new apache port (tcp/8181).

GL: Edit the loadbalancer.rb

chef-repo/roles/loadbalancer.rb

```
name 'loadbalancer'  
description 'load balancer'  
run_list 'recipe[haproxy]'  
default_attributes({  
  "apache" => {  
    "port" => 8181  
  }  
})
```

GL: Upload it to the Chef Server



```
$ knife role from file loadbalancer.rb
```

```
Updated Role loadbalancer!
```

Now you need to upload it to the Chef Server. This is done through the command 'knife role from file proxy.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file proxy.rb.

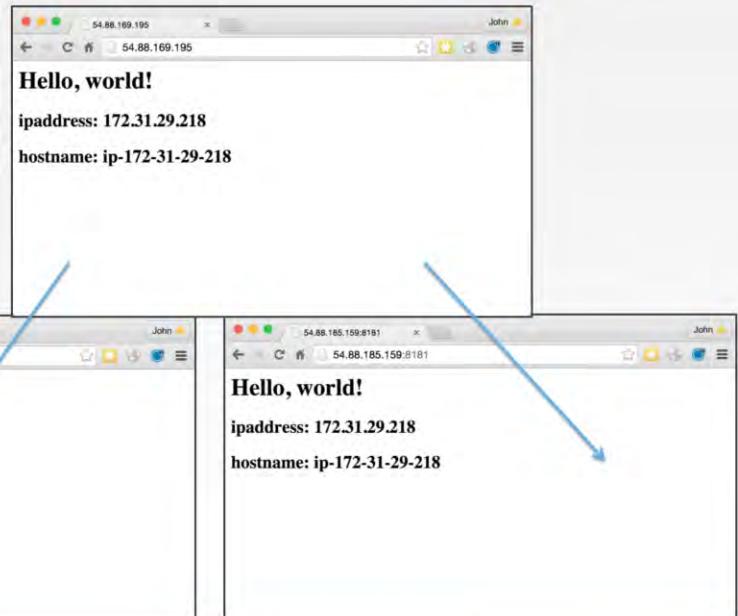
GL: Converge Load Balancer node



```
$ knife ssh "role:loadbalancer" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-88-169-195.compute-1.amazonaws.com      -      server app1 54.84.233.7:8080 weight 1
maxconn 100 check
ec2-54-88-169-195.compute-1.amazonaws.com      +      server app0 54.88.185.159:8181 weight 1
maxconn 100 check
ec2-54-88-169-195.compute-1.amazonaws.com      +      server app1 54.84.233.7:8181 weight 1
maxconn 100 check
ec2-54-88-169-195.compute-1.amazonaws.com      * service[haproxy] action start (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * service[haproxy] action enable (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * service[haproxy] action restart
ec2-54-88-169-195.compute-1.amazonaws.com      - restart service service[haproxy]
ec2-54-88-169-195.compute-1.amazonaws.com
ec2-54-88-169-195.compute-1.amazonaws.com Running handlers:
ec2-54-88-169-195.compute-1.amazonaws.com Running handlers complete
ec2-54-88-169-195.compute-1.amazonaws.com Chef Client finished, 2/5 resources updated in
9.831407575 seconds
```

GL: Test the Load Balancer



Point a web browser to the FQDN or public IP address of your load balancer. It should display the web pages of the web server nodes that the load balancer is configured to serve.

Refresh your browser a few times to see the round-robin effect.



Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- ✓ Give our loadbalancer node a "loadbalancer" Role
- ✓ Give our web nodes a "web" Role

With that we now have made it far easier to talk about our nodes. We can more casually describe a node as a 'web server' node or a 'proxy server' node.

In the future if we needed to ensure that these types of nodes needed to run additional recipes, we could return to the role file, update its run list, and then upload it to the Chef Server again.

Default Attribute Precedence



Please note this is a simplified diagram, and the precedence shown can be overridden.



Discussion

What are the benefits of using roles? What are the drawbacks?

Roles can contain roles. How many of these nested roles would make sense?

Roles are not version controlled – can you think of another way around this?

Answer these questions.

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



Q&A

What questions can we help you answer?



Community Cookbooks

Introducing the chef-client cookbook & wrapper cookbooks

Lesson Objectives

After completing this module, you should be able to:

- Find and preview cookbooks on the Chef Supermarket
- Use knife to work with the Chef Supermarket site API
- Override community cookbook defaults using wrapper cookbooks
- Upload community cookbooks to Chef Server
- Run chef-client as a service/task

Community Cookbooks



Someone already wrote that cookbook?

Available through the community site called
Supermarket

<https://supermarket.chef.io>

What if we told you someone already wrote that cookbook you need?

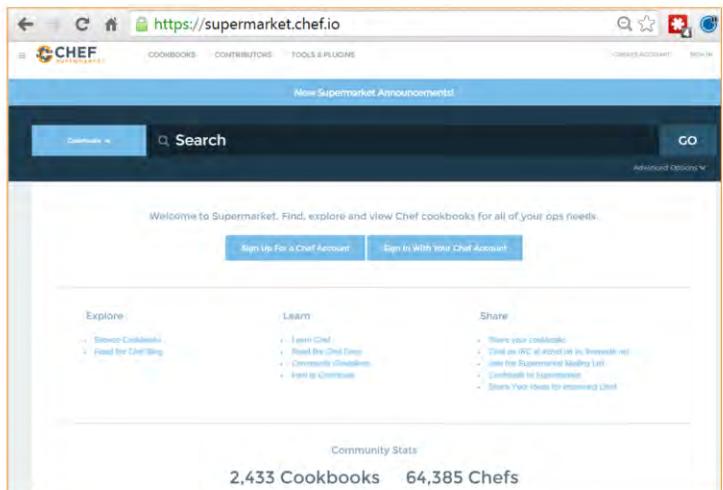
Someone already has and that cookbook is available through the community site called Supermarket. Supermarket is a public repository of all the cookbooks shared by other developers, teams, and companies who want to share their knowledge and hard work with you to save you time.

Demo: Log in to Supermarket

The screenshot shows a web browser displaying the Chef Supermarket homepage at <https://supermarket.chef.io>. The page features a navigation bar with links for COOKBOOKS, CONTRIBUTORS, TOOLS & PLUGINS, CREATE ACCOUNT, and SIGN IN. A search bar with the placeholder "Search" and a "GO" button are prominently displayed. Below the search bar, a message encourages users to "Sign in using your chef account". Two blue buttons are present: "Sign Up For a Chef Account" and "Sign In With Your Chef Account". An orange arrow points from the text "Log in to Supermarket" in the slide content to the "SIGN IN" button on the website. The footer of the page includes a copyright notice for ©2016 Chef Software Inc. and a link to page 13-4.

Demo: Community Cookbooks

- Community cookbooks are managed by individuals.
- Chef does not verify or approve cookbooks in the Supermarket.
- Cookbooks may not work for various reasons.
- Still, there are real benefits to community cookbooks.



An important thing to remember is that on the community site are cookbooks managed by individuals. Chef does not verify or approve the cookbooks found in the Supermarket. These cookbooks solved problems for the original authors and then they decided to share them. This means that the cookbooks you find in the Supermarket may not be built or designed for your platform. It may not take into special consideration your needs and requirements. It may no longer be actively maintained.

Even if the cookbook does not work as a whole, there is still value in reading and understanding 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.



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?

Exercise



GL: View How chef-client is Configured

In this group lab you will view how chef-client is configured.

GL: View chef-client config Directory



```
$ knife ssh "*:*" -x USER -P PWD "ls -F /etc/chef"
```

```
ec2-54-205-59-139.compute-1.amazonaws.com  client.pem  client.rb  first-boot.json      ohai/
ec2-54-167-232-148.compute-1.amazonaws.com  client.pem  client.rb  first-boot.json      ohai/
ec2-54-162-31-253.compute-1.amazonaws.com  client.pem  client.rb  first-boot.json      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.

GL: View chef-client Configuration



```
$ knife ssh "*:*" -x USER -P PWD "cat /etc/chef/client.rb"
```

```
ec2... log_location STDOUT
ec2-54-162-31-253.compute-1.amazonaws.com log_location STDOUT
ec2-54-162-31-253.compute-1.amazonaws.com chef_server_url "https://api.chef.io/organizations/ORG"
ec2-54-162-31-253.compute-1.amazonaws.com validation_client_name "chef-validator"
ec2-54-162-31-253.compute-1.amazonaws.com node_name "node3"
ec2-54-162-31-253.compute-1.amazonaws.com
ec2-54-205-59-139.compute-1.amazonaws.com log_location STDOUT
ec2-54-205-59-139.compute-1.amazonaws.com chef_server_url "https://api.chef.io/organizations/ORG"
ec2-54-205-59-139.compute-1.amazonaws.com validation_client_name "chef-validator"
ec2-54-205-59-139.compute-1.amazonaws.com node_name "node2"
ec2-54-205-59-139.compute-1.amazonaws.com
ec2-54-167-232-148.compute-1.amazonaws.com log_location STDOUT
...
...
```

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 client, in that they both communicate with the Chef Server over the API - the file ‘/etc/chef/client.rb’ is the equivalent to the ‘knife.rb’ file on the workstation.

DISCUSSION



Introducing the chef-client Cookbook

The chef-client cookbook allows you to manage and configure chef-client as a service on Linux-based nodes, or as a task on Windows nodes, configure logging, caching, etc.

Bootstrapping installs the chef-client executable.

The chef-client cookbook is used to configure chef-client.



Introducing chef-client Cookbook

We will use the chef-client community cookbook to configure chef-client on each of our nodes to run periodically

Objective:

- Upload chef-client cookbook to Chef Server
- Configure each of our nodes to run chef-client as a service

Discussed



Wrapper Cookbooks

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** to wrap upstream cookbooks and change their behavior without forking

See <https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/>

GL: Create my_chef_client Wrapper Cookbook



```
$ cd chef-repo  
$ chef generate cookbook cookbooks/my_chef_client
```

```
Generating cookbook my_chef_client  
- Ensuring correct cookbook file content  
- Committing cookbook files to git  
- Ensuring delivery configuration  
- Ensuring correct delivery build cookbook content  
- Adding delivery configuration to feature branch  
- Adding build cookbook to feature branch  
- Merging delivery content feature branch to master
```

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

```
...
```

Change to your chef-repo directory and then 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.

GL: Edit my_chef_client Default Recipe

cookbooks/my_chef_client/recipes/default.rb

```
#  
# Cookbook Name:: my_chef_client  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
include_recipe 'chef-client::default'
```

This recipe just calls the recipe `chef-client::default`

GL: Update my_chef_client metadata.rb

cookbooks/my_chef_client/metadata.rb

```
name          'my_chef_client'  
maintainer    'The Authors'  
maintainer_email 'you@example.com'  
license        'all_rights'  
description    'Installs/Configures my_chef_client'  
long_description 'Installs/Configures my_chef_client'  
version        '0.1.0'  
  
depends      'chef-client'
```

GL: Install the Dependencies

```
 $ cd cookbooks/my_chef_client  
$ berks install
```

```
Resolving cookbook dependencies...  
Fetching 'my_chef_client' from source at .  
Fetching cookbook index from https://supermarket.chef.io...  
Installing chef-client (5.0.0)  
Using chef_handler (1.4.0)  
Using compat_resource (12.14.0)  
Using cron (1.7.6)  
Using logrotate (2.1.0)  
Using my_chef_client (0.1.0) from source at .  
Using windows (1.44.3)
```

GL: View Berksfile.lock

 \$ cat Berksfile.lock

```
DEPENDENCIES
my_chef_client
  path: .
  metadata: true

GRAPH
chef-client (7.0.0)
cron (>= 1.7.0)
logrotate (>= 1.9.0)
windows (>= 1.42.0)
compat_resource (12.16.2)
...
```

GL: View the Berkshelf in Your Home Directory



```
$ ls -lt ~/.berkshelf/cookbooks/
```

```
drwxr-xr-x 12 YOU staff 408 2 Dec 15:46 chef_handler-1.1.6
drwxr-xr-x 12 YOU staff 408 2 Dec 15:46 cron-1.6.1
drwxr-xr-x 21 YOU staff 714 2 Dec 15:46 logrotate-1.8.0
drwxr-xr-x 15 YOU staff 510 2 Dec 15:46 windows-1.36.6
drwxr-xr-x 9 YOU staff 306 28 Nov 15:22 build-essential-2.1.3
drwxr-xr-x 25 YOU staff 850 6 Nov 16:27 build-essential-2.1.2
drwxr-xr-x 11 YOU staff 374 6 Nov 16:27 cpu-0.2.0
```

Note: Windows users can use `ls -force ~/.berkshelf/cookbooks/`

Berkshelf does not store dependent cookbook in the actual repo you're working in, but puts them in another location, usually under your home directory. This means your repo is not polluted with cookbooks that you neither need nor want. For example, the chef-client cookbook is designed to be multiplatform, and will work on all flavors of Linux as well as Windows. However, if you're a Linux shop you may not want Windows specific dependencies in your working repo.

GL: Upload Cookbooks to Chef Server



```
$ berks upload
```

```
Resolving cookbook dependencies...
Fetching 'my_chef_client' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Installing chef-client (5.0.0)
Using chef_handler (1.4.0)
Using compat_resource (12.14.0)
Using cron (1.7.6)
Using logrotate (2.1.0)
Using my_chef_client (0.1.0) from source at .
Using windows (1.44.3)
...
```

GL: View Cookbooks on Chef Server



```
$ knife cookbook list
```

```
apache          0.3.0
chef-client     7.0.0
compat_resource 12.16.2
cron            3.0.0
haproxy          0.2.1
logrotate        2.1.0
my_chef_client   0.1.0
ohai             4.2.2
windows          2.1.1
workstation      0.2.1
```



Introducing chef-client cookbook

We will use the chef-client community cookbook to configure chef-client on each of our nodes to run periodically

Objective:

- ✓ Upload chef-client cookbook to Chef Server
- ❑ Configure each of our nodes to run chef-client as a service

DISCUSSION



Lets Examine the chef-client Cookbook

We're going to use one recipe on our node from the chef-client cookbook.

`chef-client::service`
(via `chef-client::default`)

GL: View the chef-client::default Recipe

```
~/berkshelf/cookbooks/chef-client-<version>/recipes/default.rb
```

```
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
#  
  
if platform?('windows')  
  include_recipe 'chef-client::task'  
else  
  include_recipe 'chef-client::service'  
end
```

The default recipe just makes a call to the recipe 'chef-client::service'--this is the recipe that will set chef-client to run as a service.

GL: View the chef-client::service Recipe

~/berkshelf/cookbooks/chef-client-<version>/recipes/service.rb

- The recipe supports a number of **service** providers and styles.
- It works on a lot of **platforms**.
- Everything is controllable through **attributes**.

```
supported_init_styles = [
  'arch',
  'bluepill',
  'bsd',
  'daemontools',
  'init',
  'launchd',
  'runit',
  'smf',
  'upstart',
  'winsw'
]
init_style = node["chef_client"]["init_style"]
# Services moved to recipes
if supported_init_styles.include? init_style
  include_recipe "chef-client::#{init_style}_service"
else
  log "Could not determine service init style, manual
intervention required to start up the chef-client service."
end
```

There's a lot to this recipe so we won't to cover it all in detail.

There are several init styles that can be selected by setting the node attribute in the recipe. "init" is the default, and what we're going to use. Also available: smf, upstart, arch, runit, bluepill, daemontools, winsw. "cron" is a separate recipe since it is not technically running as a "service".

DISCUSSION



chef-client as a Service

We will add the chef-client default recipe to the roles for each node.

GL: Create the new 'base' role

`chef-repo/roles/base.rb`

```
name 'base'  
description 'Base Role'  
run_list 'recipe[my_chef_client]'
```

First we create a file named web.rb in the roles directory.

The name of the role is web. The description should be Web Server. The run list you define should contain the apache cookbook's default recipe.

GL: Upload it to the Chef Server



```
$ knife role from file base.rb
```

```
Updated Role base!
```

Now you need to upload it to the Chef Server. This is done through the command 'knife role from file proxy.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file proxy.rb.

GL: Add the base Role to loadbalancer Role

`chef-repo/roles/loadbalancer.rb`

```
name 'loadbalancer'  
description 'loadbalancer'  
run_list 'role[base]', 'recipe[haproxy]'
```

GL: Add the base Role to web Role

chef-repo/roles/web.rb

```
name 'web'  
description 'Web Server'  
run_list 'role[base]', 'recipe[apache]'  
default_attributes({  
  "apache" => {  
    "port" => 8181  
  }  
})
```

<http://bit.ly/236fthv>

First we create a file named web.rb in the roles directory.

The name of the role is web. The description should be Web Server. The run list you define should contain the apache cookbook's default recipe.

GL: Upload the roles File

```
└─$ cd ~/chef-repo/roles  
└─$ knife role from file web.rb loadbalancer.rb
```

```
Updated Role web  
Updated Role loadbalancer
```

You need to share the role with the Chef Server so upload that file.

Use the command 'knife role from file web.rb'. 'knife' knows where to look for that role to upload it.

GL: Converge All Nodes



```
$ knife ssh "*:*" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-162-31-253.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-167-232-148.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-205-59-139.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-167-232-148.compute-1.amazonaws.com resolving cookbooks for run list: ["chef-client", "apache"]
ec2-54-205-59-139.compute-1.amazonaws.com resolving cookbooks for run list: ["chef-client", "apache"]
ec2-54-162-31-253.compute-1.amazonaws.com resolving cookbooks for run list: ["chef-client", "haproxy"]
ec2-54-167-232-148.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-205-59-139.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-162-31-253.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-205-59-139.compute-1.amazonaws.com   - chef-client (5.0.0)
...
...
```

To verify that everything is working the same as before, run 'knife ssh' for both of these nodes. In this instance the query syntax is going to find all nodes with the role set to web.

GL: Verify chef-client is Running



```
$ knife ssh "*:*" -x USER -P PWD "ps awux | grep chef-client"
```

```
ec2-514-88-185-159.compute-1.amazonaws.com root      10369  0.0 10.1 266928 61116 ?
S1 12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 1800 -s 300
...
ec2-514-88-169-195.compute-1.amazonaws.com root      14922  0.0 10.1 267020 61092 ?
S1 12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 1800 -s 300
...
ec2-514-814-233-7.compute-1.amazonaws.com  root      10202  0.0 10.1 267036 61096 ?
S1 12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 1800 -s 300
...
```

Its using our full-stack installation path automatically, and the -d says "runs as a daemon in the background"

We log to /var/log/chef/client.log (configurable). The -i is the interval, a number of seconds (30 minutes here) (configurable) and the last bit is splay (mitigating the thundering herd problem)

Instructor Note: embedded dir path--talk about Omnibus installer if not mentioned already.



Introducing chef-client cookbook

We will use the chef-client community cookbook to configure chef-client on each of our nodes to run periodically

Objective:

- ✓ Upload chef-client cookbook to Chef Server
- ✓ Configure each of our nodes to run chef-client as a service

DISCUSSION



Change Default Settings

Wait!

There has just been a mandate that every node in the infrastructure must run chef-client every 5 minutes.

Example: Setting the chef-client run Interval

```
chef-repo/cookbooks/chef-client/attributes/default.rb

...
default['chef_client']['log_file']      = 'client.log'
default['chef_client']['interval']     = '1800'
default['chef_client']['splay']         = '300'
default['chef_client']['conf_dir']      = '/etc/chef'
default['chef_client']['bin']          = '/usr/bin/chef-client'

...
```

We need to change the value of the attribute

`default['chef_client']['interval']` from 1800 (seconds) to
300 (...but don't change it in this file.)

So we need to change the attribute `default['chef_client']['interval']` in `chef-client` cookbook.

But what if a new cookbook version is released and we want to upgrade to it?

We'd need to re-implement these changes!

- Maintenance nightmare – especially if its been refactored!
- Good practice would be to not edit a community cookbook that you have wrapped.
- So we will set the interval attribute as shown on the next slide.

GL: Update the 'base' Role

`chef-repo/roles/base.rb`

```
name 'base'
description 'Base Role'
run_list 'recipe[my_chef_client]'
default_attributes({
  "chef_client" => {
    "interval" => 300
  }
})
```

Edit the file named `base.rb` in the `roles` directory as shown in this slide.

```
name 'base'
description 'Base Role'
run_list 'recipe[my_chef_client]'
default_attributes({
  "chef_client" => {
    "interval" => 300
  }
})
```

GL: Upload it to the Chef Server



```
$ cd ~/chef-repo/roles  
$ knife role from file base.rb
```

```
Updated Role base!
```

Now you need to upload it to the Chef Server. This is done through the command 'knife role from file base.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file base.rb.

GL: Converge All Nodes



```
$ knife ssh "*:*" -x USER -P PWD "sudo chef-client"
```

```
...
ec2-514-88-169-195.compute-1.amazonaws.com      * service[chef-client] action restart
ec2-514-88-169-195.compute-1.amazonaws.com      - restart service service[chef-client]
ec2-514-814-233-7.compute-1.amazonaws.com
ec2-514-814-233-7.compute-1.amazonaws.com      Running handlers:
ec2-514-814-233-7.compute-1.amazonaws.com      Running handlers complete
ec2-514-814-233-7.compute-1.amazonaws.com      Chef Client finished, 2/15 resources
updated in 14.158836577 seconds
ec2-514-88-169-195.compute-1.amazonaws.com
ec2-514-88-169-195.compute-1.amazonaws.com      Running handlers:
ec2-514-88-169-195.compute-1.amazonaws.com      Running handlers complete
ec2-514-88-169-195.compute-1.amazonaws.com      Chef Client finished, 2/14 resources
updated in 14.179414969 seconds
```

To verify that everything is working the same as before, run 'knife ssh' for both of these nodes. In this instance the query syntax is going to find all nodes with the role set to web.

GL: Verify chef-client is Running



```
$ knife ssh "*:*" -x USER -P PWD "ps awux | grep chef-client"
```

```
ec2-514-88-169-195.compute-1.amazonaws.com root      15626  0.0 10.1 266936 61120 ?
S1 13:21  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 300 -s 300
...
ec2-514-88-185-159.compute-1.amazonaws.com root      11209  0.0 10.0 266904 61016 ?
S1 13:21  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 300 -s 300
...
ec2-514-814-233-7.compute-1.amazonaws.com  root      10906  0.0 10.1 267016 61096 ?
S1 13:21  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 300 -s 300
...
```

Its using our full-stack installation path automatically, and the -d says "runs as a daemon in the background"

We log to /var/log/chef/client.log (configurable). The -i is the interval, a number of seconds (30 minutes here) (configurable) and the last bit is splay (mitigating the thundering herd problem)

Note: embedded dir path -- talk about Omnibus installer if not mentioned already

Discussion



Using Community Cookbooks

Chef Community Cookbooks, can be used as is
But in most cases you will want to use them as a
reference 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

Discussion



Q&A

What questions can we help you answer?

- Chef Supermarket
- Wrapper Cookbooks
- chef-client

What questions can we help you answer?

In general or about specifically about Chef Super Market, wrapper cookbooks, node attributes, the 'knife ssh' command.



©2016 Chef Software Inc.

Environments

Using Environments to Reflect Organization Patterns and Workflow

©2016 Chef Software Inc.





Objectives

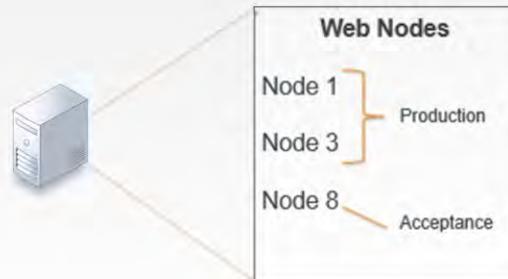
After completing this module, you should be able to:

- Create an environment
- Deploy a node to an environment
- Update a search query to be more exact



Environments

Environments can define different functions of nodes that live on the same system.



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.



Environments

Every organization or infrastructure starts with the `_default` environment.



Chef also has a concept of an environment. Chef uses environments to map an organization's real-life workflow to what can be configured and managed using the Chef server.

Every organization begins with a single environment called the `_default` (underscore default) environment, which cannot be modified or deleted.

Therefore, you must create custom environments to define your organization's workflow.



GL: Production Environment

Let's create a reliable environment for our nodes.

Objective:

- Deploy Our Site to Production

First, we need to create a Production environment. This is where we lock down our infrastructure and policies to a specific version of the myhaproxy cookbook.

GL: Searching All of Our Nodes



```
$ cd ~/chef-repo  
$ knife search node "*:*"
```

```
3 items found

Node Name:    node3
Environment: _default
FQDN:        ip-172-31-55-82.ec2.internal
IP:          54.211.221.35
Run List:    role[loadbalancer]
Roles:       loadbalancer
Recipes:     my_chef_client, my_chef_client::default, haproxy, haproxy::defaul
Platform:   centos 6.8
...
```

GL: Create an environments Directory



```
$ mkdir environments  
$ cd environments
```

First, we need to make a new environments directory. (Be sure you are still in the chef-repo before you do this.)

GL: Create the production.rb

```
chef-repo/environments/production.rb

name 'production'
description 'Where we run production code'

cookbook 'apache', '= 0.3.0'
override_attributes({
  "apache" => {
    "port" => 8181
  }
})
```

Then we need to create a production.rb file. Like in the roles.rb files, we must provide a name and description.

Additionally, we need to define cookbook restrictions to lock down specific versions of both the apache and myhaproxy cookbooks. By adding this information to production.rb, we are telling our nodes to use these specific versions of these specific cookbooks.

Obviously, what this means is that as we work on newer versions of these cookbooks, we won't break anything in the production environment.

Okay, so now that we have captured our 'good' environment in this file, let's save it and upload it.

GL: Upload the production.rb File



```
$ knife environment from file production.rb
```

```
Updated Environment production
```

```
$ knife environment show production
```

```
chef_type:           environment
cookbook_versions:
  apache: = 0.3.0
default_attributes:
description:        Where we run production code
json_class:         Chef::Environment
name:               production
override_attributes:
  apache:
    port: 8181
```

Using the knife environment command, let's upload the production.rb file. This should be familiar because it is just like the command we used to upload roles.

GL: Set node1 to production Environment

```
$ knife node environment set node1 production
```

```
node1:  
  chef_environment: production
```

```
$ knife node show node1
```

```
Node Name: node1  
Environment: production  
FQDN: ip-172-31-60-9.ec2.internal  
IP: 54.162.99.75  
Run List: role[web]  
Roles: web  
Recipes: my_chef_client, my_chef_client::default, apache, apache::default, ...  
Platform: centos 6.8  
Tags:
```

So, let's do that for node1.

The results don't really tell us much, so let's take a look at node1.



GL: Set More Nodes to Production

- ❑ Set node2 & node3's environment to production

GL: Set node2's Environment to Production

 \$ knife node environment set node2 production

```
node2:  
  chef_environment: production
```

\$ knife node environment set node3 production

```
node3:  
  chef_environment: production
```

GL: Verify Nodes are in Production Environment



```
$ knife search node "*:*"
```

```
...
Node Name: node3
Environment: production
FQDN: ip-172-31-55-82.ec2.internal
IP: 54.211.221.35
Run List: role[loadbalancer]
Roles: loadbalancer
Recipes: my_chef_client, my_chef_client::default, haproxy, haproxy::default, chef-client::default,
Platform: centos 6.8
Tags:

Node Name: node2
Environment: production
```

GL: Converge All Web Nodes



```
$ knife ssh "role:web" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-236-195-37.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-162-99-75.compute-1.amazonaws.com  Starting Chef Client, version 12.13.37
ec2-54-236-195-37.compute-1.amazonaws.com resolving cookbooks for run list:
["my_chef_client", "apache"]
ec2-54-162-99-75.compute-1.amazonaws.com  resolving cookbooks for run list:
["my_chef_client", "apache"]
ec2-54-236-195-37.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-236-195-37.compute-1.amazonaws.com      - my_chef_client (0.1.0)
ec2-54-236-195-37.compute-1.amazonaws.com      - chef-client (5.0.0)
ec2-54-236-195-37.compute-1.amazonaws.com      - cron (1.7.6)
ec2-54-236-195-37.compute-1.amazonaws.com      - logrotate (2.1.0)
ec2-54-162-99-75.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-236-195-37.compute-1.amazonaws.com      - compat_resource (12.14.0)
...
```

To verify that everything is working the same as before, run 'knife ssh' for all nodes. In this instance the query syntax is going to find all nodes with the role set to web.



Production

Let's create a reliable environment for our nodes.

Objective:

- ✓ Deploy our site to Production

Discussion



Acceptance Environment

Scenario: There is a new mandate requirement for the **Public IP**, the **Public Hostname** and the **Environment** to be displayed on the homepage.

So we need to update the apache cookbook and it needs to be fully tested before going into production.

DISCUSSION



Acceptance Environment

In the next steps we need to spin up a new node in a new environment called the Acceptance.

And then we'll test the new version of apache cookbook against the new node.



GL: Acceptance Environment

- Bump the Apache cookbook version and update the homepage accordingly
- Create an environment named "acceptance"
- Bootstrap a new web node into the acceptance environment using the new apache cookbook
- Run chef-client on all the nodes

Now, let's create the environment we can use to change and update the cookbooks without affecting our production environment. A sandbox, if you will.

Let's call this our "Acceptance environment".

GL: Bump the Cookbook Version

cookbooks/apache/metadata.rb

```
name          'apache'
maintainer    'The Authors'
maintainer_email 'you@example.com'
license        'all_rights'
description    'Installs/Configures apache'
long_description 'Installs/Configures apache'
version        '0.3.1'
```

GL: Update index.html.erb

```
chef-repo/cookbooks/apache/templates/default/index.html.erb
```

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

Ensure your entire file looks like this slide, including html tags:

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

The variable `node.chef_environment` holds the name of the environment that the node is running it. Since the search is being invoked from the HAProxy loadbalancer, then the value of this variable will be 'production'.

This means that this search... `allwebservers = search('node', "role:web AND chef_environment:#'{node.chef_environment}'")` ...is effectively saying this: `allwebservers = search('node', "role:web AND chef_environment:production")`

If the HAProxy was in the environment 'acceptance', then the search would become this: `allwebservers = search('node', "role:web AND chef_environment:acceptance")`

This means the load balancer will always only find the web servers that are in the same environment as itself.

GL: Upload the Cookbook

```
 $ cd cookbooks/apache  
$ berks install
```

```
Resolving cookbook dependencies...  
Fetching 'apache' from source at .  
Fetching cookbook index from https://supermarket.chef.io...  
Using apache (0.3.1) from source at .
```

```
$ berks upload
```

```
Uploaded apache (0.3.1) to: 'https://api.opscode.com:443/organizations/ORGNAME'
```

We use the Berkshelf to upload our cookbooks. This is where Berkshelf really shines as a tool.

Run the command "berks install".

When you run this command for a cookbook that has a dependency, you'll see that Berkshelf will download the haproxy cookbook and its dependencies as well. The haproxy cookbook is dependent on the build-essential cookbook and the cpu cookbook. If any of those cookbooks had dependencies, berkshelf would find those and download them as well.

Instructor Note: Berkshelf downloads these cookbooks into a common directory within your home path. They are not added alongside your other cookbooks.



GL: Acceptance Environment

- ✓ Bump the Apache cookbook version and update the homepage accordingly
- ❑ Create an environment named "acceptance"
- ❑ Bootstrap a new web node into the acceptance environment using the new apache cookbook
- ❑ Run chef-client on all the nodes

GL: Create a New Environment File

`chef-repo/environments/acceptance.rb`

```
name 'acceptance'  
description 'Where code and applications are tested'  
  
cookbook 'apache', '= 0.3.1'  
override_attributes({  
  "apache" => {  
    "port" => 8181  
  }  
})
```

First, let's create a new rb file in our chef-repo/environments directory. Let's name it acceptance.

In the Acceptance environment, we don't want to lock-down the cookbook versions, so we are not going to place restrictions on the cookbooks.

```
name 'acceptance'  
description 'Where code and applications are tested'  
  
cookbook 'apache', '= 0.3.1'  
override_attributes({  
  "apache" => {  
    "port" => 8181  
  }  
})
```

GL: Upload the .rb File

```
 $ cd ~/chef-repo/environments  
$ knife environment from file acceptance.rb
```

```
Updated Environment acceptance
```

```
$ knife environment list
```

```
_default  
production  
acceptance
```

Let's upload that .rb file to the Chef server.



GL: Acceptance Environment

- ✓ Bump the Apache cookbook version and update the homepage accordingly
- ✓ Create an environment named "acceptance"
- ❑ Bootstrap a new web node into the acceptance environment using the new apache cookbook
- ❑ Run chef-client on all the nodes

GL: Bootstrap a New Node into the Acceptance Environment



```
$ knife bootstrap FQDN -x USER -P PWD --sudo -N node4 -r 'role[web]' -E acceptance
```

```
Connecting to ec2-52-90-193-197.compute-1.amazonaws.com
ec2-52-90-193-197.compute-1.amazonaws.com -----> Existing Chef installation detected
ec2-52-90-193-197.compute-1.amazonaws.com Starting the first Chef Client run...
ec2-52-90-193-197.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-52-90-193-197.compute-1.amazonaws.com resolving cookbooks for run list:
["my_chef_client", "apache"]
ec2-52-90-193-197.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-52-90-193-197.compute-1.amazonaws.com   - my_chef_client (0.1.0)
ec2-52-90-193-197.compute-1.amazonaws.com   - chef-client (5.0.0)
ec2-52-90-193-197.compute-1.amazonaws.com   - logrotate (2.1.0)
ec2-52-90-193-197.compute-1.amazonaws.com   - cron (1.7.6)
ec2-52-90-193-197.compute-1.amazonaws.com   - compat_resource (12.14.0)
ec2-52-90-193-197.compute-1.amazonaws.com   - windows (1.44.3)
```

GL: Verify that the Environment Was Set



```
$ knife node show node4
```

```
Node Name:    node4
Environment:  acceptance
FQDN:        ip-172-31-53-224.ec2.internal
IP:          52.90.193.197
Run List:    role[web]
Roles:       web
Recipes:     my_chef_client, my_chef_client::default, apache, apache::default,
             chef-client::default, chef-client::service
Platform:   centos 6.8
Tags:
```



GL: Acceptance Environment

- ✓ Bump the Apache cookbook version and update the homepage accordingly
- ✓ Create an environment named "test"
- ✓ Bootstrap a new web node into the test environment using the new apache cookbook
- Run chef-client on all the nodes

GL: Update Load Balancing Pool



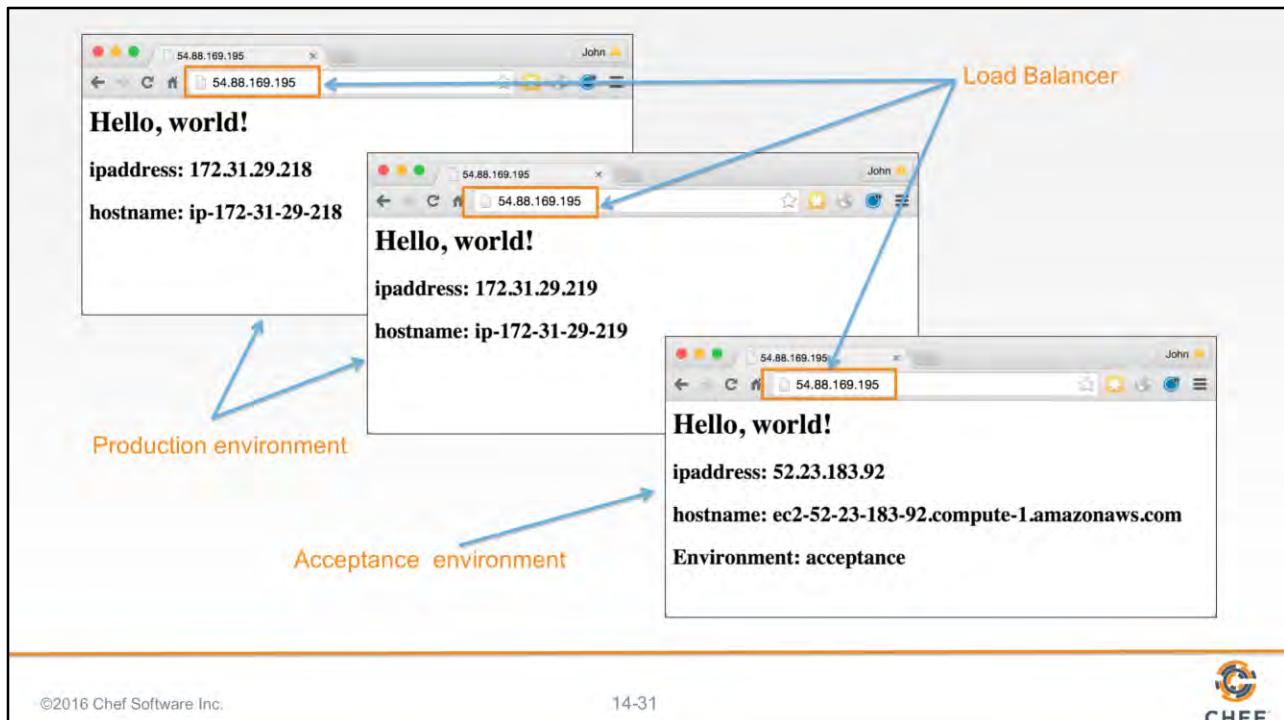
```
$ knife ssh "role:loadbalancer" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-211-221-35.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-211-221-35.compute-1.amazonaws.com resolving cookbooks for run list:
["my_chef_client", "haproxy"]
ec2-54-211-221-35.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-211-221-35.compute-1.amazonaws.com   - my_chef_client (0.1.0)
ec2-54-211-221-35.compute-1.amazonaws.com   - chef-client (5.0.0)
ec2-54-211-221-35.compute-1.amazonaws.com   - cron (1.7.6)
ec2-54-211-221-35.compute-1.amazonaws.com   - logrotate (2.1.0)
ec2-54-211-221-35.compute-1.amazonaws.com   - compat_resource (12.14.0)
ec2-54-211-221-35.compute-1.amazonaws.com   - windows (1.44.3)
ec2-54-211-221-35.compute-1.amazonaws.com   - chef_handler (1.4.0)
ec2-54-211-221-35.compute-1.amazonaws.com   - haproxy (0.2.0)
...
...
```



GL: Acceptance Environment

- ✓ Bump the Apache cookbook version and update the homepage accordingly
- ✓ Create an environment named "acceptance"
- ✓ Bootstrap a new web node into the acceptance environment using the new apache cookbook
- ✓ Run chef-client on all the nodes



Point your web browser at your load balancer (i.e., node3). You will notice that the load balancer is distributing traffic to each node across all environments in a round robin fashion. You can refresh your browser a number of times to see this effect.

Also notice that the Acceptance node is displaying its environment because of the changes we made to the version 0.3.1 of the apache cookbook. `chef-repo/cookbooks/apache/templates/default/index.html.erb`

DISCUSSION



Separating Environments

The load balancer is distributing load across all environments.

In this section you will modify the load balancer to distribute to nodes in a "Production" environment only.



Separating Environments

Objective:

- Use Search to separate out the environments



Balancing Nodes

Which cookbook handles balancing the requests between web nodes?

Which recipe within that cookbook sets up the request balancing between the two nodes?

How do we do that?

First, let's answer a couple of questions. As you think about the infrastructure we have created, which cookbook handles balance requests between nodes?

So if we want to make changes to that cookbook, which recipe would we change?

Answer 1: myhaproxy Answer 2: default.rb

Search Criteria

`chef-repo/cookbooks/haproxy/recipes/default.rb`

```
#  
# Cookbook Name:: myhaproxy  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
allwebservers = search('node', 'recipes:apache\:\:default')  
  
template '/etc/haproxy/haproxy.cfg' do  
#...
```

Looking at the default.rb file in the myhaproxy cookbook, we can review the original search syntax. If we want to search by environments, what would we need to add here?

GL: Modify the myhaproxy default.rb

chef-repo/cookbooks/haproxy/recipes/default.rb

```
#  
# Cookbook Name:: myhaproxy  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
allwebservers = search('node', "role:web AND chef_environment:#{{node.chef_environment}}")  
  
template '/etc/haproxy/haproxy.cfg' do  
#...
```

```
#  
# Cookbook Name:: myhaproxy  
# Recipe:: default  
#  
# Copyright (c) 2016 The Authors, All Rights Reserved.  
  
allwebservers = search('node', "role:web AND  
chef_environment:#{{node.chef_environment}}")  
  
template '/etc/haproxy/haproxy.cfg' do  
#...
```

GL: Bump the haproxy Cookbook Version

`chef-repo/cookbooks/haproxy/metadata.rb`

```
name 'haproxy'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'all_rights'
description 'Installs/Configures haproxy'
long_description 'Installs/Configures haproxy'
version '0.3.0'

depends 'apache', '>= 0.3.0'
```

GL: Run 'berks install'



```
$ cd cookbooks/haproxy
```

```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at ../apache
Fetching 'haproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using apache (0.3.1) from source at ../apache
Using haproxy (0.3.0) from source at .
```

```
$ berks upload
```

```
Skipping apache (0.3.1) (frozen)
Uploaded haproxy (0.3.0) to: 'https://api.chef.io:443/organizations/ORG'
```

We are going to need to use Berks to upload this cookbook because it has dependencies. So first we need to cd into the cookbook.

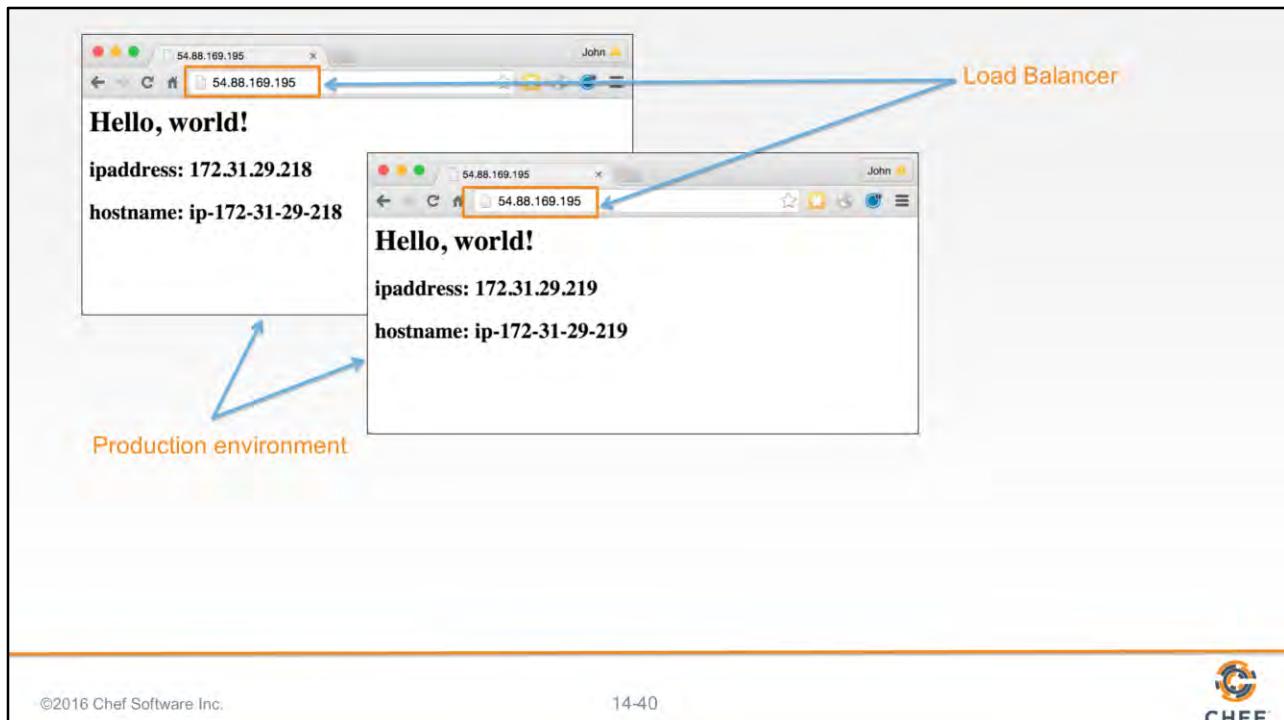
Then run 'berks install'.

GL: Run chef-client on all nodes



```
$ knife ssh "*:*" -x USERNAME -P PASSWORD "sudo chef-client"
```

```
...
ec2-54-88-169-195.compute-1.amazonaws.com      * directory[/var/log/chef] action create (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * directory[/etc/chef] action create (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * template[/etc/init.d/chef-client] action create (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * template[/etc/sysconfig/chef-client] action create (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * service[chef-client] action enable (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com      * service[chef-client] action start (up to date)
ec2-54-88-169-195.compute-1.amazonaws.com
ec2-54-88-169-195.compute-1.amazonaws.com Running handlers:
ec2-54-88-169-195.compute-1.amazonaws.com Running handlers complete
ec2-54-88-169-195.compute-1.amazonaws.com Chef Client finished, 0/13 resources updated in 28.735845396 seconds
```



If you point your browser to your load balancer, you'll see it is only distributing traffic to the web servers that are in its own environment (i.e., Production, not Acceptance). This is because of the changes we made to the search in **chef-repo/cookbooks/haproxy/recipes/default.rb**.



GL: Separate Environments

- ✓ Use Search to separate out the environments.



A Brief Recap

We restricted the production environment to specific cookbook version.
We created an acceptance environment with no cookbook restrictions.
We set specific nodes to each of these environments.
We updated the haproxy's default recipe to include environment search criteria.
And we changed the version number in the myhaproxy metadata.rb file.



Q&A

What questions can we help you answer?



Further Resources

Other Places to Talk About, Practice, and Learn Chef

©2016 Chef Software Inc.





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



Practice Chef

First, let's talk about stuff you can read to help you learn Chef.



learn.chef.io

Interactive learning for those new to Chef.

Another great place to practice Chef is our awesome Learn Chef site. These interactive modules provide you with an opportunity to run through exercises similar to those we did in this class, and it is updated when new Chef features are introduced. This is one of the most robust self-guided tutorial sites out there.



Beyond Essentials

What happens during a knife bootstrap?

What happens during a chef-client run?

What is the security model used by chef-client
Explains Attribute Precedence

<https://learn.chef.io/skills>

There is a special section of Learn Chef called the Skills Library where you will find some additional content that will answer some of the questions that you may have after completing this content. It is included there on Learn Chef to provide you with a resource that you can return back to again-and-again after this training.

<https://learn.chef.io/skills>



Community Resources

Example Cookbooks, Articles, Podcasts, and More

<https://github.com/obazoud/awesome-chef>

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://github.com/obazoud/awesome-chef>



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.

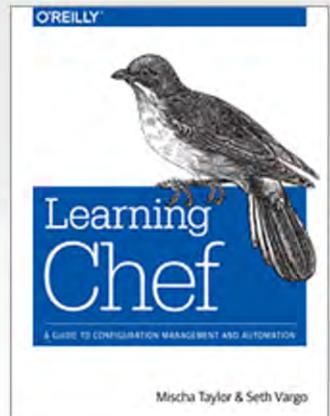
Remember, how often we referred to the Docs site throughout this workshop. That wasn't by accident. We wanted you to become comfortable with using our Docs site to resolve issues and learn about the many Chef tools out there.

Docs are there, available to you, 24 hours a day, 7 days a week. Any question you have, you probably will find the answer on our Docs site.

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

Learning Chef

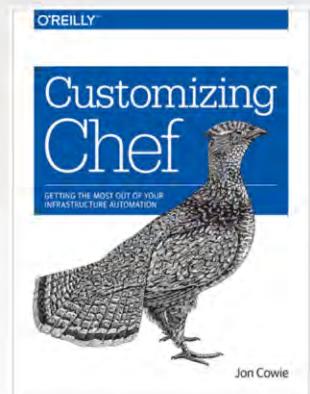
A Guide to Configuration Management
and Automation



Some people who have used Chef for years have written some excellent books about Chef. Check out Mischa Taylor's and Seth Vargo's *Guide to Configuration Management and Automation*. You can find it on O'Reilly. It's a great book.

Customizing Chef

Getting the Most Out of Your
Infrastructure Automation

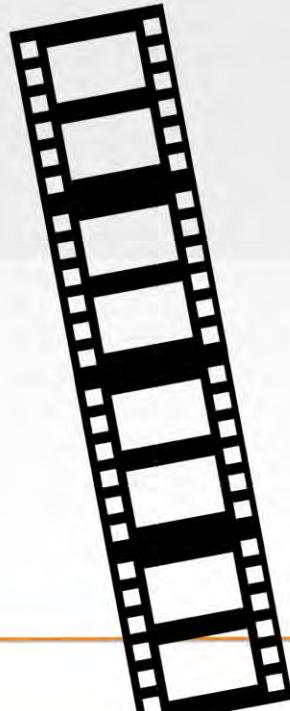


Additionally, you may want to read Jon Cowie's *Getting the Most Out of Your Infrastructure Automation*. It's also available on O'Reilly.

YouTube Channel

- ChefConf Talks
- Training Videos

<https://www.youtube.com/user/getchef/playlists>



We have uploaded a number of videos to the Chef YouTube channel, including training videos and talks from past Chef conferences.

<https://www.youtube.com/user/getchef/playlists>

arresteddevops.com

Arrested DevOps is the podcast that helps you achieve understanding, develop good practices, and operate your team and organization for maximum DevOps awesomeness.

<https://www.arresteddevops.com/>



<https://www.arresteddevops.com/>

foodfightshow.org



The Podcast where DevOps chefs do battle

Food Fight is a bi-weekly podcast for the Chef community. We bring together the smartest people in the Chef community and the broader DevOps world to discuss the thorniest issues in system administration.



Chef Developers' IRC Meeting

<https://github.com/chef/chef-community-irc-meetings>

Join members of the Chef Community in a weekly meeting for Chef Developers where we'll discuss the future of the Chef project and other things pertinent to the community. The agenda and schedule can be found at this link.

<https://github.com/chef/chef-community-irc-meetings>



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>

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>



ChefConf is a gathering of hundreds of Chef community members. We get together to learn about the latest and greatest in the industry (both the hows and the whys), as well as exchange ideas, brainstorm solutions, and give hugs, which has become the calling card of the DevOps community, and the Chef community in particular.

ChefConf 2017 will be held in Austin, Texas during May.

<https://www.chef.io/chefconf/>



**SHARE
YOUR
VOICE**

CHEF

CALL FOR PRESENTATIONS NOW OPEN!

CHEFCONF 2017 | AUSTIN, TX | MAY 22-24

CFP CLOSES JANUARY 18TH



CHEFTM

Thank You!