



CHEFTM

Chef Training Services

Chef Infra Foundations

Participant Guide



Chef Infra Foundations

Introduction

1W

Course v3.1.0

©2020 Chef Software Inc.

1-1

 CHEF

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

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

This course was tested on Chef Workstation version: 0.15.18 so please use that version of Chef Workstation. The Linux workstation we provide already has Chef Workstation version: 0.15.18 on it. But later in the course you'll need to install that version on your laptop.

Instructor Note: Be sure to read Appendix Z found at the end of the instructor guide for training lab set up notes and additional instructor notes. The "1W" on this slide means you will need 1 windows WS for each student in this module.



Introductions

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

- Introduce ourselves
- Introduce this training experience

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

Introduce Ourselves

Name

Current job role

Previous job roles/background

Experience with Chef and/or config management

Favorite Text Editor

Location

Instructor Note: Often times with groups I opt to have people introduce themselves to each other in a more casual fashion. Asking the learners to face one another and ask each other these questions. During that time you can walk around and insert yourself into the introductions to help odd numbers and learn more about the learners.



Introductions

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

- ✓ Introduce ourselves
- Introduce this training experience

Now that we know a little bit more about each other, let's take some time to introduce the class.

Course Objectives

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

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

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

Agenda: Day 1

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

Agenda: Day 2

- ❖ Apache Lab
- ❖ Workstation Installation
- ❖ Sign up for a Managed Chef account
- ❖ Chef Server
- ❖ Policyfiles
- ❖ Attribute Files and Dependencies

Agenda: Day 3

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

- ❖ Optional: Data Bags

Introductions



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

- ✓ Introduce ourselves
- ✓ Introduce this training experience

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

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



Pre-built Workstation

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

- Login to the Remote Workstation

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

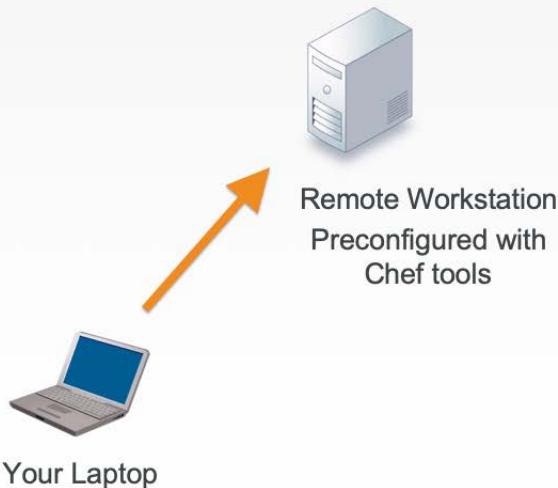
Setting up Your Workstation

On Day 2 we will install the necessary tools on your workstation.



Near the beginning of Day 2 we have set aside time to install the necessary tools on your local computer. If you already have the tools we will ensure that they are working correctly and troubleshoot any issues to ensure you have a smooth experience for when you leave this training.

Chef Lab System Architecture



For now you are going to use your local computer to connect to a remote workstation. This workstation has all the necessary tools installed.

Logging in to the Workstation

Use the **address**, **user name**, and **password** to connect to the remote workstation.



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

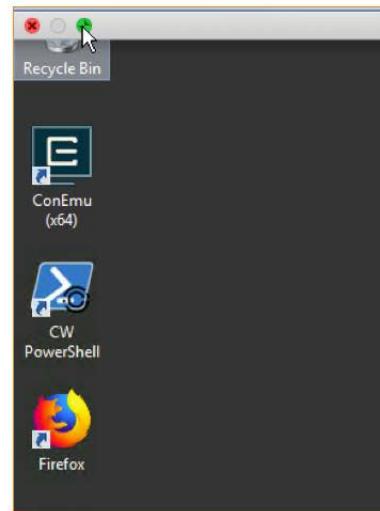
Instructor Note: You should assign the participants their Day 1 virtual workstations (AMIs) at this time. The login credentials and password for the virtual workstations are:

user: Administrator
password: Automat3Me!

Logging in to the Workstation

Tip for Mac users:

If you don't want the remote desktop to consume your entire screen, you can click the green icon shown here to reduce and resize it.





Pre-built Workstation

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

Objective:

- ✓ Login to the Remote Workstation



©2020 Chef Software Inc.

Chef Resources and Recipes

Chef's Fundamental Building Blocks

Objectives



After completing this module, you should be able to:

- Define Chef Resources
- Create a basic Chef recipe file
- Use Chef to set the policy on your workstation
- Use the chef-client command

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



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/resource_reference.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.

Resource Type Examples:

- file
- template
- remote_file
- package
- service
- directory
- user
- group
- cron

- python
- reboot
- powershell_script
- execute



Resource Definition

```
file 'C:\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 'C:\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'.



Resource Definition

```
file 'C:\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 'C:\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 properties 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.

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



Resource Definition

```
file 'C:\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.

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

Example: powershell_script

```
powershell_script 'Install IIS' do
  code 'Add-WindowsFeature Web-Server'
  action :run
end
```

The powershell_script resource named 'Install IIS' is run with the code 'Add-WindowsFeature Web-Server'.

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

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

Example: service

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

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

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

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

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

Example: file

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

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

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

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

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

Example: file

```
file 'C:\PHP\php.ini' do
  action :delete
end
```

The file name 'c:\PHP\php.ini' is deleted.

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

In this example, the file named 'C:\PHP\php.ini' 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.



Group Lab: Hello, World?

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

Objective:

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

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 'C:\hello.txt' with the contents 'Hello, world!'.

GL: Start the Chef Workstation Powershell



Double-click the **Chef Workstation Powershell** shortcut on your virtual workstation's desktop.

It may take a few seconds for the prompt to display.



GL: Ensure You Are in Your Home Dir

> **pwd**

C:\Users\Administrator\Desktop>

Important: A lot of the work you'll do today will be done within your home directory (on Windows it's **C:\Users\Administrator**) or in a subdirectory to your home directory. So you should always ensure you are in the proper directory as indicated in these course materials.

Generally, you can type **cd ~** to return to your home directory if you had navigated away from your home directory.

When opening up Powershell you are dropped into the **C:\Users\Administrator\Desktop** directory.

GL: Move to Your Home Directory



```
> cd ~
```

```
C:\Users\Administrator
```

This is our working directory for today.

Your ChefDK command line prompt in Windows should always show your current directory.

GL: List the Contents of Your Home Directory



> dir

```
Directory: C:\Users\Administrator

Mode                LastWriteTime     Length Name
<-----              -----          ----- ----
d----    8/3/2018 10:33 PM           .atom
d----    7/11/2020 4:00 PM           .chef
d----    6/26/2020 7:21 PM           .chef-workstation
d-r--   11/13/2017 8:03 PM           Contacts
d-r--   6/28/2020 4:38 PM           Desktop
d-r--   11/13/2017 8:26 PM           Documents
d-r--   7/11/2020 2:31 PM           Downloads
d-r--   11/13/2017 8:03 PM           Favorites
d-r--   8/3/2018 10:21 PM           Links
...
...
```

GL: Create and Open a Recipe File



```
> atom hello.rb
```

It could take some time for Atom to launch the first time you launch it.

Now that we have seen a few examples of resources let's get to work creating a text file. Using your editor open the file named 'hello.rb'. 'hello.rb' is a recipe file. It has the extension '.rb' because it is a ruby file.

GL: Create a Recipe File Named hello.rb

```
~\hello.rb
```

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

Type these contents into your hello.rb file in Atom.

The file named 'C:\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 'C:\hello.txt'. We also are stating that the contents of that file should contain 'Hello, world!'.
Save the file and return to the command prompt.

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

GL: Create a Recipe File Named hello.rb



In Atom, the blue dot here indicates your changes have not been saved yet.

So save your changes using the **File** menu or **Ctrl > s**.

```
hello.rb — C:\Users\Administrator — Atom
File Edit View Selection Find Packages Help
New Window Ctrl+Shift+N
New File Ctrl+N
Open File... Ctrl+O
Open Folder... Ctrl+Shift+O
Add Project Folder... Ctrl+Shift+A
Reopen Project
Reopen Last Item Ctrl+Shift+T
Settings Ctrl+Comma
Config...
Init Script...
Keymap...
Snippets...
Stylesheet...
Save Ctrl+S
Save As... Ctrl+Shift+S
```

```
1 file 'C:\hello.txt' do
2 content 'Hello, world!'
3 end
4
```



Group Lab: Hello, World?

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

Objective:

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

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



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 a Recipe File



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

```
[2020-07-12T16:53:48+00:00] WARN: No cookbooks directory found at or above current directory.  
Assuming C:/Users/Administrato.r.  
✓ 2 product licenses accepted.  
+-----+  
...  
Recipe: @recipe_files::C:/Users/Administrator/hello.rb  
* file[C:\hello.txt] action create  
  - create new file C:\hello.txt  
  - update content in file C:\hello.txt from none to 315f5b  
  --- C:\hello.txt      2020-07-12 16:53:57.798746200 +0000  
  +++ C:\chef-hello20200712-3056-1rev5o6.txt  2020-07-12 16:53:57.798746200 +0000  
  @@ -1 +1,2 @@  
  +Hello, world!  
...
```

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

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

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

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

GL: What Does hello.txt Say?



```
> gc C:\hello.txt
```

```
Hello, world!
```

The **gc** command stands for Get Content in Windows. It's similar to the **cat** command in Linux systems.

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



Group Lab: Hello, World?

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

Objective:

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



Discussion

What would happen if you ran the command again?

What would happen if the file were removed?

What happens when I run the command again?

Again, before you run the command -- think about it. What are your expectations now from the last time you ran it? What will the output look like?

And, of course, what would happen if the file was removed?



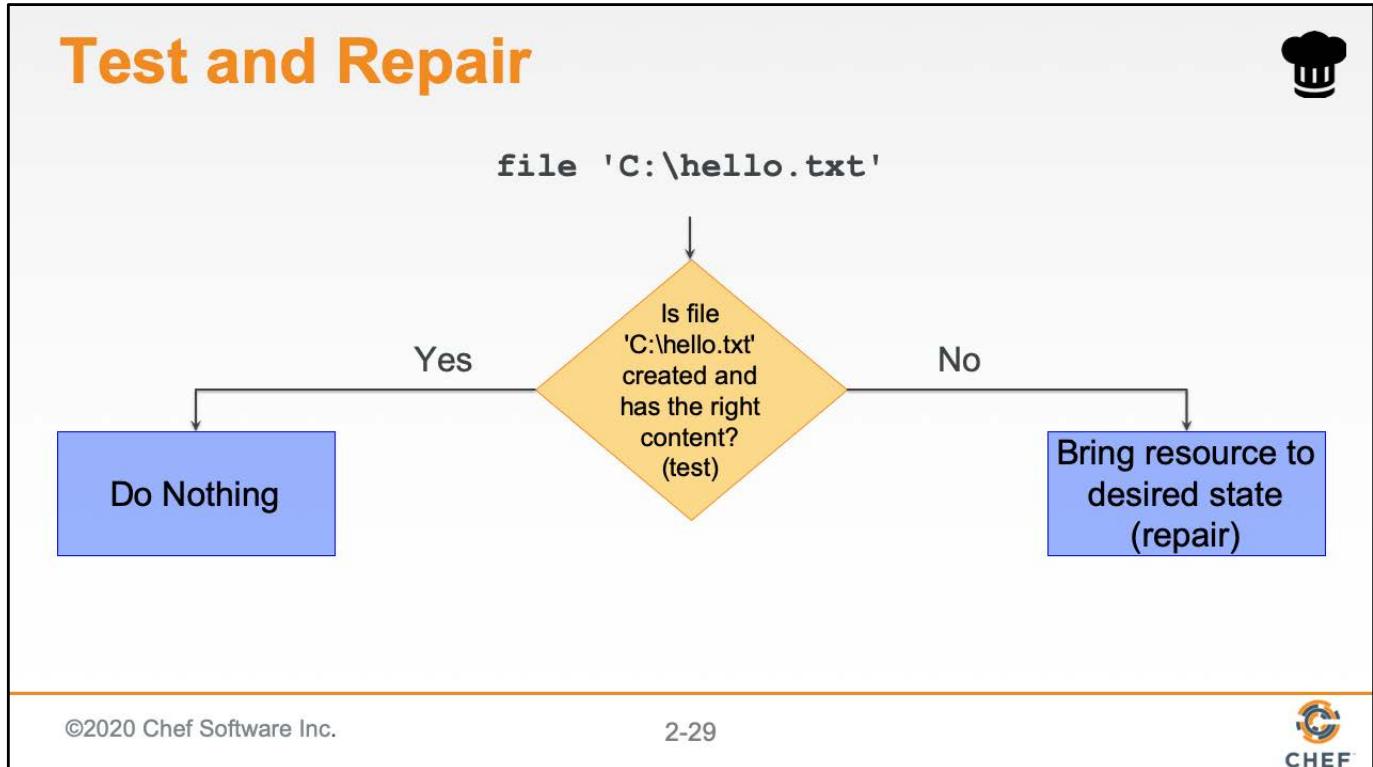
Test and Repair

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

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

Hopefully it is clear from running the `chef-client` command a few times that the resource we defined only takes action when it needs to take action.

We call this test and repair. Test and repair means the resource is first tested on the system before it takes action.



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

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



Lab: Goodbye Recipe

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

Objective:

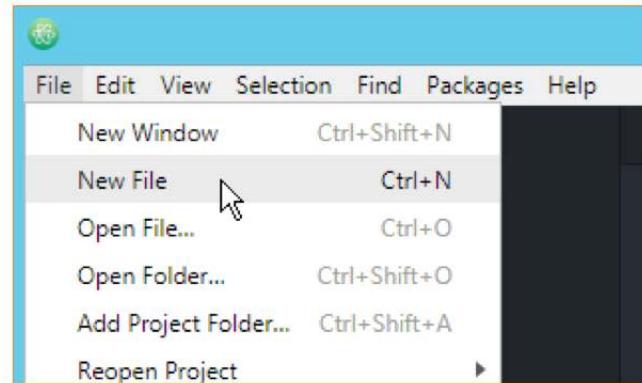
- Create a recipe named 'goodbye.rb' that removes 'C:\hello.txt'
- Apply the recipe to the workstation

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

Lab: Atom Tip



In the next step you will create another file under the Administrator directory. You can do that in Atom by clicking **File > New File** and then naming the file via **Save As**.



Lab: Adding a file Resource to Delete a File

```
~\goodbye.rb
```

```
file 'C:\hello.txt' do
  action :delete
end
```

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

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



Lab: Goodbye Recipe

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

Objective:

- Create a recipe that removes 'C:\hello.txt'
- Apply the recipe to the workstation

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

Lab: Apply a Recipe File



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

```
resolving cookbooks for run list: []
...
[2020-06-27T16:17:42+00:00] WARN: Node WIN-NJ5007IAJNR.ec2.internal has an
empty run list.
Converging 1 resources

Recipe: @recipe_files::C:/Users/Administrator/goodbye.rb

  * file[C:\hello.txt] action delete
    - delete file C:\hello.txt
```

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

Lab: Test that the File was Deleted



```
> Test-Path C:\hello.txt
```

```
False
```

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



Lab: Goodbye Recipe

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

Objective:

- ✓ Create a recipe that removes 'C:\hello.txt'
- ✓ Apply the recipe to the workstation

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



GL: Disable Limited User Account

Managing files is nice but what about Registry keys?

Objective:

- Create a recipe that disables Limited User Account
- Apply that recipe

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

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

GL: Disable the Limited User Account

```
~\disable-uac.rb
```

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

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

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

Instructor Note: The lab that follows this group exercise will use the same registry key so the learner will need to use the variable. The use of the variable here also makes the column length smaller so that the font size on the slide can remain at a reasonable size without the content breaking across multiple lines.

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

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

```
}]  
end
```

GL: Disable the Limited User Account

~\disable-uac.rb

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

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

Here we are using a new resource named 'registry_key' that takes the name of a registry key. We then provide to the values attribute the values we want to set/insert in the registry. Here we are setting the EnableLUA key to have a dword value of 0. This will make it so that Windows will no longer notify the user when programs try to make changes to the computer.

See the following documentation for more information:
<https://technet.microsoft.com/en-us/library/ff715520.aspx>.



GL: Disable Limited User Account

Managing files is nice but what about Registry keys?

Objective:

- Create a recipe that disables Limited User Account
- Apply that recipe

The policy you defined in the recipe has now updated the system and you should now be prompted that Limited User Account has been disabled.

GL: Apply a Recipe File



```
> chef-client --local-mode disable-uac.rb
```

```
Starting Chef Infra Client, version 15.0.300
```

```
...
```

```
Converging 1 resources
```

```
Recipe: @recipe_files::C:/Users/Administrator/disable-uac.rb
```

```
  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]
    action create
```

```
    - set value {:name=>"EnableLUA", :type=>:dword, :data=>0}
```

```
Running handlers:
```

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



GL: Disable Limited User Account

Managing files is nice but what about Registry keys?

Objective:

- ✓ Create a recipe that disables Limited User Account
- ✓ Apply that recipe

The policy you defined in the recipe has now updated the system and you should now be prompted that Limited User Account has been disabled.



Lab: Disable Consent Prompt

- Update the recipe file named "disable-uac.rb" to also define:

The registry_key named
'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System' has the
values:

```
[ { :name => 'ConsentPromptBehaviorAdmin', :type => :dword, :data => 0 } ]
```

- Use **chef-client** to apply the recipe file named "disable-uac.rb"

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

Instructor Note: Allow 10 minutes to complete this exercise.

Lab: Disable the Limited User Account

```
~\disable-uac.rb
```

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

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

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

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

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

Instructor Note: The previous registry key resource is represented here with a comment to allow more space for the new registry key being added.



Lab: Disable Consent Prompt

- ✓ Update the recipe file named "disable-uac.rb" to also define:

The registry_key named
'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System' has the
values:

```
[ { :name => 'ConsentPromptBehaviorAdmin', :type => :dword, :data => 0 } ]
```

- Use **chef-client** to apply the recipe file named "disable-uac.rb"

Lab: Apply a Recipe File



```
> chef-client --local-mode disable-uac.rb
```

```
Starting Chef Infra Client, version 15.0.300
...
[2020-06-27T16:25:53+00:00] WARN: Node WIN-NJ5007IAJNR.ec2.internal has an empty run list.
Converging 2 resources

Recipe: @recipe_files::C:/Users/Administrator/disable-uac.rb

  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System] action
    create (up to date)

  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System] action
    create

  - set value {:name=>"ConsentPromptBehaviorAdmin", :type=>:dword, :data=>0}
```

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



Lab: Disable Consent Prompt

- ✓ Update the recipe file named "disable-uac.rb" to also define:

The registry_key named
'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System' has the
values:

```
[ { :name => 'ConsentPromptBehaviorAdmin', :type => :dword, :data => 0 } ]
```

- ✓ Use **chef-client** to apply the recipe file named "disable-uac.rb"



Review Questions

What is a resource?

What is a Chef recipe?

What is chef-client and what does it do?

Answer these questions:

Q: What is a resource?

A resource is a statement of configuration policy. It describes the desired state of an element of your infrastructure and the steps needed to bring that item to the desired state.

Q: What is a Chef recipe?

It's mostly a collection of resources defined using patterns (resource names, attribute-value pairs, and actions).

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

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.



Q&A

What questions can we answer for you?

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

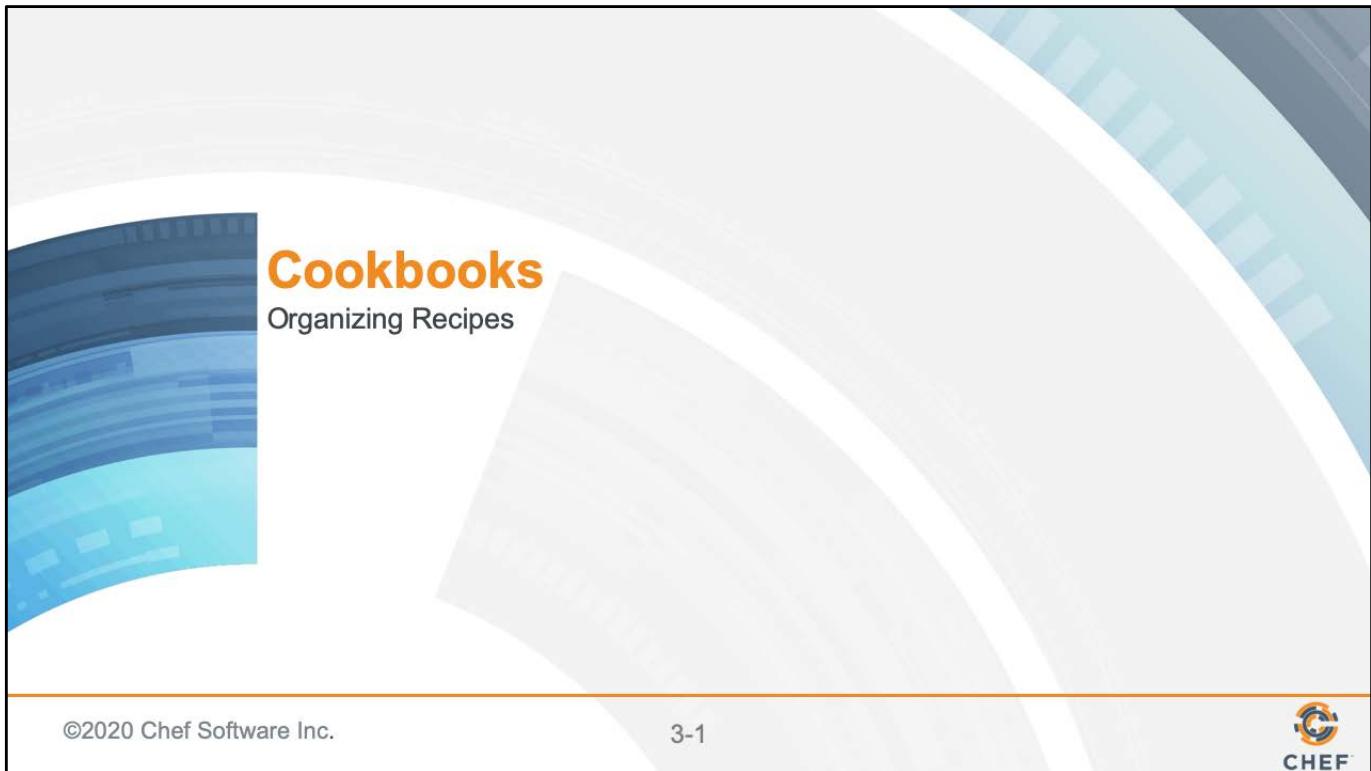
What questions can we answer for you?

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



CHEFTM

©2020 Chef Software Inc.



Objectives

After completing this module you should be able to:

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

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

Collaboration and Version Control

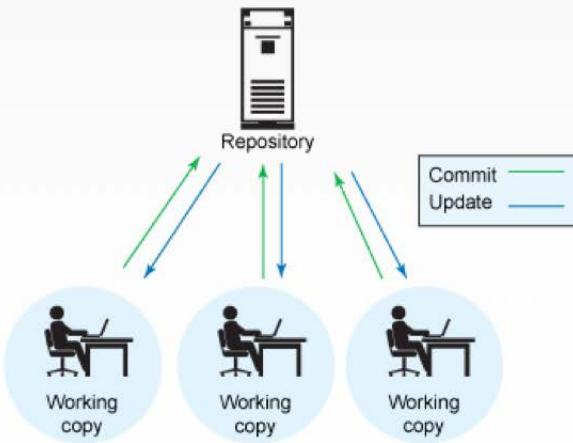
A versioning system should include:

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

Each revision should be stored as a new version.

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

Centralized version control system



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

A versioning system should include:

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

Each revision should be stored as a new version.

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

Versioning Pros and Cons

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

Saving a copy of the original file as another filename.

Let's explore this first option of renaming the file by adding a quick extension, like in the first example shown here. In this way we can keep working on the original file as we add more features. As a group let's talk about the pros and cons of using this strategy.

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



How about we use git? What are the pros and cons of this approach?

For the rest of this course we will be using git. This may not be the version control software you use on your teams or within your organization and that is alright. Our use of git within this course is used solely to demonstrate the use of version control when developing Chef code. When you develop with Chef you are welcome to use the version control system of your choice.

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

Git Version Control

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

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

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

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



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

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

GL: Setup Your Global git Configuration



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

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

Set up your email you want to associate with your git commits. Run this from
C:\Users\Administrator

GL: Setup Your Global git Configuration



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

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

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

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>



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.



Cookbook

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

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



GL: Setting up a Workstation

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

Objective:

- Create a cookbook
- Use git
- Copy the disable-uac recipe within the cookbook
- Use the include_recipe method to insert disable-uac recipe
- Apply the default recipe to the workstation

The disable-uac recipe is one of many recipes that we could define to setup our workstations. But before we 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'.

Instructor Note:

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

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

GL: Create a 'cookbooks' Directory



```
> mkdir cookbooks
```

```
Directory: C:\Users\Administrator
```

Mode	LastWriteTime	Length	Name
d----	7/12/2020 4:28 PM		cookbooks

We will want to have a location for our cookbooks to reside with the 'Administrator' directory so let's create a 'cookbooks' folder.

GL: Locating the cookbooks Directory



> dir

```
Directory: C:\Users\Administrator
d----          8/3/2018 10:33 PM      .atom
d----          6/27/2020 3:54 PM      .chef
d----          6/26/2020 7:21 PM      .chef-workstation
d-r--          11/13/2017 8:03 PM      Contacts
d----          6/27/2020 4:56 PM      cookbooks
d-r--          6/26/2020 7:50 PM      Desktop
...
d----          6/27/2020 3:54 PM      nodes
d-r--          11/13/2017 8:03 PM      Pictures
...
-a---          6/27/2020 4:56 PM      48 .gitconfig
-a---          6/27/2020 4:24 PM      349 disable-uac.rb
-a---          6/27/2020 4:17 PM      47 goodbye.rb
-a---          6/26/2020 7:49 PM      56 hello.rb
-a---          11/13/2017 8:26 PM      880 kitchen-template.yml
```



What is 'chef'?

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

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

The Chef Development Kit (Chef DK) comes with a tool named 'chef'. This command-line tool has a number of features.

GL: What Can 'chef' Do?



```
> chef --help
```

Usage:

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

Available Commands:

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

'chef' is a command-line application that does quite a few things. The most important thing to us right now is its ability to generate cookbooks and components.

GL: What Can 'chef generate' Do?



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

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

```
Available generators:
```

cookbook	Generate a single cookbook
recipe	Generate a new recipe
attribute	Generate an attributes file
template	Generate a file template
file	Generate a cookbook file
lwrp	Generate a lightweight resource/provider
repo	Generate a Chef policy repository
policyfile	Generate a Policyfile for use with the install/push commands (experimental)

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: What Can 'chef generate cookbook' Do?



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

```
Usage: chef generate cookbook NAME [options]
```

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

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

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

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

GL: Let's Create a Cookbook



```
> chef generate cookbook cookbooks\workstation
```

```
+-----+
```

```
Chef License Acceptance
```

```
Before you can continue, 1 product license
```

```
must be accepted. View the license at
```

```
https://www.chef.io/end-user-license-agreement/
```

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

```
License that need accepting:
```

```
* Chef Workstation
```

```
Do you accept the 1 product license (yes/no)?
```

```
> yes
```

```
Persisting 1 product license...
```

```
1 product license persisted
```

```
+-----+
```

```
Generating cookbook workstation
```

We have you covered. Call the cookbook workstation. That's a generic enough name.

We want you to use 'chef generate' to generate a cookbook named workstation.

GL: The Cookbook Has a README



```
> tree /f cookbooks\workstation
```

```
Folder PATH listing
Volume serial number is B04A-119C
C:\USERS\ADMINISTRATOR\COOKBOOKS\WORKSTATION
|
|   ...
|   .kitchen.yml
|   Berksfile
|   chefignore
|   metadata.rb
|   README.md
|
|---recipes
|     default.rb
|
```

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.

GL: Let's Take a Look at the README



```
> gc cookbooks\workstation\README.md
```

```
# workstation
```

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

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 Some Metadata



```
> tree /f cookbooks\workstation
```

```
Folder PATH listing
Volume serial number is B04A-119C
C:\USERS\ADMINISTRATOR\COOKBOOKS\WORKSTATION
|
|   ...
|   .kitchen.yml
|   Berksfile
|   chefignore
|   metadata.rb
|   README.md
|
|---recipes
|     default.rb
|
```

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.

GL: Let's Take a Look at the Metadata



> gc cookbooks\workstation\metadata.rb

```
name 'workstation'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures workstation'
long_description 'Installs/Configures workstation'
version '0.1.0'
chef_version '>= 14.0'
```

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

If you view the contents of your new cookbook's metadata, you'll see a number of details that help describe the cookbook:

The name of the cookbook, its maintainer, a way to reach them, how the cookbook is licensed, descriptions, and the cookbook's version number.

GL: The Cookbook Has a Folder for Recipes



```
> tree /f cookbooks\workstation
```

```
C:\USERS\ADMINISTRATOR\COOKBOOKS\WORKSTATION
+-- kitchen.yml
+-- LICENSE
+-- metadata.rb
+-- Policyfile.rb
+-- README.md
+-- .delivery
    +-- project.toml

+-- recipes
    +-- default.rb
...
```

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



```
> gc cookbooks\workstation\recipes\default.rb
```

```
# Cookbook:: workstation

# Recipe:: default

#
# Copyright:: 2020, 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.



Group Exercise: Version Control

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

- ✓ Use chef to generate a cookbook to store our disable-uac recipe.
- ❑ Add the "workstation" cookbook to version control.

Now that we have our cookbook with its README and version number, it's time to start tracking our changes with git.

GL: Move Your Recipe into the Cookbook



```
> mv disable-uac.rb cookbooks\workstation\recipes
```

GL: Move to Cookbook Directory and Initialize as a git Repository



```
> cd cookbooks\workstation  
> git init
```

```
Reinitialized existing Git repository in  
C:/Users/Administrator/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 detected that we have git installed and automatically initialized the cookbook as a git repository and added the new files to the local git repo.



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.

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



```
> git add .
```

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

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

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.

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

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



```
> git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
new file:   recipes/disable-uac.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.

Note: We only see the recipes/disable-uac.rb file being added because the cookbook generator previously added the other files.

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 excepts for one or simply manage everything before you commit.

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



```
> git commit -m "Initial Commit"
```

```
[master f530a8f] Initial Commit
 1 file changed, 13 insertions(+)
 create mode 100644 recipes/disable-uac.rb
```

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

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

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

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

Git Version Control

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

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



The screenshot shows a GitHub repository page for 'chef-training / chefdk-fundamentals-repo'. The 'apache' branch is selected. The commit history is displayed, showing the following commits:

File	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. GitHub is an example of a central git repository.



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 to store our disable-uac recipe.
- ✓ Add the "workstation" cookbook to version control.

The workstation cookbook now has its history being tracked in version control.



GL: Setting up a Workstation

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

- Create a cookbook
- Copy the disable-uac recipe within the cookbook
- Use the include_recipe method to insert disable-uac recipe
- Apply the default recipe to the workstation



chef-client

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

How would we apply the workstation's setup recipe?

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



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.



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

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

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

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

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

"recipe[COOKBOOK::RECIPE]"

COOKBOOK means the name of the Cookbook.

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

GL: Applying the workstation's disable-uac recipe



```
> chef-client --local-mode --runlist "recipe[workstation::disable-uac]"  
Starting Chef Infra Client, version 15.0.300  
resolving cookbooks for run list: ["workstation::disable-uac"]  
Synchronizing Cookbooks:  
  - workstation (0.1.0)  
  
Installing Cookbook Gems:  
Compiling Cookbooks...  
Converging 2 resources  
Recipe: workstation::disable-uac  
* registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]  
action create (up to date)  
* registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]  
action create (up to date)
```

Let's apply the workstation cookbook's recipe named 'disable-uac'.



GL: Setting up a Workstation

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

Objective:

- Create a cookbook
- Copy the disable-uac recipe within the cookbook
- Use the include_recipe method to insert disable-uac recipe
- Apply the default recipe to the workstation



include_recipe method

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

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

GL: The Default Recipe Includes the Disable Recipe

```
~\cookbooks\workstation\recipes\default.rb

#
# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2017 The Authors, All Rights Reserved.

include_recipe 'workstation::disable-uac'
```

We are interested in having the default recipe for our workstation cookbook run the contents of the 'disable-uac' 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.

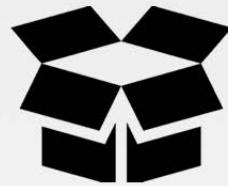


GL: Setting up a Workstation

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

Objective:

- ✓ Create a cookbook
- ✓ Copy the disable-uac recipe within the cookbook
- ✓ Use the include_recipe method to insert disable-uac recipe
- ✓ Apply the default recipe to the workstation



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

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

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

Actually, we didn't tell you everything about specifying the run list for the `chef-client` command.

When defining a recipe in the run list you may omit the name of the recipe and only use the cookbook name when that recipe's name is 'default'.

Similar to how resources have default actions and default properties, Chef uses the concept of providing sane defaults. A cookbook doesn't have to have a default recipe but most every cookbook has one. It's called default because when you think of a cookbook it is the recipe that defines the most common configuration policy.

GL: Apply the Cookbook's Default Recipe



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

```
...
```

```
resolving cookbooks for run list: ["workstation"]
```

```
Synchronizing Cookbooks:
```

```
  - workstation (0.1.0)
```

```
...
```

```
Converging 2 resources
```

```
Recipe: workstation::disable-uac
```

```
  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]  
action create (up to date)
```

```
  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]  
action create (up to date)
```

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 '`disable-uac`' recipe.



GL: Setting up a Workstation

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

Objective:

- ✓ Create a cookbook
- ✓ Copy the disable-uac recipe within the cookbook
- ✓ Use the include_recipe method to insert disable-uac recipe
- ✓ Apply the default recipe to the workstation

Instructor Note:

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

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



Lab: Setting up a Web Server

- Use `chef generate` to create a cookbook named "myiis".
- Create a recipe named "`server.rb`" with the following policy:

The `powershell_script` named 'Install IIS' is run with the code: '[Add-WindowsFeature Web-Server](#)'.

The `file` named '[C:\inetpub\wwwroot\Default.htm](#)' is created with the content:

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

The `service` named '[w3svc](#)' is started and enabled.

- Use the `include_recipe` method to include the server recipe from within the default recipe
- Apply the default recipe and verify with `Invoke-WebRequest localhost`

Now. Here is your last challenge: Deploying a Web Server with Chef.

We need a cookbook named iis that has a server recipe. Within that server recipe we need to use a `powershell_script` to install the IIS windows feature, write out an example HTML file, and then start and enable the `w3svc` service.

Then we should apply that recipe and make sure the site is up and running by running a command to visit that site.

Instructor Note:

Allow 15 minutes to complete this exercise.

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

GL: Return to Your Home Directory



```
> cd ~  
PS C:\Users\Administrator>
```

Lab: Create a Cookbook



```
> chef generate cookbook cookbooks\myiis
```

```
Generating cookbook myiis
```

- Ensuring correct cookbook content
- Committing cookbook files to git

```
Your cookbook is ready. Type `cd cookbooks\myiis` to enter it.
```

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

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

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

```
test/integration/default/default_test.r
```

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

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



Lab: Setting up a Web Server

- ✓ Use `chef generate` to create a cookbook named "myiis".
- ❑ Create a recipe named "`server.rb`" with the following policy:

The `powershell_script` named 'Install IIS' is run with the code: '[Add-WindowsFeature Web-Server](#)'.

The `file` named '[C:\inetpub\wwwroot\Default.htm](#)' is created with the content:

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

The `service` named '[w3svc](#)' is started and enabled.

- ❑ Use the `include-recipe` method to include the server recipe from within the default recipe
- ❑ Apply the default recipe and verify with `Invoke-WebRequest localhost`

Lab: Create a Recipe



```
> chef generate recipe cookbooks\myiis server
...
      (diff output suppressed by config)
      * directory[cookbooks/myiis/test/integration/default] action create (up to
date)
      * template[cookbooks/myiis/test/integration/default/server_test.rb] action
create_if_missing
        - create new file cookbooks/myiis/test/integration/default/server_test.rb
        - update content in file
cookbooks/myiis/test/integration/default/server_test.rb from none to 6becfa
      (diff output suppressed by config)
      * template[cookbooks/myiis/recipes/server.rb] action create

        - create new file cookbooks/myiis/recipes/server.rb
        - update content in file cookbooks/myiis/recipes/server.rb from none to
b16ff8
...
...
```

From the Chef home directory, run the command 'chef generate recipe myiis server'. This will create a recipe called server.rb in the myiis cookbook.

Lab: Add Code to the Server Recipe

```
~\cookbooks\myiis\recipes\server.rb
```

```
powershell_script 'Install IIS' do
  code 'Add-WindowsFeature Web-Server'
end

file 'C:\inetpub\wwwroot\Default.htm' do
  content '<h1>Hello, world!</h1>'
end

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

The server recipe, found at ~cookbooks/myiis/recipes/server.rb, defines the policy:

Install the web server feature, create an example html file, and ensure the w3svc service is started and enabled

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: Setting up a Web Server

- ✓ Use `chef generate` to create a cookbook named "myiis".
- ✓ Create a recipe named "`server.rb`" with the following policy:

The `powershell_script` named 'Install IIS' is run with the code: '[Add-WindowsFeature Web-Server](#)'.

The `file` named '[C:\inetpub\wwwroot\Default.htm](#)' is created with the content:

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

The `service` named '[w3svc](#)' is started and enabled.

- Use the include-recipe method to include the server recipe from within the default recipe
- Apply the default recipe and verify with `Invoke-WebRequest localhost`

Lab: The Default Recipe Includes the Disable Recipe

```
~\cookbooks\myiis\recipes\default.rb

#
# Cookbook Name:: myiis
# Recipe:: default
#
# Copyright (c) 2017 The Authors, All Rights Reserved.

include_recipe 'myiis::server'
```

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

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



Lab: Setting up a Web Server

- ✓ Use `chef generate` to create a cookbook named "`myiis`".
- ✓ Create a recipe named "`server.rb`" with the following policy:
The `powershell_script` named 'Install IIS' is run with the code: '[Add-WindowsFeature Web-Server](#)'.
The `file` named '[C:\inetpub\wwwroot\Default.htm](#)' is created with the content:
`'<h1>Hello, world!</h1>'`
The `service` named '[w3svc](#)' is started and enabled.
- ✓ Use the include-recipe method to include the server recipe from within the default recipe
- Apply the default recipe and verify with `Invoke-WebRequest localhost`

Lab: Apply the Default Recipe



```
> chef-client -z --runlist "recipe[myiis]"
```

```
Synchronizing Cookbooks:
  - myiis (0.1.0)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 3 resources
Recipe: myiis::server
  * powershell_script[Install IIS] action run
    - execute "C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe" -NoLogo -
NonInteractive -NoProfile -ExecutionPolicy Bypass -InputFormat None -File
"C:/Users/ADMINI~1/AppData/Local/Temp/2/chef-script20200627-5836-9ilclx.ps1"

  * file[C:\inetpub\wwwroot\Default.htm] action create
    - create new file C:\inetpub\wwwroot\Default.htm
    - update content in file C:\inetpub\wwwroot\Default.htm from none to 17d291
      --- C:\inetpub\wwwroot\Default.htm 2020-06-27 18:01:15.543906000 +0000
      +++ C:\inetpub\wwwroot\chef-Default20200627-5836-f1t5x1.htm 2020-06-27
18:01:15.543906000 +0000
...
...
```

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

Lab: Verify That the Website is Available



> **Invoke-WebRequest localhost**

```
StatusCode      : 200
StatusDescription : OK
Content          : <h1>Hello, world!</h1>
RawContent       : HTTP/1.1 200 OK
                    Accept-Ranges: bytes
                    Content-Length: 22
                    Content-Type: text/html
                    Date: Thu, 27 Jun 2020 18:03:25 GMT
                    ETag: "d4653250122dd51:0"
                    Last-Modified: Thu, 27 Jun 2020 18:01:15 GMT
                    Server...
Forms           : {}
Headers         : {[Accept-Ranges, bytes], [Content-Length, 22], [Content-Type, text/html], [Date, Thu, 27 Jun 2020 18:03:25]}
...
```

...So verify that the website is available and returns the content we expect to see.



Lab: Setting up a Web Server

- ✓ Use `chef generate` to create a cookbook named "myiis".
- ✓ Create a recipe named "`server.rb`" with the following policy:
The `powershell_script` named 'Install IIS' is run with the code: '[Add-WindowsFeature Web-Server](#)'.
The `file` named '[C:\inetpub\wwwroot\Default.htm](#)' is created with the content:
`'<h1>Hello, world!</h1>'`
The `service` named '[w3svc](#)' is started and enabled.
- ✓ Use the include-recipe method to include the server recipe from within the default recipe
- ✓ Apply the default recipe and verify with `Invoke-WebRequest localhost`

Instructor Note:

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



GL: Commit Your Work

```
> cd ~\cookbooks\myiis  
> git init  
> git add .  
> git status  
> git commit -m "Initial Commit"
```

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

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



Committing Your Work to a Version Control Repo

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

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

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

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



Committing Your Work to Version Control

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

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

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

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

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



Linting with cookstyle

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



Linting with cookstyle

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

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

Example: Running cookstyle in workstation Cookbook



```
$ cookstyle
```

```
Inspecting 9 files
...C.....
Offenses:

recipes/server.rb:15:1: C: 1 trailing blank lines detected.
recipes/server.rb:15:1: C: Trailing whitespace detected.

9 files inspected, 2 offenses detected
```

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

Notice that the line number where the offense exists is indicated. This command was run from ~/cookbooks/apache

Example: Running cookstyle in a Cookbook



```
$ 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.

Example: `cookstyle -a`



```
$ cookstyle -a
```

```
Inspecting 2 files
CC
Offenses:
default.rb:1:1: C: Layout/EndOfLine: Carriage return character detected.
disable-uac.rb:1:1: C: Layout/EndOfLine: Carriage return character detected.
...
2 files inspected, 10 offenses detected, 8 offenses corrected
```

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



Group Lab: cookstyle

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

GL: Run cookstyle at the Cookbook Level



```
> cd ~\cookbooks\workstation  
> cookstyle
```

```
Inspecting 7 files  
CCCC.CC  
Offenses:  
metadata.rb:1:1: C: Layout/EndOfLine: Carriag  
name 'workstation' ...  
^^^^^^^^^^^^  
Policyfile.rb:1:1: C: Layout/EndOfLine: Carriage return character detected.  
# Policyfile.rb - Describe how you want Chef Infra Client to build your system. ...  
....  
7 files inspected, 14 offenses detected recipes/default.rb:1:1: C: Layout/EndOfLine: Carriage  
return character detected.
```

In this example, 7 files were inspected. Depending on how you typed your code into the cookbook's files, you should see quite a few non-critical syntax violations.

GL: Run cookstyle at the Recipes Level



```
> cd ~\cookbooks\workstation\recipes  
> cookstyle
```

```
disable-uac.rb:13:5: C: Style/HashSyntax: Use the new Ruby 1.9 hash syntax.  
  :name => 'ConsentPromptBehaviorAdmin',  
  ^^^^^^^^^  
...  
disable-uac.rb:15:5: C: Style/TrailingCommaInHashLiteral: Put a comma after the last item of  
a multiline hash.  
  :data => 0  
  ^^^^^^^^^^  
  
2 files inspected, 10 offenses detected
```

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

GL: Run cookstyle on an Individual File



```
> cookstyle default.rb
```

```
Inspecting 1 file
```

```
c
```

```
Offenses:
```

```
default.rb:1:1: C: Layout/EndOfLine: Carriage return character detected.  
# ...  
^  
1 file inspected, 1 offense detected
```

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



Review Questions

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

Answer these questions.

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



Q&A

What questions can we answer for you?

- Cookbooks
- Recipes
- Run-lists
- include_recipe method

What questions can we help you answer?

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



©2020 Chef Software Inc.

Ohai and the Node Object

Finding and Displaying Information About Our System

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

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



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.

Some Useful System Data

- platform
- 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 platform, hostname, memory, and CPU megahertz of our current system.

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.

Demo: Finding Platform Info



```
> Get-WMIObject Win32_OperatingSystem
```

```
SystemDirectory : C:\Windows\system32
Organization    : Amazon.com
BuildNumber     : 9600
RegisteredUser  : EC2
SerialNumber    : 00252-70000-00000-AA535
Version         : 6.3.9600
```

In this demo, we might run a command like this to discover platform information

Demo: Finding the Hostname



```
> $env:computername
```

```
WIN-KRQSVD3RFM7
```

Running the following command will allow us to find the system's hostname.

Demo: Finding the Total Memory



```
> wmic ComputerSystem get TotalPhysicalMemory
```

```
TotalPhysicalMemory  
8052654080
```

Running the following command will return to the total physical memory of the system.

Demo: Finding the CPU MHz



```
> wmic cpu get name
```

```
Name
```

```
Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz
```

Running the following command will return to the cpu name which includes the clock speed of the CPU.



Capturing System Data

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

How accurate will our recipe be if we hard code this information within our resources?

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 Default page 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.



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).



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 platform, hostname, total memory, and cpu megahertz.

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

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

```node object = (ohai data) + (cookbook attributes)```



## 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` automatically executes Ohai. This information is stored within a variable we call 'the node object'



## GL: Details About the Node

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

**Objective:**

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

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

Instructor Note:

The GitHub repo for the 0.2.0 version of the 'myiis' cookbook can be found at:  
<https://github.com/chef-training/devops-cookbooks-myiis.git>

# GL: Running Ohai!

> ohai

```
{
 "kernel": {
 "os_info": {
 "boot_device": "\Device\HarddiskVolume1",
 "build_number": "9600",
 "build_type": "Multiprocessor Free",
 "caption": "Microsoft Windows Server 2012 R2 Standard",
 "code_set": "1252",
 "country_code": "1",
 "creation_class_name": "Win32_OperatingSystem",
 "cs_creation_class_name": "Win32_ComputerSystem",
 "csd_version": null,
 "cs_name": "WIN-DCK5NTVLEBH",
 "description": "Windows Server 2012 R2 Standard Edition"
 }
 }
}
```

Ohai is also a command-line application that is part of the Chef Development Kit (ChefDK).

## GL: Running Ohai to Show the Platform



```
> ohai platform
```

```
[
 "windows"
]
```

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

## GL: Running Ohai to Show the Hostname



```
> ohai hostname
```

```
[
```

```
"WIN-NJ5007IAJNR"
```

```
]
```

## GL: Running Ohai to Show the Memory



```
> ohai memory
```

```
{

 "swap": {

 "total": "8388608kB",

 "free": "8388608kB"

 },

 "total": "8388208kB",

 "free": "6841060kB"

}
```

## GL: Running Ohai to Show the Total Memory



```
> ohai memory/total
```

```
[
```

```
"8388208kB"
```

```
]
```

In some cases you will find that there are nested node attributes, to access these child attributes you use this forward slash syntax.

## GL: Running Ohai to Show the CPU



```
> ohai cpu
```

```
{
 "0": {
 "cores": 2,
 "vendor_id": "GenuineIntel",
 "family": "1",
 "model": "16130",
 "stepping": "2",
 "physical_id": "CPU0",
 "model_name": "Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz",
 "description": "Intel64 Family 6 Model 63 Stepping 2",
 "mhz": "2400",
 "cache_size": " KB"
 },
 "total": 2,
 "cores": 2,
 "real": 1
}
```

## GL: Running Ohai to Show the First CPU



```
> ohai cpu/0
```

```
{
 "cores": 2,
 "vendor_id": "GenuineIntel",
 "family": "1",
 "model": "16130",
 "stepping": "2",
 "physical_id": "CPU0",
 "model_name": "Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz",
 "description": "Intel64 Family 6 Model 63 Stepping 2",
 "mhz": "2400",
 "cache_size": " KB"
}
```

## GL: Running Ohai to Show the First CPU MHz



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

```
[
 "2400"
]
```



## The Node Object

The node object is accessible within recipes as well as from the command line.

Let's take a look at the syntax.

<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.

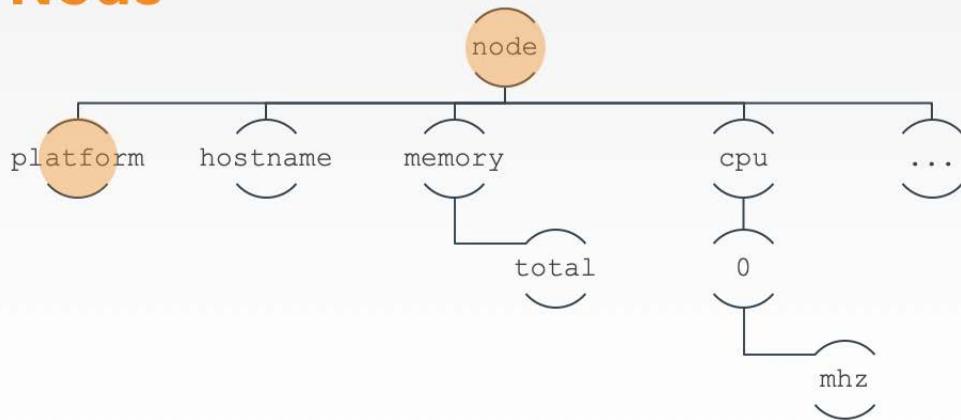
An attribute is a specific detail about a node, such as an IP address, a host name, a list of loaded kernel modules, the version(s) of available programming languages that are available, and so on.

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

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

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

## The Node



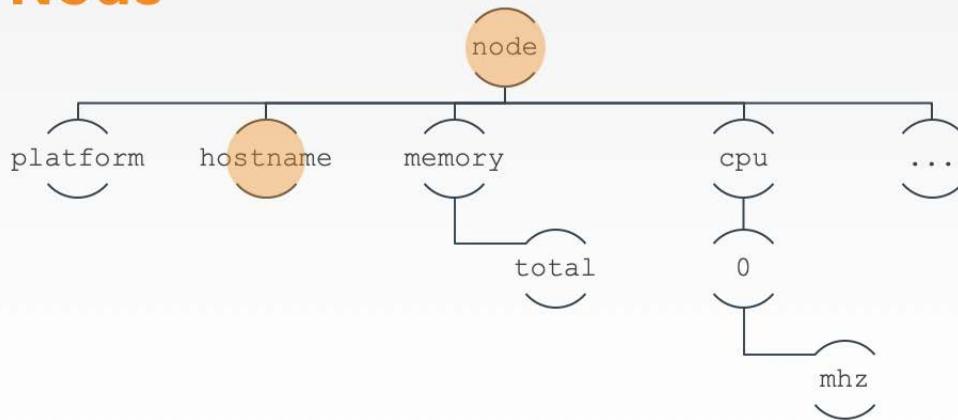
```
CLI: ohai platform
```

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

```
OUTPUT: windows
```

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

## The Node



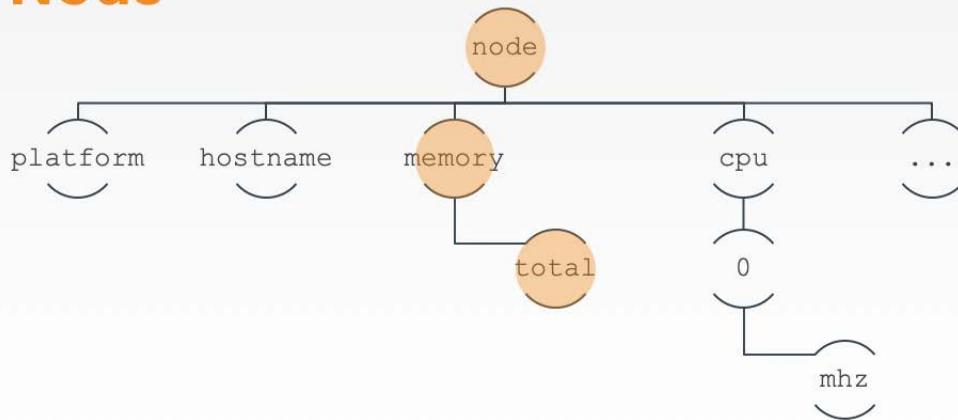
```
CLI: ohai hostname
```

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

```
OUTPUT: WIN-KRQSVD3RFM7
```

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

## The Node



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

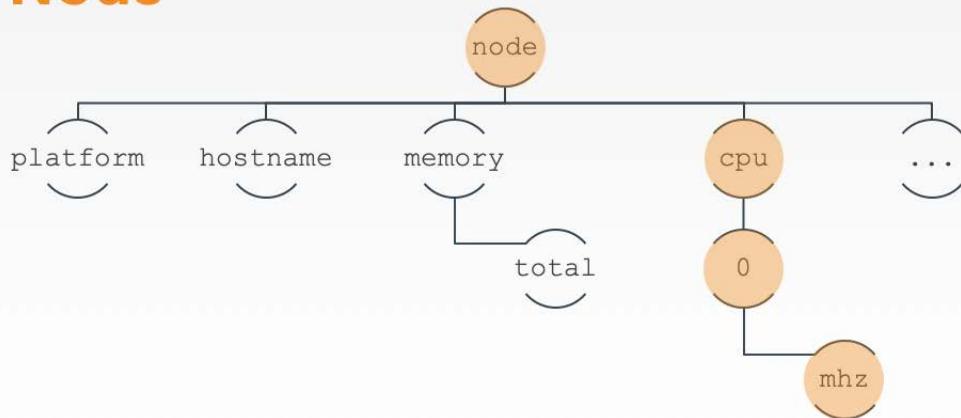
```
RECIPE: node['memory']['total']
```

```
OUTPUT: 7863920kB
```

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

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

## The Node



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

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

```
OUTPUT: 2900
```

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](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](http://en.wikipedia.org/wiki/String_interpolation#Ruby)

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



## String Interpolation

```
I have 4 apples
```

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



To escape out to display a ruby variable or ruby code you use the following sequence: number sign, left curly brace, the ruby variables or ruby code, and then a right curly brace.



## GL: Details About the Node

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

### **Objective:**

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

## GL: Details About the Node

~\cookbooks\myiis\recipes\server.rb

```
... POWERSHELL_SCRIPT RESOURCE ...

file 'c:\inetpub\wwwroot\Default.htm' do
 content "<h1>Hello, world!</h1>
<h2>PLATFORM: #{node['platform']}</h2>
<h2>HOSTNAME: #{node['hostname']}</h2>
<h2>MEMORY: #{node['memory']['total']}</h2>
<h2>CPU Mhz: #{node['cpu'][0]['mhz']}</h2>"
end

... SERVICE RESOURCE ...
```

Update the content attribute of the file resource to include the node details. Remember to include the details about the node in the content string we need to use string interpolation. String interpolation requires that the string be a double-quoted string. Ensure you change the single-quotes to double quotes. Then use `#{}` to escape out of the double-quoted string. Between the curly braces you can define the node object and specify which attribute you would like to display.



## GL: Details About the Node

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

### **Objective:**

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



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



## Semantic Versions

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

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

<http://semver.org>

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.



## Major, Minor, or Patch?

What kind of changes did you make to the cookbook?

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

Changing the contents of an existing resource--by adding the attributes of the node doesn't seem like a bug fix and it doesn't seem like a major rewrite. It is like a new set of features while remaining backwards compatible. That sounds like a Minor changes based on the definitions provided by the Semantic Versioning documentation.

## GL: Update the Cookbook Version

```
~\cookbooks\myiis\metadata.rb
```

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

Edit the "myiis" cookbook's metadata file. Find the line that specifies the version and increase the minor value by one.



## GL: Details About the Node

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

**Objective:**

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

## GL: Return Home and Apply myiis Cookbook



```
> cd ~
> chef-client --local-mode -r "recipe[myiis]"
```

```
...
Synchronizing Cookbooks:
- myiis (0.2.0)
...
* powershell_script[Install IIS] action run
 - execute "C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe" -NoLogo -NonInteractive -NoProfile -
ExecutionPolicy Bypass -Input

Format None -File "C:/Users/ADMINI~1/AppData/Local/Temp/2/chef-script20200205-4560-giw545.ps1"
* file[C:\inetpub\wwwroot\Default.htm] action create
 - update content in file C:\inetpub\wwwroot\Default.htm from 17d291 to afadfd
--- C:\inetpub\wwwroot\Default.htm 2020-02-05 18:15:08.678449100 +0000
+++ C:\inetpub\wwwroot/chef-Default20200205-4560-e2bb3h.htm 2020-02-05 18:29:46.178300200 +0000
@@ -1,2 +1,6 @@
 <h1>Hello, world!</h1>
+<h2>PLATFORM: windows</h2>
+<h2>HOSTNAME: WIN-NJ5007IAJNR</h2>
+<h2>MEMORY: 8283740kB</h2>
+<h2>CPU Mhz: 2500</h2>

Running handlers:...
```

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 'myiis' cookbook's default recipe.

## GL: Verify the Default Page Returns the Details



> `Invoke-WebRequest localhost`

```
StatusCode : 200
StatusDescription : OK
Content : <h1>Hello, world!</h1>
 <h2>PLATFORM: windows</h2>
 <h2>HOSTNAME: WIN-8694LT97S51</h2>
 <h2>MEMORY: 8388208kB</h2>
 <h2>CPU Mhz: 2400</h2>
RawContent : HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 137
...
```

Verify that the Default page is returning the new updated content with the details about the system.



## GL: Details About the Node

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

**Objective:**

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

**Instructor Note:**

The GitHub repo for the 0.2.0 version of the 'myiis' cookbook can be found at:  
<https://github.com/chef-training/devops-cookbooks-myiis.git>



## Review

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

Answer these questions.

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



## Q&A

What questions can we help you answer?

- Ohai
- Node Object
- Node Attributes
- String Interpolation

With that we have added all of the requested features.

What questions can we help you answer?

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



©2020 Chef Software Inc.

## Testing Cookbooks – Prelude

Setting up our test environment in advance of this lesson

©2020 Chef Software Inc.



**Instructor Note:** It can take up to 15 mins to `kitchen converge` a Windows node. So have the class kick it off and let it run as you they take a break and cover some of the theory. The gotcha is they may need to run some the early commands in a new PowerShell window.

## Before We Start this Lesson



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

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

Just follow along

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

## GL: Replace the Kitchen Config



```
> cp ~\kitchen-template.yml ~\cookbooks\workstation\kitchen.yml
```

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

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

## GL: Add Your Name, Company & Cookbook

~\cookbooks\workstation\kitchen.yml

... CONTENT OMITTED FOR CLARITY...

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

Edit the file to insert your name and company information.

## GL: Add chef-client 15 Requirements

~\cookbooks\workstation\kitchen.yml

... CONTENT OMITTED FOR CLARITY...

```
provisioner:
 name: chef_zero
 cookbook_path: C:\Users\Administrator\cookbooks
client_rb:
 chef_license: accept
 product_name: chef
 product_version: 15
```

Add these lines below line 20.

provisioner:

```
 name: chef_zero
 cookbook_path: C:\Users\Administrator\cookbooks
client_rb:
 chef_license: accept
 product_name: chef
 product_version: 15
```

## GL: Add Your Name, Company & Cookbook

~\cookbooks\workstation\kitchen.yml

```
... CONTENT OMITTED FOR CLARITY...

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

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

Edit the file to insert your name and company information.

## GL: Create the TK Instance



```
> cd ~\cookbooks\workstation
> kitchen create
-----> Creating <default-windows-2012r2>...
Detected platform: windows version 2012rtm on x86_64. Instance Type:
m3.large. Default username: administrator (default).
...
Waited 160/600s for instance <i-03e36657d5fd9d8b0> to become ready.
Waited 165/600s for instance <i-03e36657d5fd9d8b0> to become ready.
Waited 170/600s for instance <i-03e36657d5fd9d8b0> to become ready.
Waited 175/600s for instance <i-03e36657d5fd9d8b0> to become ready.
EC2 instance <i-03e36657d5fd9d8b0> ready (hostname: ec2-100-25-15-
150.compute-1.amazonaws.com).
[WinRM] Established
Finished creating <default-windows-2012r2> (3m18.83s).
```

Execute `kitchen converge` to validate that our workstation cookbook's default recipe is able to converge on the Windows 2012 R2 instance.

Instructor Note: This process can easily take 5 minutes before the instance is ready and able to be converged. You might want to take a break at this point.

# Testing Cookbooks

Validating Our Recipes in Virtual Environments

©2020 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
- Refer to the InSpec documentation
- 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.



## We Should Test Cookbooks

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

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

Will the recipes that we created work on another system similar to this one? Will they work in production?

When we develop our automation we need to start thinking about verifying it. Because it is all too common a story of automation failing when it reaches production because it was never validated against anything other than "my machine".

So how could we solve a problem like this?

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

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

# Steps to Verify Cookbooks



Create Virtual Machine

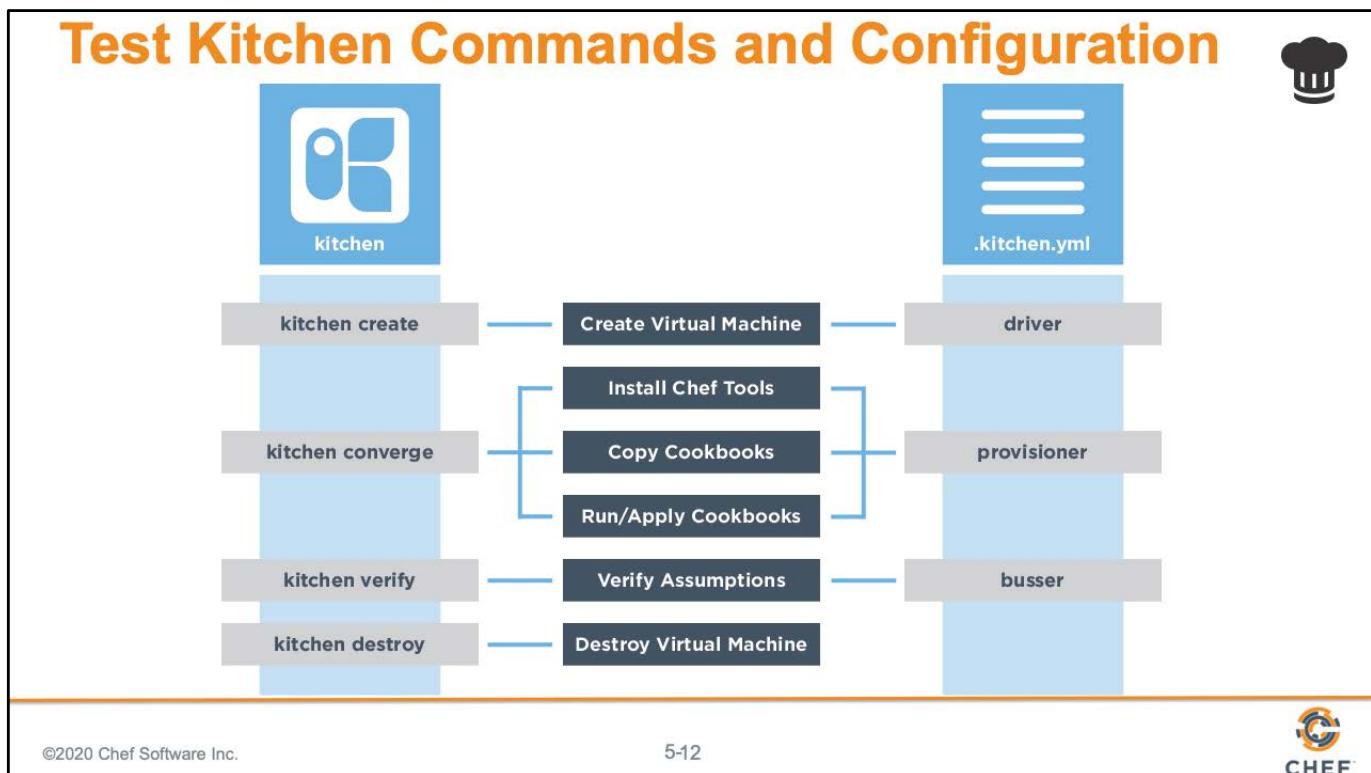
Install Chef Tools

Copy Cookbooks

Run/Apply Cookbooks

Verify Assumptions

Destroy Virtual Machine



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

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

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

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

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.



## 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 a Windows 2012R2 platform
- Apply the "workstation" cookbook's default recipe to that virtual machine

Well if Chef is to replace our existing tools, it is going to need to provide a way to make testing the policies more delightful.



## kitchen.yml

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

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

<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 lets learn more about the various fields in the configuration file.

## Demo: The kitchen Driver



~\cookbooks\workstation\kitchen.yml

```

driver:
 name: vagrant

provisioner:
 name: chef_zero

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

KITCHEN CONFIGURATION CONTINUES...
```

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

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

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

The driver is responsible for creating the instance that we will use to test our cookbook. There are lots of different drivers available.

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

## Demo: The kitchen Provisioner



~\cookbooks\workstation\kitchen.yml

```

driver:
 name: vagrant

provisioner:
 name: chef_zero

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

KITCHEN CONFIGURATION CONTINUES...
```

This tells Test Kitchen how to run Chef, to apply the code in our cookbook to the machine under test.

The default and simplest approach is to use `chef_zero`.

The second key is provisioner, which also has a single key-value pair which is the name of the provisioner Kitchen will use when executed. This provisioner is responsible for how it applies code to the instance that the driver created. Here the default value is `chef_zero`.

## Demo: The kitchen Platforms



~\cookbooks\workstation\kitchen.yml

```
TOP PART OF KITCHEN CONFIGURATION

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

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

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

The third key is platforms, which contains a list of all the platforms that Kitchen will test against when executed. This should be a list of all the platforms that you want your cookbook to support.

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

## Demo: The kitchen Suites

~\cookbooks\workstation\kitchen.yml

```
TOP PART OF KITCHEN CONFIGURATION

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

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

This section defines what we want to test. It includes the Chef run-list of recipes that we want to test.

We define a single suite named "default".

The fourth key is suites, which contains a list of all the test suites that Kitchen will test against when executed. Each suite usually defines a unique combination of run lists that exercise all the recipes within a cookbook.

In this example, this suite is named 'default'.

## Demo: The kitchen Suites

~\cookbooks\workstation\kitchen.yml

```
TOP PART OF KITCHEN CONFIGURATION

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

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

The suite named "default" defines a run\_list.

Run the "workstation" cookbook's "default" recipe file.

This default suite will execute the run list containing the workstation cookbook's default recipe.



## Kitchen Test Matrix

Kitchen defines a list of instances, or test matrix, based on the platforms multiplied by the suites.

PLATFORMS x SUITES

Running `kitchen list` will show that matrix.

It is important to recognize that within the `.kitchen.yml` file we defined two fields that create a test matrix; The number of platforms we want to support multiplied by the number of test suites that we defined.

## Example: View the Kitchen Test Matrix

```
> kitchen list
```

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

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

In this example, if we were to run this command on a local workstation, we can visualize this test matrix by running the command `kitchen list`.

In the output you can see that an instance is created in the list for every suite and every platform. In our current file we have one suite, named 'default', and two platforms. First the ubuntu 12.04 platform.

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

## Example: View the Kitchen Test Matrix

```
> kitchen list
```

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

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

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

And the second centos 6.5 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.

# PROBLEM

## Virtualization



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

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

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

We have provided a template file that has all the settings properly configured for our Training Amazon EC2 cloud. In the next series of steps we are going to copy that file into the cookbook and edit it with specific values.

## REVIEW ONLY: You Added Your Name and Company

~\cookbooks\workstation\.kitchen.yml

```

driver:
 name: ec2
 # ... OTHER DRIVER DETAILS ...
 instance_type: m3.large
 tags:
 # Replace YOURNAME and YOURCOMPANY here
 Name: "Chef Training Node for YOURNAME, YOURCOMPANY"
 created-by: "test-kitchen"
 user: <%= ENV['USER'] %>
... REMAINDER OF THE CONFIGURATION FILE ...
```

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

## Review: You added chef-client 15 Requirements

~\cookbooks\workstation\.kitchen.yml

... CONTENT OMITTED FOR CLARITY...

```
provisioner:
 name: chef_zero
 cookbook_path: C:\Users\Administrator\cookbooks
client_rb:
 chef_license: accept
 product_name: chef
 product_version: 15
```

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

**REVIEW: You Updated the kitchen.yml to Test the Workstation Cookbook**

```
~\cookbooks\workstation\.kitchen.yml
... TOP PART OF THE CONFIGURATION FILE ...

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

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

## GL: Ensure You Are in ~\cookbooks\workstation



```
> cd ~\cookbooks\workstation
```



Move into the workstation cookbook directory. Within that directory you are able to run the various kitchen commands.

## GL: Look at the Test Matrix



&gt; kitchen list

Instance	Driver	Provisioner	Verifier	Transport	Last Action	Last Error
default-windows-2012r2	Ec2	ChefZero	Inspec	Winrm	Created	<None>

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



## 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 a Windows 2012R2 platform
- ❑ Apply the "workstation" cookbook's default recipe to that virtual machine

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



## Kitchen Create

kitchen  
create

kitchen  
converge

kitchen  
verify

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

**Create one or more instances.**

The first kitchen command is `kitchen create`.

To create an instance means to turn on virtual or cloud instances for the platforms specified in the kitchen configuration.

In our case, this command would use the ec2 driver to create a Windows 2012 R2 instance.

**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 (v2.2.5)
----> Converging <default-windows-2012r2>...
 Preparing files for transfer
$$$$$$ You must set your run_list in your Policyfile instead of kitchen config. The run_list
in your config will be ignored.
$$$$$$ Ignored run_list: ["recipe[workstation::default]"]
 Policy lock file doesn't exist, running `chef install` for Policyfile
C:/Users/Administrator/cookbooks/workstation/Policyfile.rb...
 Building policy workstation
 Expanded run list: recipe[workstation::default]
 Caching Cookbooks...
 Installing workstation >= 0.0.0 from path
Building policy workstation
 Expanded run list: recipe[workstation::default]
 Caching Cookbooks...
```

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

Execute `kitchen converge` to validate that our workstation cookbook's default recipe is able to converge on the Windows 2012 R2 instance.

Instructor Note: This process can easily take 5 minutes before the instance is ready and able to be converged.

Building policy workstation

Expanded run list: recipe[workstation::default]

Caching Cookbooks...

Installing workstation >= 0.0.0 from path

Lockfile written to C:/Users/Administrator/cookbooks/workstation/Policyfile.lock.json

Policy revision id: 4fec780f39cd37fe1800a0078f09604d0ca36943aa49c99d4cbcfdc9b6301e31

Preparing dna.json

Exporting cookbook dependencies from Policyfile  
C:/Users/ADMINI~1/AppData/Local/Temp/2/default-windows-2012r2-sandbox-20200628-8016-1grvnc...

Exported policy 'workstation' to C:/Users/ADMINI~1/AppData/Local/Temp/2/default-windows-2012r2-sandbox-20200628-8016-1grvnc



## 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 a Windows 2012R2 platform
- ✓ Apply the "workstation" cookbook's default recipe to that virtual machine

We successfully configured the workstation cookbook to use the Amazon EC2 driver and target a Windows 2012 R2 Instance.

# DISCUSSION



## Test Kitchen

What does this test when kitchen converges a recipe?

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

So what does this test when kitchen converges a recipe?

What does it NOT test when kitchen converges a recipe?

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

# DISCUSSION



## 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:

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

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 an expectation that will validate that UAC has been properly disabled on our test instance.

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



## InSpec

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

[https://docs.chef.io/inspec\\_reference.html](https://docs.chef.io/inspec_reference.html)

InSpec is one of many possible test frameworks that Test Kitchen supports. It is a popular choice for those doing Chef cookbook development because InSpec is written by Chef and is built on a Ruby testing framework named RSpec.

RSpec is similar to Chef - as it is a Domain Specific Language, or DSL, layered on top of Ruby. Where Chef gives us a DSL to describe the policy of our system, RSpec allows us to describe the expectations of tests that we define. InSpec adds a number of helpers to RSpec to make it easy to test the state of a system.

[https://docs.chef.io/inspec\\_reference.html](https://docs.chef.io/inspec_reference.html)

## Example: Is the Registry Key Set Correctly?

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

describe registry_key('System Policies', system_policies) do
 its('EnableLUA') { should eq 0 }
end
```

I expect the 'System Policies' found at registry key 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System' to have the property 'EnableLUA' with the value 0.

[https://docs.chef.io/inspec\\_reference.html#registry-key](https://docs.chef.io/inspec_reference.html#registry-key)

Here is an InSpec example.

We define a local variable named 'system\_policies' that will store the registry key that stores our particular key we want to verify. We use RSpec's 'describe' to begin to illustrate the element of our system that we are testing. InSpec provides a helper method named 'registry\_key' that allows to provide two parameters. The first parameter is a common name for the registry key. The second is the registry key itself which can be found inside the variable that we have defined.

Within the describe block we define an its block where we are targeting the particular key we are interested in examining. We want to ensure that the 'EnableLUA' key has a value that is equal to zero.



## Where do Tests Live?

```
workstation\test\integration\default\default_test.rb
```

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

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

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

## GL: Replace the Existing Test File With This

```
~\cookbooks\workstation\test\integration\default\default_test.rb

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

describe registry_key('System Policies', system_policies) do
 its('EnableLUA') { should eq 0 }
end
```

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

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

describe registry_key('System Policies', system_policies) do
 its('EnableLUA') { should eq 0 }
end
```



## The First Test

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

**Objective:**

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

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



## 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.

## GL: Ensure you are in the Cookbook

```
💻 > cd ~\cookbooks\workstation
```

Change into the workstation cookbook directory.

## GL: Running the Test



> kitchen verify

```
----> Starting Kitchen (v2.2.5)
----> Setting up <default-windows-2012r2>...
 Finished setting up <default-windows-2012r2> (0m0.00s).
----> Verifying <default-windows-2012r2>...
[2020-06-28T17:54:25+00:00] WARN: DEPRECATION: InSpec Attributes are being renamed to InSpec Inputs to
avoid confusion with Chef Attributes. Use --input-file on the command line instead of --attrs.
 Loaded tests from
{:path=>"C:/Users/Administrator.cookbooks.workstation.test.integration.default"}
Profile: tests from {:path=>"C:/Users/Administrator/cookbooks/workstation/test/integration/default"}
(tests from {:path=>"C:/Users/Administrator.cookbooks.workstation.test.integration.default"})
...
 Registry Key System Policies
 [PASS] EnableLUA should eq 0
Test Summary: 1 successful, 0 failures, 0 skipped
 Finished verifying <default-windows-2012r2> (0m4.98s).
----> Kitchen is finished. (0m13.04s)
```

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



## The First Test

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

### Objective:

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

Congratulations you wrote and executed your first test.



## Lab: Additional Test

- Add tests that validate that the registry key 'ConsentPromptBehaviorAdmin' has been set to the value to 0. HINT: You can have multiple 'its' lines.
  
- Run kitchen verify to validate the test meets the expectations that you defined

As a lab exercise, we want you to define an additional test to verify that the remaining registry key has been set properly.

Instructor Note: Allow 8 minutes to complete this exercise.

## Lab: Add This to the Existing Test File

```
~\cookbooks\workstation\test\integration\default\default_test.rb

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

describe registry_key('System Policies', system_policies) do
 its('EnableLUA') { should eq 0 }
 its('ConsentPromptBehaviorAdmin') { should eq 0 }
end
```

## Lab: Running the Test



> kitchen verify

```
-----> Starting Kitchen (v2.2.5)
-----> Verifying <default-windows-2012r2>...
...
 Loaded tests from
{:path=>"C: .Users Administrator.cookbooks.workstation.test.integration.default"}
...
Target: winrm://Administrator@http://ec2-18-235-249-206.compute-
1.amazonaws.com:5985/wsman:3389
 Registry Key System Policies

 [PASS] EnableLUA should eq 0
 [PASS] ConsentPromptBehaviorAdmin should eq 0
Test Summary: 2 successful, 0 failures, 0 skipped
 Finished verifying <default-windows-2012r2> (0m4.17s).
```

Now verify our expectation using the `kitchen verify` command. If you have defined the test correctly and the system is in the desired state you should see a summary of the execution. Showing 2 examples completed with 0 failures.



## Lab: Additional Test

- ✓ Add tests that validate that the registry key 'ConsentPromptBehaviorAdmin' has been set to the value to 0.
- ✓ Run kitchen verify to validate the test meets the expectations that you defined

Great. Now there are tests to validate both of the User Access Controls that are set by the disable-uac recipe.



## Kitchen Destroy



**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  
destroy

kitchen  
create

kitchen  
converge

kitchen  
verify

kitchen  
destroy

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

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

There is a single command that encapsulates the entire workflow - that is 'kitchen test'.

Kitchen test ensures that if the instance was in any state - created, converged, or verified - that it is 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.



## Lab: kitchen destroy

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

Now you are able to deploy the default website and verify that the website is running successfully.

## Lab: Run `kitchen destroy` to Clean Everything Up



> kitchen destroy

```
-----> Starting Kitchen (v2.2.5)
-----> Destroying <default-windows-2012r2>...
 EC2 instance <i-042b15bbb360a9a59> destroyed.
 Finished destroying <default-windows-2012r2> (0m1.36s) .
-----> Kitchen is finished. (0m9.44s)
```

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

# DISCUSSION



## Chef Intermediate

You can learn more about using Test Kitchen in the **Chef Intermediate** class.

There are also some Learn Chef Rally modules that cover Test Kitchen that you can take at any time.

<https://learn.chef.io/#/>

<https://learn.chef.io/#/>

# DISCUSSION



## Review Questions

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

Answer these questions.

1. We run kitchen commands from the cookbook directory because Test Kitchen is built as a unit tester, and we define units roughly as cookbooks.
2. Additional platforms for Test Kitchen are defined in the Platforms section of the .kitchen.yml
3. You might define a new test suite if you had a single cookbook but multiple recipes that helped accomplish the overall goal. For instance, in the chef-client community cookbook, there's several different methods you could choose from to configure Chef Client; if you're just testing on Windows, you wouldn't care about any of the other recipes in that cookbook other than "task.rb" and the default.

# 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.



**CHEF**<sup>TM</sup>

©2020 Chef Software Inc.

test\integration\default\server\_test.rb

# Templates

Desired State vs Data

# 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

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



## Cleaner Recipes

Previously, we updated our ‘myiis’ web server cookbook to display information about our node.

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

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

## Demo: The 'myiis' Server Recipe

~/cookbooks/myiis/recipes/server.rb

```
powershell_script 'Install IIS' do
 code 'Add-WindowsFeature Web-Server'
end

file 'c:\inetpub\wwwroot\Default.htm' do
 content "<h1>Hello, world!</h1>
<h2>PLATFORM: #{node['platform']}</h2>
<h2>HOSTNAME: #{node['hostname']}</h2>
<h2>MEMORY: #{node['memory']['total']}</h2>
<h2>CPU Mhz: #{node['cpu'][0]['mhz']}</h2>""
end

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

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

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

What if new changes are given to us for the website splash page?

For each new addition we would need to return to this recipe and carefully paste the contents of the new HTML into the string value of the content attribute.

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



## Manual Editing

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

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

This sounds like a bug waiting to happen.

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



## 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.



## Template

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

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

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

## EXAMPLE: Template File's Source Matches Up

```
> tree /f cookbooks\myiis\templates\default
Folder PATH listing
Volume serial number is B04A-119C
C:\USERS\ADMINISTRATOR\COOKBOOKS\MYIIS\TEMPLATES
 Default.htm.erb
No subfolders exist
```

```
template 'C:\inetpub\wwwroot\Default.htm' do
 source 'Default.htm.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](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](https://docs.chef.io/resource_template.html#using-templates)



## GL: Cleaner Recipes

*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 'myiis' cookbook
- Use chef-client to apply the "myiis" cookbook's "default" recipe
- Update the "myiis" cookbook's version for this patch

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.

### Instructor Note:

The GitHub repo for the 0.2.1 version of the 'myiis' cookbook can be found at:  
<https://github.com/chef-training/devops-cookbooks-myiis.git>

## 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

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, mit, gplv2, gplv3 - defaults
 to
 -s, --source SOURCE_FILE Copy content from SOURCE_FILE
 -g GENERATOR_COOKBOOK_PATH,
 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\myiis Default.htm

Recipe: code_generator::template
* directory[.\cookbooks/myiis/templates/default] action create
 - create new directory .\cookbooks/myiis/templates/default
* template[.\cookbooks/myiis/templates/Default.htm.erb] action create
 - create new file .\cookbooks/myiis/templates/Default.htm.erb
 - update content in file .\cookbooks/myiis/templates/Default.htm.erb from
none to e3b0c4
 (diff output suppressed by config)
```

Use '**chef generate template**' to create a template in the myiis cookbook found in the cookbooks\myiis directory and the file we want to create is named Default.htm.

## GL: Lets Look at the Template File



```
> tree /f cookbooks\myiis\templates
```

```
Folder PATH listing
```

```
Volume serial number is E04D-394D
```

```
C:\USERS\ADMINISTRATOR\COOKBOOKS\MYIIS\TEMPLATES
```

```
 Default.htm.erb
```

```
No subfolders exist
```

That is the first step. Now that the template exists, we are ready to define the content within the template file.



## GL: Cleaner Recipes

*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 'myiis' cookbook
- Use chef-client to apply the "myiis" cookbook's "default" recipe
- Update the "myiis" cookbook's version for this patch



## 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.

## Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

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

Here is an example of a text file that has several ERB tags defined in it.

## Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
50 + 50 = <%= 50 + 50 %>
<% else %>
At some point all of MATH I learned in school changed.
<% end %>
```

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

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

## Text Within an ERB Template

```
<% if (50 + 50) == 100 %>
```

```
50 + 50 = <%= 50 + 50 %>
```

```
<% else %>
```

~~At some point all of MATH I learned in school changed.~~

```
<% end %>
```

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

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.



## 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 server.rb Source to the Template

```
~\cookbooks\myiis\templates\Default.htm.erb
```

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

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\myiis\templates\Default.htm.erb
```

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

We are going to need to change string interpolation sequence with the ERB template syntax. And it seems for this content we want to display the output so we want to make sure that we are using ERB's angry squid opening tag.

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



## GL: Cleaner Recipes

*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 'myiis' cookbook
- ❑ Use chef-client to apply the "myiis" cookbook's "default" recipe
- ❑ Update the "myiis" cookbook's version for this patch

## GL: Remove the Existing Content Attribute from server.rb

```
~\cookbooks\myiis\recipes\server.rb

... POWERSHELL_SCRIPT RESOURCE ...

file 'c:\inetpub\wwwroot\Default.htm' do
 content "<h1>Hello, world!</h1>
<h2>ipaddress: #{node['ipaddress']}</h2>
<h2>hostname: #{node['hostname']}</h2>
<h2>total memory: #{node['memory']['total']}</h2>
<h2>CPU Mhz: #{node['cpu'][0]['mhz']}</h2>""
end
... SERVICE RESOURCE ...
```

Let's open the myiis 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 the File Resource to a Template

```
~\cookbooks\myiis\recipes\server.rb

... POWERSHELL_SCRIPT RESOURCE ...

template 'c:\inetpub\wwwroot\Default.htm' do
end

... SERVICE RESOURCE ...
```

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.

## GL: Change the File Resource to a Template

```
~\cookbooks\myiis\recipes\server.rb
```

```
template 'c:\inetpub\wwwroot\Default.htm' do
end
```

Now we have the path to our template so we can update the template resource's source attribute value.

## GL: Completed server.rb Code

```
~\cookbooks\myiis\recipes\server.rb

powershell_script 'Install IIS' do
 code 'Add-WindowsFeature Web-Server'
end

template 'c:\inetpub\wwwroot\Default.htm' do
 source 'Default.htm.erb'
end

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

The code should look like this:

```
powershell_script 'Install IIS' do
 code 'Add-WindowsFeature Web-Server'
end

template 'c:\inetpub\wwwroot\Default.htm' do
 source 'Default.htm.erb'
end

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



## GL: Cleaner Recipes

*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 'myiis' cookbook
- ❑ Use chef-client to apply the "myiis" cookbook's "default" recipe
- ❑ Update the "myiis" cookbook's version for this patch

## GL: Change Directories and Apply the Cookbook



```
> cd ~
> chef-client --local-mode -r "recipe[myiis]"

Starting Chef Infra Client, version 15.0.300
resolving cookbooks for run list: ["myiis"]
Synchronizing Cookbooks:
 - myiis (0.2.0)...
 template[c:\inetpub\wwwroot\Default.htm] action create

 - update content in file c:\inetpub\wwwroot\Default.htm from ffa7b2 to d7b236

 ...
Running handlers:

Running handlers complete

Chef Client finished, 2/4 resources updated in 06 seconds
```

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



## GL: Cleaner Recipes

*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 'myiis' cookbook
- ✓ Use chef-client to apply the "myiis" cookbook's "default" recipe
- ❑ Update the "myiis" cookbook's version for this patch

## GL: Update the Cookbook's Patch Number

```
~\cookbooks\myiis\metadata.rb
```

```
name 'myiis'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'all_rights'
description 'Installs/Configures myiis'
long_description 'Installs/Configures myiis'
version '0.2.1'
```

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

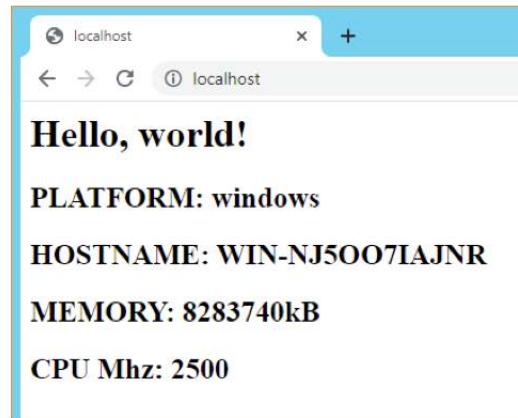
## GL: Test the Cookbook's Output



Point a web browser to `localhost` to ensure your cookbook works as expected.

Or you can type this at the command line:

```
Invoke-WebRequest localhost
```





## GL: Cleaner Recipes

*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 'myiis' cookbook
- ✓ Use chef-client to apply the "myiis" cookbook's "default" recipe
- ✓ Update the "myiis" cookbook's version for this patch

### Instructor Note:

The GitHub repo for the 0.2.1 version of the 'myiis' cookbook can be found at:  
<https://github.com/chef-training/devops-cookbooks-myiis.git>



## Summary

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

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



## Summary

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

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



## Review Questions

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

Answer these questions.

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



## Q&A

What questions can we help you answer?

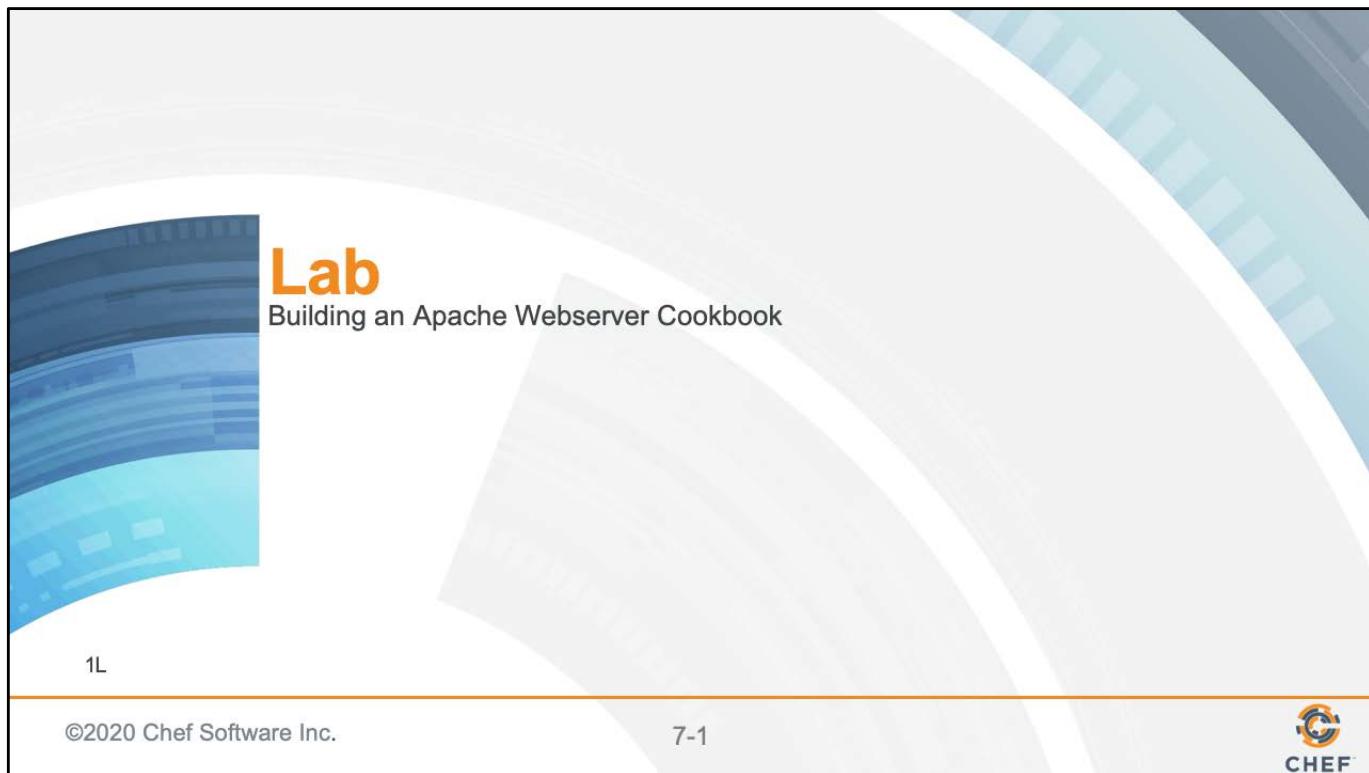
- Resources
- Templates
- ERB

What questions can we help you answer?

Generally or specifically about resources, templates, and ERB.



©2020 Chef Software Inc.



**Lab**  
Building an Apache Webserver Cookbook

1L

©2020 Chef Software Inc. 7-1

 CHEF

This module is going to present a challenge to you to exercise all the things that you have learned over the last few modules.

Instructor Note: The "1L" means you will need 1 Linux node for each student in this module.

Give the students 30 minutes to finish this module. At about 15 minutes in, the instructor can start doing the lab and let anyone who may be stuck to silently follow along. And if they're still stuck after that, not to worry because we'll all go through the slides together after the time is up.



## Lab: Pre-built Linux Node

*We will provide for you a Linux node with all the tools installed. You can do this lab on your own.*

### Objective:

- Login to the Linux Node

## GL: Login to the Linux Node



> **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 ~]$
```

You should ssh into the Linux node from your own laptop  
(much simpler) but if you're having trouble, the instructor might  
ask you to connect from the Windows workstation that you  
have already been using in the course.

We will provide you with the address, username and password of the Linux node. 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 2 'apache\_web' virtual machines at this time. The login credentials and password for the virtual workstations is chef/Cod3Can!



## Choose an Editor

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

*In your 'participant guide', found below, you can find tips for using these editors:*

**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!
```



## Lab: Pre-built Linux Node

*We will provide for you a Linux node with all the tools installed.*

### Objective:

- ✓ Login to the Linux Node



## Lab: Setting up an Apache Web Server

- Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
- Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
- Verify the site is available by running `curl localhost`

Your challenge is to deploy the Apache webserver. To do that you will need to create a cookbook, create a recipe, define the following policy within that recipe, create a template with specific content, apply the policy defined in the recipe, and verify that the policy applied successfully.

The lab's activity here is to challenge you to employ all that you have done already but in a new context. When everyone has completed the activity we will review the commands and code in the following slides.

Note: Examples of the resources you need are listed in the Resources module.

Instructor Note:

Allow 30 minutes to complete this exercise.

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

## Lab: From Your Linux Node - Working within Home Directory



```
$ cd ~
```

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

## Lab: 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.

## Lab: Accept the Chef License



```
$ sudo chef env
```

```
+-----+
 Chef License Acceptance

Before you can continue, 1 product license
must be accepted. View the license at
https://www.chef.io/end-user-license-agreement/

License that need accepting:
 * Chef Workstation

Do you accept the 1 product license (yes/no)?

> yes

Persisting 1 product license...
✓ 1 product license persisted.
```

Here we are pre-emptively accepting all Chef licenses.

Accepting Chef licenses is mandatory in all environments from workstation, to chef-client, Chef Habitat, etc.

## Lab: Creating the apache Cookbook



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

```
Generating cookbook apache
```

- Ensuring correct cookbook content
- Committing cookbook files to git

```
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 of local testing commands.
```

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

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

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



## Lab: Setting up an Apache Web Server

- ✓ Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- ❑ Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- ❑ Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- ❑ Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
- ❑ Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
- ❑ Verify the site is available by running `curl localhost`

# Lab: Creating the server Recipe



```
> chef generate recipe cookbooks/apache server

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
e5ca2c
```

Generate a 'server' recipe

## Lab: Defining the Policy in the server Recipe

```
~/cookbooks/apache/recipes/server.rb
```

```

Cookbook Name:: apache
Recipe:: server

Copyright (c) 2017 The Authors, All Rights Reserved.
package 'httpd'

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

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

Note: You can use vi/vim, nano, or emacs to edit these files on Linux.

Here within 'server.rb' we define the three resources that will configure our node as an Apache web server.

```
package 'httpd'

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

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



## Lab: Setting up an Apache Web Server

- ✓ Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- ✓ Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- ❑ Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- ❑ Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
- ❑ Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
- ❑ Verify the site is available by running `curl localhost`

## Lab: Creating the html Template



```
> 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)
```

Generate the index.html.erb template file where we will define the html content for our Default page.

# Lab: Defining the index.html Template

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

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

We again want to display the node's platform, hostname, total memory, and cpu speed within the page.

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



## Lab: Setting up an Apache Web Server

- ✓ Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- ✓ Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- ✓ Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- ❑ Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
- ❑ Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
- ❑ Verify the site is available by running `curl localhost`

## Lab: Including the server Recipe

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

```

Cookbook Name:: apache
Recipe:: default

Copyright (c) 2016 The Authors, All Rights Reserved.

include_recipe 'apache::server'
```

We include the server recipe with the default.



## Lab: Setting up an Apache Web Server

- ✓ Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- ✓ Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- ✓ Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- ✓ Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
  - Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
  - Verify the site is available by running `curl localhost`

## Lab: Applying the apache Cookbook's default Recipe



```
> sudo chef-client --local-mode --runlist "recipe[apache]"
Starting Chef Infra Client, version 15.0.300
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
 - apache (0.1.0)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 3 resources
...
* service[httpd] action enable
 - enable service service[httpd]
* service[httpd] action start
 - start service service[httpd]

Running handlers:
Running handlers complete
Chef Infra Client finished, 4/4 resources updated in 13 seconds
```

Later in this course you'll learn how to set run lists using Policyfiles.

When we apply the default recipe of the apache cookbook, we must do so with sudo privileges because we will be installing packages on a Linux system.



## Lab: Setting up an Apache Web Server

- ✓ Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- ✓ Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- ✓ Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- ✓ Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
- ✓ Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
- Verify the site is available by running `curl localhost`

## Lab: Verifying the Default Website is Available



```
> curl localhost
```

```
<html>
 <body>
 <h1>Hello, world!</h1>
 <h2>PLATFORM: centos</h2>
 <h2>HOSTNAME: ip-172-31-62-68</h2>
 <h2>MEMORY: 1013192kB</h2>
 <h2>CPU Mhz: 2400.234</h2>
 </body>
</html>
```

As you might imagine, you could also point a web browser to the external IP address of your Linux workstation/node to see similar output.

Finally, we verify the page is being hosted by running 'curl localhost'.



## Lab: Setting up an Apache Web Server

- ✓ Create a '`cookbooks`' directory and generate a cookbook named '`apache`'
- ✓ Create a '`server`' recipe that defines the following policy:
  - The `package` named '`httpd`' is installed.
  - The `template` named '`/var/www/html/index.html`' is created with the source '`index.html.erb`'
  - The `service` named '`httpd`' is both started and enabled.
- ✓ Create a template named '`index.html.erb`' and populate it with a welcome message, the node's platform, hostname, total memory, and CPU speed.
- ✓ Use the `include_recipe` method to include the '`server`' recipe within the '`default`' recipe
- ✓ Use '`sudo chef-client`' to locally apply the apache cookbook's default recipe
- ✓ Verify the site is available by running `curl localhost`

### Instructor Note:

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



## Q&A

What questions can we help you answer?

What questions can we help you answer?



©2020 Chef Software Inc.

# Workstation Installation

Configuring Your Laptop as a Workstation

# Objectives

After completing this module, you should be able to

- Install Chef Workstation on your laptop
- Execute commands to ensure everything is installed
- Install a text editor

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 Chef Workstation

*Installing the tools on your system*

## **Objective:**

- Install Chef Workstation
- Execute commands to ensure everything is installed
- 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 Chef Workstation.

After the installation is complete, we will verify that the various tools are working.

After that you can optionally download a number of other tools that will help you in your journey using Chef.

Let's get started.



## CHEF Workstation

Chef Workstation provides everything you need to get started with Chef.

Chef Workstation includes ChefDK tools plus additional ad-hoc configuration management tools.

Chef Workstation comes with the new **chef-run** utility, which can be used to execute chef code on any remote system accessible via SSH or WinRM. This provides a quick way to apply config changes to the systems you manage whether or not they're being actively managed by Chef

<https://www.chef.sh/about/chef-workstation/>

<https://www.chef.sh/about/chef-workstation/>

Throughout this course we have been using a number of tools found within the Chef Workstation. Chef Workstation contains tools like 'chef' and 'chef-client'. It also has a number of other great tools that we will use when connecting to a Chef Server, managing cookbook dependencies, or ensuring the quality of the cookbooks that we write.

Instructor Note: Prior to attending this course they may have received correspondence that informed them to setup Chef Workstation on their systems. It is possible that they did not and this slide acts as a good reminder to ensure that they have the necessary tools before continuing on to the next section.



## CHEF Workstation

**Chef Workstation** is a tool chain built on top of the Ruby programming language.

The Chef Workstation installer does not install any particular graphical-user-interface—it installs a CLI instead.

Chef Workstation is a tool chain built on top of the Ruby programming language. To assist with making the tools more portable to all platforms we package Ruby and all these tools together in a single platform specific installation package.

The installer does not install any particular graphical user interface, GUI, but instead installs the command-line tools we have been using thus far.



## GL: Install Chef Workstation

In this course, we won't be using the ad-hoc commands (`chef-run`) that Chef Workstation provides. But we will be using the commands and tools that ChefDK has historically provided.



## GL: Install Chef Workstation

Click the link below the slide in your participant guide or simply type the URL shown on the slide into a browser.

You'll need to select the correct download for your laptop's system.

This course has been tested on **Chef Workstation version: 0.15.18** so please install that for your system type.

<https://downloads.chef.io/chef-workstation/stable/0.15.18>

<https://downloads.chef.io/chef-workstation/stable/0.15.18>

## GL: Installing Chef Workstation

After it downloads, launch the installer.

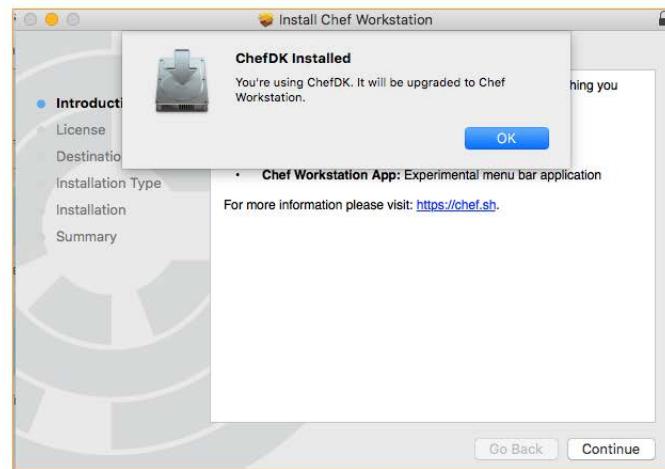
Then double-click the Chef Workstation package.



Follow the Chef Workstation installation wizard's instructions. It could take a few minutes to install.

## GL: Installing Chef Workstation

If you get this prompt, allow your old ChefDK to be upgraded to Chef Workstation.



Follow the installation wizard's instructions. It could take over a few minutes to install it.

Chef Workstation will be installed into an opscode folder on your laptop.



# Installing the Chef Workstation

*Installing the tools on your system*

## **Objective:**

- Install Chef Workstation
- Execute commands to ensure everything is installed
- Install a text editor (optional)

## GL: On Your Laptop Verify Installation of ChefDK



```
> chef --version
```

```
Chef Workstation version: 0.15.16
Chef Infra Client version: 15.7.32
Chef InSpec version: 4.18.51
Chef CLI version: 2.0.0
Test Kitchen version: 2.3.4
Cookstyle version: 5.20.0
```

Open a bash session or something like Windows Power Shell and then run this command.

We see some familiar tools like 'chef-client', and test kitchen. These are the ones that we have used on our remote workstation. Some of these tools you have not seen yet.

## Lab: Accept All Chef Licenses on Your Laptop



```
$ sudo chef env
```

```
+-----+
 Chef License Acceptance

Before you can continue, 3 product licenses
must be accepted. View the license at
https://www.chef.io/end-user-license-agreement/

Licenses that need accepting:
 * Chef Workstation
 * Chef Infra Client
 * Chef InSpec

Do you accept the 3 product licenses (yes/no)?

> yes

Persisting 3 product licenses...
✓ 3 product licenses persisted.
```

Here we are pre-emptively accepting all Chef licenses.

Accepting Chef licenses is mandatory in all environments from workstation, to chef-client, Chef Habitat, etc.

(If you installed the latest version of Chef Workstation atop an older version where you already accepted the licenses, the license acceptance may not be apparent in the output.)



## A Note About Chef Workstation

Chef Workstation comes with the new **chef-run** utility, which can be used to execute Chef code on any remote system accessible via SSH or WinRM. It is typically used for ad hoc tasks.

You can learn more about the chef-run utility at these links:

<https://www.chef.sh/about/chef-workstation/>

<https://learn.chef.io/modules/try-chef-infra#/>

Chef Workstation comes with the new **chef-run** utility, which can be used to execute chef code on any remote system accessible via SSH or WinRM. This provides a quick way to apply config changes to the systems you manage whether or not they're being actively managed by Chef, without requiring any pre-installed software. With **chef-run**, you can execute individual resources, or pre-existing Chef recipes on any number of servers with a single, simple command.

<https://www.chef.sh/about/chef-workstation/>

This Learn Chef Rally module provides an overview of using Chef Workstation:

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



# Installing Chef Workstation

*Installing the tools on your system*

## Objective:

- ✓ Install Chef Workstation
- ✓ Execute commands to ensure everything is installed
- Install a text editor (optional)



## 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>

## Visual Studio Code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running and version control. - <https://code.visualstudio.com>

## Sublime Text

A sophisticated text editor for code, markup and prose - <https://www.sublimetext.com/>

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, VSC, or Sublime at this time, if you don't already have it.



# Installing Tools

*Installing the tools on your system*

## **Objective:**

- ✓ Install Chef Workstation
- ✓ Execute commands to ensure everything is installed
- ✓ Install a text editor (optional)

## Q&A

What questions can we answer for you?





**CHEF**<sup>TM</sup>

---

©2020 Chef Software Inc.

## Signing Up For Managed Chef

Get to Know the Benefits

# Objectives

After completing this module, you should be able to

- Utilize a Managed Chef Account and create a new organization
- Download a starter kit
- Connect your local workstation (laptop) to a Chef Server

In this module you will learn how to connect your local workstation to a Chef Server via a Hosted Chef Account.



## Hosted Chef

*A Hosted Chef Account will allow us to manage our cookbooks across multiple nodes.*

### **Objective:**

- Create a Hosted Chef Account
- Create organization
- Download Starter Kit
- Extract Starter Kit and place chef-repo on local machine
- Verify successful communication with Chef Server

In the interest of getting things done with a relatively small node count, it seems like the Hosted Chef Server option is best.

Instructor Note: We are using hosted chef in this course until Chef Infra Server is fully integrated into the Chef Automate UI.

# Signing Up for a Hosted Chef Account

## Steps

1. Navigate to <https://manage.chef.io>
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 'Sign In' page for the Chef Manage interface. At the top, there's a logo with the word 'CHEF' and 'MANAGE'. Below it, the 'Sign In' heading is centered. There are two input fields: 'Username or Email Address' and 'Password', each with a corresponding text input box. Below these fields is a 'Sign In' button. To the right of the input fields, there's a section titled 'Don't have an account?' containing descriptive text about the benefits of using Chef and a link 'Click here to get started!'. At the bottom left of the form area, there's a link 'Forgot your password?'

To get started with Hosted Chef Server, visit the Chef website and sign up for a Hosted Chef Account.

# Signing Up for a Hosted Chef Account

## Steps

1. Navigate to <https://manage.chef.io>
2. Fill out the form as indicated in this image using your name and a valid email address and then click **Get Started**.

### Note

You should write down your new user name and remember your password.

#### Start your free trial of Hosted Chef

You're one step away from access to all the power and flexibility of Chef. Get ready to automate your infrastructure, accelerate your time to market, manage scale and complexity, and safeguard your systems. Just complete the form to get started.

Full Name

Company

Email

Username



I agree to the [Terms of Service](#) and the [Master License and Services Agreement](#).

[Get Started](#)

<https://manage.chef.io>

Instructor Note: Learners that already have an account can login instead of creating an account. The learner's account may be tied to a 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.



## Hosted Chef

*A Hosted Chef Account will allow us to manage our cookbooks across multiple nodes.*

### **Objective:**

- ✓ Create a Hosted Chef Account
- Create organization
- Download Starter Kit
- Extract Starter Kit and place chef-repo on local machine
- Verify successful communication with Chef Server

# Signing Up for a Hosted Chef Account

## Steps

3. From the resulting page, click the **Create New Organization** button.
4. Fill out the resulting Create Organization form and then click Create Organization.
5. From the resulting page, click your new organization to highlight it and then click Starter Kit.
6. From the resulting window, click the Download Starter Kit button.

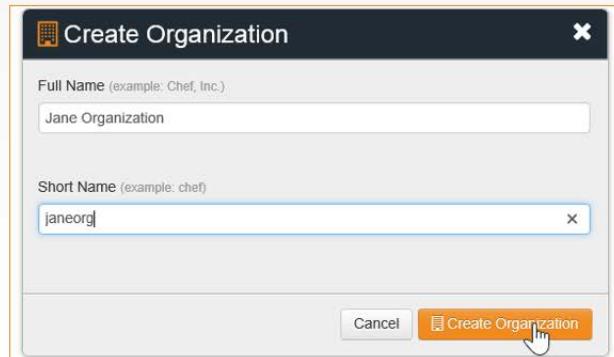


Instructor Note: Learners that already have an account will already have an organization. They are welcome to use that organization if it does not have any cookbooks or nodes from previous exercises or from their production systems. It is often easier to have them create a new organization for the purposes of this training. That can be done through the website.

# Signing Up for a Hosted Chef Account

## Steps

3. From the resulting page, click the Create New Organization button.
4. Fill out the resulting Create Organization form and then click **Create Organization**.
5. From the resulting page, click your new organization to highlight it and then click Starter Kit.
6. From the resulting window, click the Download Starter Kit button.



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.



## Hosted Chef

*A Hosted Chef Account will allow us to manage our cookbooks across multiple nodes.*

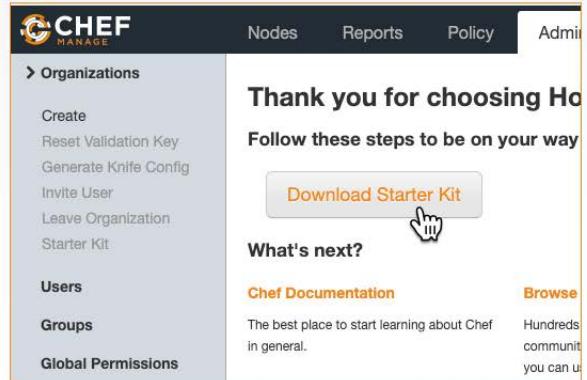
### **Objective:**

- ✓ Create a Hosted Chef Account
- ✓ Create organization
- Download Starter Kit
- Extract Starter Kit and place chef-repo on local machine
- Verify successful communication with Chef Server

# Signing Up for a Hosted Chef Account

## Steps

3. From the resulting page, click the Create New Organization button.
4. Fill out the resulting Create Organization form and then click Create Organization.
5. From the resulting page, click **Download Starter Kit**.
6. From the resulting window, click the Download Starter Kit button.

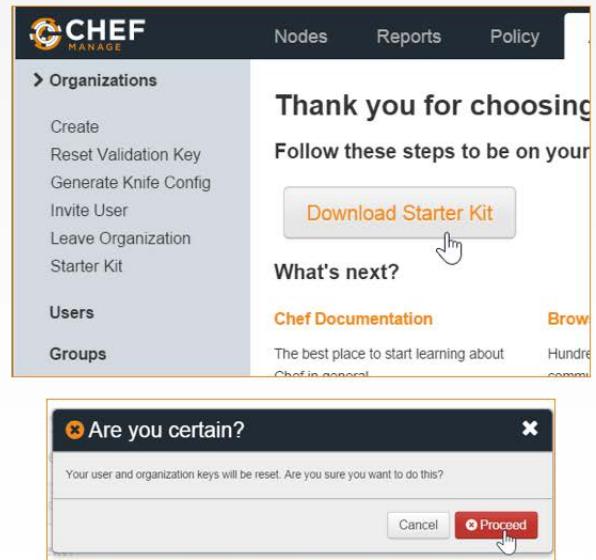


Instructor Note: By far the easiest way to get setup with Managed Chef Server is download the Starter Kit. The most important pieces of the starter kit are found in a hidden directory (".".chef") which contains the organization key, user key, and organization configuration file.

# Signing Up for a Hosted Chef Account

## Steps

3. From the resulting page, click the Create New Organization button.
4. Fill out the resulting Create Organization form and then click Create Organization.
5. From the resulting page, click your new organization to highlight it and then click Starter Kit.
6. From the resulting window, click the [Download Starter Kit](#) button.



The starter kit will warn that it will reset your organization key and personal 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: If the learner already has an account and an organization tied to that account this will reset their personal key and organizational key. This means that the other chef repository that they were previously maintaining will no longer be able to communicate with the Chef Server.



## Hosted Chef

*A Hosted Chef Account will allow us to manage our cookbooks across multiple nodes.*

### **Objective:**

- ✓ Create a Hosted Chef Account
- ✓ Create organization
- ✓ Download Starter Kit
- Extract Starter Kit and place chef-repo on local machine
- Verify successful communication with Chef Server

# Signing Up for a Hosted Chef Account

## Steps

7. Open the download zip file and copy the **chef-repo** folder that's contained in the zip file.
8. Paste the **chef-repo** folder to a location on your laptop, such as your home directory

Name

 chef-repo

### 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.



## Hosted Chef

*A Hosted Chef Account will allow us to manage our cookbooks across multiple nodes.*

### **Objective:**

- ✓ Create a Hosted Chef Account
- ✓ Create organization
- ✓ Download Starter Kit
- ✓ Extract Starter Kit and place chef-repo on local machine
- Verify successful communication with Chef Server

## Verify Communication with Chef Server



```
> cd ~/chef-repo
> knife client list
```

```
your_org_name-validator
```

You will learn more about knife in  
the next module.

At this point go ahead and open up terminal or Powershell on your local machine and run this command. Seeing the short name of your organization followed by –validator indicates that you are successfully communicating with your Chef Server and are ready for the following labs.



## Hosted Chef

*A Hosted Chef Account will allow us to manage our cookbooks across multiple nodes.*

### **Objective:**

- ✓ Create a Hosted Chef Account
- ✓ Create organization
- ✓ Download Starter Kit
- ✓ Extract Starter Kit and place chef-repo on local machine
- ✓ Verify successful communication with Chef Server



## Q&A

What questions can we answer for you?



**CHEF**<sup>TM</sup>

---

©2020 Chef Software Inc.



The Chef Infra Server  
A Hub for Configuration Data

1W and 1L

©2020 Chef Software Inc. 10-1 

Instructor Note: The 1W means each student will need their Day 2 Windows node available and running during this module.

As of this writing, specifically Essentials – Windows 2012 Node – 1.0.2 (ami-4a80ac20).

We'll also be reusing the Linux node you used in Module 7 Lab and bootstrapping it for another purpose.

# Objectives



After completing this module, you should be able to

- Describe the purpose of the Chef Infra Server
- Clone cookbooks from a GitHub repository
- Connect your local workstation (laptop) to a Chef Infra Server
- Bootstrap two nodes

In this module you will learn about the purpose of the Chef Infra Server, clone cookbooks from a GitHub repository, and connect your local workstation (laptop) to a Chef Infra Server. You will also bootstrap that will eventually become web servers.

## Managing an Additional System Without Chef Infra Server



To manage another system, you would need to:

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

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

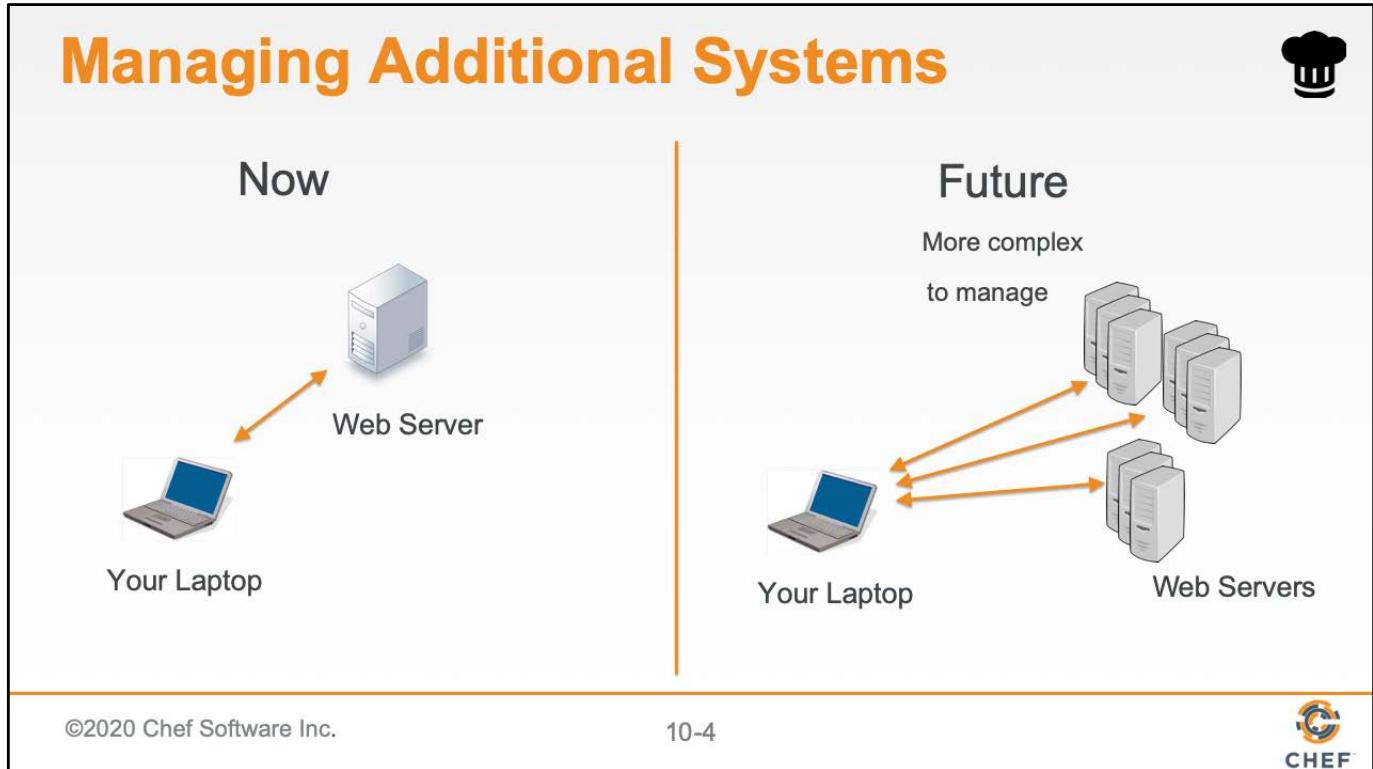
A new system would require us to Log into and provision a new node within your company or appropriate cloud provider with the appropriate access to login to administrate the system.

Install the Chef tools.

Transfer the apache or myiis cookbook to the new node.

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.



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.

# The Chef Infra Server



An easier way to set up and maintain multiple nodes.

Local Workstation  
(Your laptop)



Chef Infra Server

Cookbooks etc.

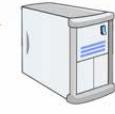
Cookbooks etc.

Cookbooks etc.

Web Server



Web Server



Load Balancer

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

The Chef Infra Server is designed to help us manage multiple nodes in this situation. The Chef Infra Server acts as a hub for configuration data. The Chef Infra Server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by 'chef-client'. Nodes, such as web servers, load balancers, etc., use 'chef-client' to ask the Chef Infra Server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef Infra Server). In a production environment, the 'chef-client' runs in an automated mode—it polls the Chef Infra Server for updates at set intervals and then applies any configuration changes. This scalable approach distributes the configuration effort throughout the organization.



## GL: Download Cookbooks and Bootstrap Nodes

*We'll download copies of the cookbooks we wrote previously in this course and test our connection to Chef Infra Server*

### Objective:

- Download and copy the required cookbooks to your local machine
- Confirm that you can connect to the Chef Infra Server
- Bootstrap two nodes

We are going to need a copy of the myiis and apache cookbooks that we created on day one and eventually get this up to the Chef Infra Server.



## GL: Code Repository

This GitHub repository contain copies of the work that you have done up to this point for the 'myiis' and 'apache' cookbooks:

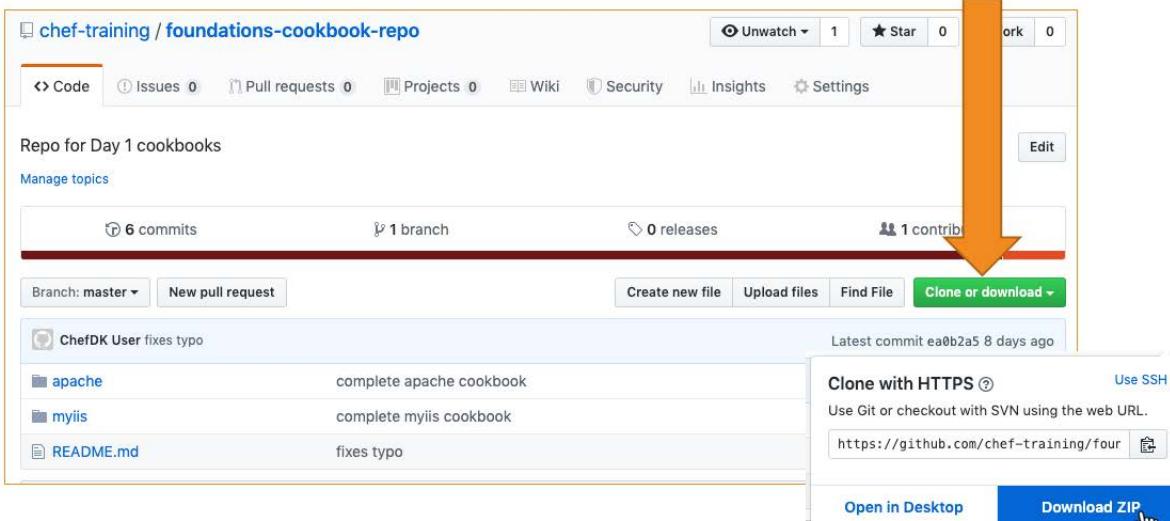
<https://github.com/chef-training/foundations-cookbook-repo>

<https://github.com/chef-training/foundations-cookbook-repo>

The 'apache' and 'myiis' cookbooks that were created from previous modules can be found here. These may not be exact copies of cookbooks that you created but will function for the purpose of the class.

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 Cookbooks Repository

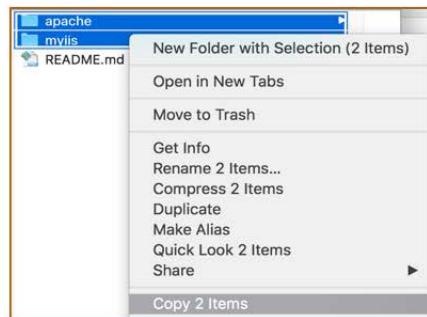


## GL: Move Cookbooks to the Cookbooks Folder

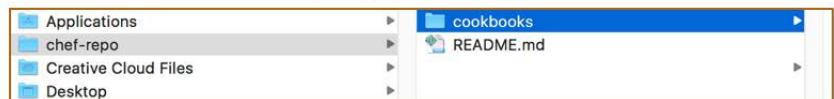
### Steps:

1. Open the downloaded [foundations-cookbook-repo](#) zip file and then copy **only** the **apache** and **myiis** folders.
2. Paste the **apache** and **myiis** folders into the **cookbooks** folder of your chef-repo directory.

foundations-cookbooks-repo-master



chef-repo/cookbooks



## GL: Navigate to the cookbooks Directory

```
└─ $ cd ~/chef-repo/cookbooks
└─ $ ls
```

```
apache chefignore myiis starter
```

Open a terminal or Powershell and navigate to the chef-repo/cookbooks directory. You should see the cookbooks you just placed within the directory.

## GL: Remove the starter Example Cookbook



```
$ rm -rf starter
```

**Note:** Powershell users can use '**Remove-Item starter**'

The 'starter' cookbook is a generic example of a cookbook that came with the starter kit and is not necessary to keep for class.

## GL: Navigate to the 'myiis' Directory



```
$ cd ~/chef-repo/cookbooks/myiis
$ ls -l (or dir if using Powershell)
```

```
total 56
-rwxr-xr-x@ 1 sdeflante staff 148 Jul 10 07:33 CHANGELOG.md
-rwxr-xr-x@ 1 sdeflante staff 70 Jul 10 07:33 LICENSE
-rwxr-xr-x@ 1 sdeflante staff 504 Jul 10 07:33 Policyfile.rb
-rwxr-xr-x@ 1 sdeflante staff 53 Jul 10 07:33 README.md
-rwxr-xr-x@ 1 sdeflante staff 1176 Jul 10 07:33 chefignore
-rwxr-xr-x@ 1 sdeflante staff 741 Jul 10 07:33 kitchen.yml
-rwxr-xr-x@ 1 sdeflante staff 717 Jul 10 07:33 metadata.rb
drwxr-xr-x@ 4 sdeflante staff 128 Jul 10 07:33 recipes
drwxr-xr-x@ 4 sdeflante staff 128 Jul 10 07:33 spec
drwxr-xr-x@ 3 sdeflante staff 96 Jul 10 07:33 templates
drwxr-xr-x@ 3 sdeflante staff 96 Jul 10 07:33 test
```

Here you should find that the myiis cookbook has all of the work that we did on Day 1.

## GL: Navigate to the 'apache' Directory



```
$ cd ~/chef-repo/cookbooks/apache
$ ls -l (or dir if using Powershell)
```

```
-rwxr-xr-x@ 1 sdeflante staff 150 Jul 10 2019 CHANGELOG.md
-rwxr-xr-x@ 1 sdeflante staff 70 Jul 10 2019 LICENSE
-rwxr-xr-x@ 1 sdeflante staff 507 Jul 10 2019 Policyfile.rb
-rwxr-xr-x@ 1 sdeflante staff 54 Jul 10 2019 README.md
-rwxr-xr-x@ 1 sdeflante staff 1176 Jul 10 2019 .chefignore
-rwxr-xr-x@ 1 sdeflante staff 741 Jul 10 2019 kitchen.yml
-rwxr-xr-x@ 1 sdeflante staff 722 Jul 10 2019 metadata.rb
drwxr-xr-x@ 4 sdeflante staff 128 Jul 10 2019 recipes
drwxr-xr-x@ 4 sdeflante staff 128 Jul 10 2019 spec
drwxr-xr-x@ 3 sdeflante staff 96 Jul 10 2019 tests
drwxr-xr-x@ 3 sdeflante staff 96 Jul 10 2019 .travis.yml
```

We will upload these cookbooks to Chef Server later in this course via a Policyfile.

Here you should find that the apache cookbook has all of the work that we did on Day 1.



## knife

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

knife is a command-line tool that allows us to request and send information to the Chef Infra Server.

knife helps users manage nodes, cookbooks, roles, environments, and more. knife does this through a series of sub-commands.

## GL: Run 'knife node --help'



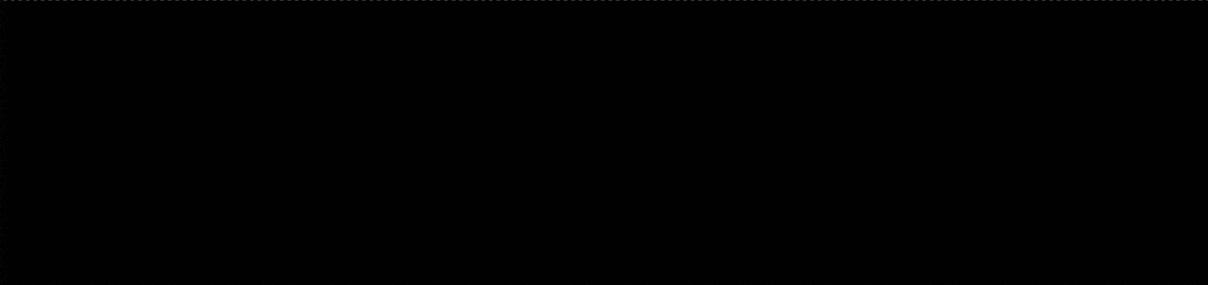
```
$ knife node --help
```

```
** NODE COMMANDS **
knife node bulk delete REGEX (options)
knife node create NODE (options)
knife node delete NODE (options)
knife node edit NODE (options)
knife node environment set NODE ENVIRONMENT
knife node from file FILE (options)
knife node list (options)
knife node run_list add [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list remove [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list set NODE ENTRIES (options)
knife node show NODE (options)
```

## GL: Run 'knife node list'



```
$ knife node list
```



We will use knife for a number of tasks in a moment.

Run "knife node list" to see that you have no nodes currently registered with your Chef Infra Server. At this point the results should be blank but this shows that we can connect to our Chef Infra Server.



## GL: Download Cookbooks and Bootstrap Nodes

*We'll download copies of the cookbooks we wrote previously in this course and test our connection to Chef Infra Server*

### Objective:

- ✓ Download and copy the required cookbooks to your local machine
- ✓ Confirm that you can connect to the Chef Infra Server.
- ❑ Bootstrap two nodes



## GL: Bootstrap Your Nodes

In this lab you will use a new Windows instance and bootstrap it as a managed node. You will do the same to Day 1 Linux node as well.

You'll need the FQDN or Public IP of those instances to perform this lab.

At this point we will be placing under management a Windows machine to act as our IIS Web server.



## Bootstrapping a Node

Often, the node you are bootstrapping may not have Chef installed. It may also not have details of where the Chef Infra Server is located or the credentials to securely talk to that Server.

To add those credentials we can **bootstrap** that node to install all those components.

<https://learn.chef.io/modules/beyond-the-basics#/>

We want to add a new instance as a node within our organization. Often times, the node you are bootstrapping may not have Chef installed on it. It also probably does not know where the Chef Infra Server is or have the credentials to even talk to it securely. We could manually configure a node but there is an easier way of doing that through a process called 'bootstrapping'.

Bootstrapping will install Chef if necessary and then configure the node to talk securely to a specified Chef Infra Server.

<https://learn.chef.io/modules/beyond-the-basics#/>

## GL: Change to the chef-repo



```
$ cd ~/chef-repo
```

Return to the root of the chef repository.

## 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 Infra Server; Running chef-client to apply a default run list.

## GL: Bootstrap Your Windows Node



```
> knife bootstrap -o winrm IPADDRESS -U Administrator -P PWD -N iis_web
```

Connecting to 34.195.38.226  
Creating new client for iis\_web  
Creating new node for iis\_web  
Bootstrap [34.195.38.226] Fully Qualified Domain Name or IP  
...  
[34.195.38.226] C:\Users\Administrator\Documents\chef-client\log\2020-07-22T20:49:28+00:00, chef/client.rb [c:/chef/first-boot.json]  
[34.195.38.226] +-----  
[34.195.38.226] "2 product licenses accepted."  
[34.195.38.226] +-----  
[34.195.38.226] Starting Chef Infra Client, version 15.1.36  
[[34.195.38.226] [2020-07-22T20:49:28+00:00] WARN: Node iis\_web has an empty run list.  
...  
[34.195.38.226] Running handlers complete  
[34.195.38.226] Chef Infra Client finished, 0/0 resources updated in 30 seconds

The licenses were accepted because we ran this command from our laptops which already have accepted the licenses.

©2020 Chef Software Inc.

10-22



To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-U' flag and the password '-P' flag. Name the node with the '-N' flag. This is optional but makes it easier for us to communicate with the node. When we ask you to look at the details of iis\_web or login to iis\_web, it will be easier to remember than the fully-qualified domain name.

When executing the command, the output will tell us what it installed and ran.

## GL: Run 'knife node list'



```
$ knife node list
```

```
iis_web
```

Run "knife node list" to see that you have no nodes currently registered with your Chef Infra Server. At this point there just one node.

## GL: View More Information About Your Node



```
$ knife node show iis_web
```

```
Node Name: iis_web
Environment: _default
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 3.88.178.251
Run List:
Roles:
Recipes:
Platform: windows 6.3.9600
Tags:
```

You can see more information about a particular node with the command 'knife node show iis\_web'. This will display a summary of the node information that the Chef Infra Server stores.

## GL: Bootstrap Your Day 1 Linux Node



```
> knife bootstrap IP -U USER -P PWD --sudo -N apache_web
Bootstrapping 18.206.64.141
[18.206.64.141] -----> Existing Chef Infra Client installation detected
[18.206.64.141] Starting the first Chef Infra Client Client run...
[18.206.64.141] Fully Qualified Domain Name or IP
[18.206.64.141] [REDACTED] Client, version 15.7.32
[18.206.64.141] [REDACTED] ks for run list: []
[18.206.64.141] [REDACTED] kbo user name
[18.206.64.141] [REDACTED] password
[18.206.64.141] [REDACTED] node name
[18.206.64.141] Installing Cookbook Gems...
Compiling Cookbooks...
[2020-02-10T21:53:19+00:00] WARN: Node apache_web has an empty run list.
[18.206.64.141] Converging 0 resources
[18.206.64.141]
Running handlers:
Running handlers complete
Chef Infra Client finished, 0/0 resources updated in 03 seconds
```

To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-U' flag and the password '-P' flag. Name the node with the '-N' flag. This is optional but makes it easier for us to communicate with the node.

When executing the command, the output will tell us what it installed and ran.

If you get this, type Y:

Connecting to 18.206.64.141

The authenticity of host '18.206.64.141 ()' can't be established.  
fingerprint is SHA256:q7AVSJ3Ai6fNFG8u/uwnUGOMP1MZo7QQrG8KwLSUml.

Are you sure you want to continue connecting  
? (Y/N)

## GL: Run 'knife node list'



```
$ knife node list
```

```
apache_web
iis_web
```

Run "knife node list" to see that you now have two nodes currently registered with your Chef Infra Server.

## GL: View More Information About Your Node



```
$ knife node show apache_web
```

```
Node Name: apache_web
Environment: _default
FQDN: ip-172-31-62-68.ec2.internal
IP: 18.206.64.141
Run List:
Roles:
Recipes:
Platform: centos 7.6.1810
Tags:
```

You can see more information about a particular node with the command 'knife node show apache\_web'. This will display a summary of the node information that the Chef Infra Server stores.



## GL: Download Cookbooks and Bootstrap Nodes

*We'll download copies of the cookbooks we wrote previously in this course and test our connection to Chef Infra Server*

### Objective:

- ✓ Download and copy the required cookbooks to your local machine
- ✓ Confirm that you can connect to the Chef Infra Server.
- ✓ Bootstrap two nodes



## Review Questions

1. What is the benefit of storing cookbooks in a central repository?
2. What is the primary tool for communicating with the Chef Infra Server?
3. How did you add a node to your organization?

1. So it can be used at scale for a number of nodes.
2. knife
3. When we bootstrapped the node, its organization was set based on the validator.pem and API in our knife.rb.

For example:

```
~/chef-repo/.chef
more knife.rb
```

```
See http://docs.chef.io/config_rb_knife.html for more information on knife
configuration options
```

```
current_dir = File.dirname(__FILE__)
log_level :info
log_location STDOUT
node_name "janejones"
client_key "#{current_dir}/janejones.pem"
chef_server_url "https://api.chef.io/organizations/janejones"
cookbook_path ["#{current_dir}/../cookbooks"]
```



## Q&A

What questions can you help you answer?

- Chef Infra Server
- knife
- Bootstrapping Nodes

With all of the objectives complete you are finished with this section. What questions can I answer for you?



©2020 Chef Software Inc.

## Policyfiles

Combining the functions of Berkshelf, Environments, and Roles into a single artifact

# Objectives

After completing this module, you should be able to

- Explain what Policyfiles are used for
- Use Policyfiles to set run lists for your nodes
- Describe how Policyfiles replace the legacy Roles, Environments, and Berkshelf
- Create a policyfile and a policyfile.lock.json
- Push the policyfile.lock.json to Chef Infra Server and converge a node

In this module you will learn how to use Policyfiles. Later in this course you will use Policyfiles when you manage a number of nodes.



## Policyfiles

A Policyfile (aka Policyfile.rb) is a file that contains information about a node's:

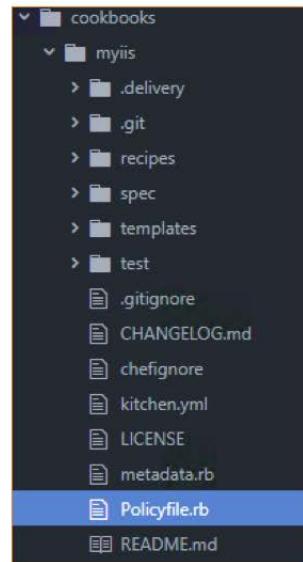
- Policy\_name
- Default source for fetching cookbooks
- Run list (or multiple run lists)
- Custom cookbook paths
- Optional cookbook attributes

# Policyfiles

When you generate a Chef cookbook using a modern version of Chef Workstation, a Policyfile.rb file is automatically created. (\* see speaker notes below slide)

Notice there is no longer a Berksfile generated like in older versions of Chef Workstation/ChefDK.

A Policyfile is the best way to manage run lists, and community cookbook data with a single document that is uploaded to the Chef Infra Server.



\*Older versions of ChefDK also support Policyfiles but the creation of the Policyfile was not automated by default. You would need to specify a -P option. For example, `chef generate cookbook COOKBOOKNAME -P`

When Policyfiles are autogenerated, they are named Policyfile.rb, which Test Kitchen automatically recognizes. This is fine if the policyfile is living within the cookbook. The name can be changed to a more meaningful name if it is outside of the cookbook. If this is done, it needs to be pointed out in test-kitchen and when installing and pushing. i.e., `chef install POLICYNAME` when the name is different.

# Policyfile

This is an example of the Policyfile that was auto generated when you ran `chef generate cookbook myiis` earlier in this course.

**name:** Used as not just a name for this policyfile, but it replaces the old **role** object. Use a name that reflects the purpose of the machines where the policy will run.

The name must be unique.

Nodes using this policyfile will possess the **myiis** role.

```
Policyfile.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with
7 name 'myiis'
8
9 # Where to find external cookbooks:
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order
13 run_list 'myiis::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'myiis', path: '..'
17
```

Regarding "**name:** Used as not just a name for this policyfile, but it replaces the old **role** object."

**name** lets you roll this policy out across hundreds or even thousands of nodes.

As you will learn later in this course, applying the concept of a role to a number of nodes greatly simplifies the configuration of those nodes. As a best practice, legacy role objects are replaced with the Policyfile **name** value.

# Policyfile

**default\_source:** Specifies where we get cookbooks if they're not specifically declared in cookbook section below.

This will usually be the public or a private supermarket or Chef Infra Server.

**default\_source** can also be used for an internal repository where all your cookbooks reside.

```
Policyfile.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with
7 name 'myiis'
8
9 # Where to find external cookbooks:
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order
13 run_list 'myiis::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'myiis', path: '..'
17
```

# Policyfile

**run\_list:** This sets the run list for any nodes using this Policyfile.

**cookbook `COOKBOOK`, path:** declares the non-default location where cookbooks can be found.

This may be a path to the top-level of a cookbook repository or to the ./cookbooks directory within that repository.

```
Policyfile.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with
7 name 'myiis'
8
9 # Where to find external cookbooks:
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order
13 run_list 'myiis::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'myiis', path: '..'
17
```

# Policyfile Location

A node can have only one policy, so its policyfile will likely have a run list containing multiple cookbooks. Therefore, it wouldn't make sense for the Policyfile to be in a particular cookbook like this.

So we can use the auto-generated Policyfile as a guideline for creating our own Policyfile or simply ignore it.

We recommend you create a **policyfiles** directory at the same level as the **cookbooks** directory. Then you can generate all your policyfiles within the **policyfiles** directory.

```
chef-repo-092619
├── .chef
└── cookbooks
 ├── apache
 ├── company_web
 │ ├── .delivery
 │ ├── .git
 │ ├── attributes
 │ ├── recipes
 │ ├── spec
 │ ├── templates
 │ └── test
 └── .gitignore
 └── Policyfile.rb
 └── README.md
 ├── mychef_client
 ├── myhaproxy
 ├── myilis
 ├── myusers
 └── .DS_Store
 └── cheftignore
 └── data_bags
 └── policyfiles
 └── Policyfile.lock.json
 └── Policyfile.rb
```

In practice you could generate your Policyfile anywhere you like as long as you provide the correct path to the cookbooks.

# Cookbook Location

Assuming you place your policyfiles in the recommended policyfiles directory, you would need to specify the location of its cookbooks like in this example, where

'..../cookbooks/company\_web' in this policyfile is saying, go up one level and then down into the cookbooks directory to find the cookbook(s).

```
company_web.lock.json company_web.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to build your system.
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with Chef does.
7 name 'company_web'
8
9 # Where to find external cookbooks;
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order specified.
13 run_list 'mychef_client::default','company_web::default','myusers::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'company_web', path: '../cookbooks/company_web'
17 cookbook 'myiis', path: '../cookbooks/myiis'
18 cookbook 'apache', path: '../cookbooks/apache'
19 cookbook 'mychef_client', path: '../cookbooks/mychef_client'
20 cookbook 'myusers', path: '../cookbooks/myusers'
```

In practice you could generate your policyfile anywhere you like as long as you provide the correct path to the cookbooks. But as mentioned, placing them in a single policyfiles directory would be a best practice.

# Cookbook Location

In this way, when you eventually upload your Policyfile (actually a Policyfile.lock.json) to Chef Infra Server, the required cookbooks will also be uploaded simultaneously.

```
company_web.lock.json | company_web.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to build your system.
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with Chef does.
7 name 'company_web'
8
9 # Where to find external cookbooks;
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order specified.
13 run_list 'mychef_client::default','company_web::default','myusers::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'company_web', path: '../cookbooks/company_web'
17 cookbook 'myiis', path: '../cookbooks/myiis'
18 cookbook 'apache', path: '../cookbooks/apache'
19 cookbook 'mychef_client', path: '../cookbooks/mychef_client'
20 cookbook 'myusers', path: '../cookbooks/myusers'
```

# Policyfile Naming

Also in this example is the correct way to name your policyfiles.

In this example we named the policyfile **company\_web.rb** so we can differentiate it from other policyfiles that will reside in the **policyfiles** directory.

```
company_web.lock.json company_web.rb
1 # Policyfile.rb - Describe how you want Chef Infra Client to build your system.
2 #
3 # For more information on the Policyfile feature, visit
4 # https://docs.chef.io/policyfile.html
5
6 # A name that describes what the system you're building with Chef does.
7 name 'company_web'
8
9 # Where to find external cookbooks;
10 default_source :supermarket
11
12 # run_list: chef-client will run these recipes in the order specified.
13 run_list 'mychef_client::default','company_web::default','myusers::default'
14
15 # Specify a custom source for a single cookbook:
16 cookbook 'company_web', path: '../cookbooks/company_web'
17 cookbook 'myiis', path: '../cookbooks/myiis'
18 cookbook 'apache', path: '../cookbooks/apache'
19 cookbook 'mychef_client', path: '../cookbooks/mychef_client'
20 cookbook 'myusers', path: '../cookbooks/myusers'
```

In practice you could generate your Policyfile anywhere you like as long as you provide the correct path to the cookbooks.



## Policyfile.lock.json

Before you upload your Policyfile to Chef Infra Server, you actually need to generate Policyfile.lock.json based on the Policyfile.rb.

In other words, we never upload the Policyfile.rb file. We only upload Policyfile.lock.json, which in turn enables the uploading of any required cookbooks.



## Policyfile.lock.json

Policyfile.lock.json also identifies the checksum of each cookbook in the run\_list, then creates a master checksum (the revision ID) adding up the checksum of all the cookbooks.

This revision ID is the unique identifier for this group of cookbooks and their included files. If you and I have the EXACT same cookbooks (same down to each individual character), we'll have the same revision id.

## Example: Policyfile.lock

To generate the Policyfile.lock.json if the policyfile name is **company\_web**:

```
chef install company_web.rb
```

That will take the contents of your **company\_web.rb** policyfile and convert it to the lock file.

Then you upload the **company\_web.lock.json** to the Chef Infra Server like this:

```
chef push policy_group company_web
```

Note: We will cover the **policy\_group** in a moment.



```
company_web.lock.json | company_web.rb
1 { |
2 "revision_id": "36a13b7080421c92db8bda55739f954" |
3 "name": "company_web", |
4 "run_list": [|
5 "recipe[mychef_client::default]", |
6 "recipe[company_web::default]", |
7 "recipe[myusers::default]" |
8], |
9 "included_policy_locks": [|
10], |
11 }, |
12 "cookbook_locks": { |
13 "apache": { |
14 "version": "0.1.0", |
15 "identifier": "1388ab3a29c60e80c2c8306203b8", |
16 "dotted_decimal_identifier": "5498293554103", |
17 "source": "cookbooks/apache", |
18 "cache_key": null, |
19 "scm_info": null, |
20 "source_options": { |
21 "path": "cookbooks/apache" |
22 } |
23 } |
24 } |
25}
```

# Policyfile.lock

When you generate the Policyfile.rb.lock file, a **revision\_id** is generated in the form of a hash.

That hash is the version number with which you can identify versions of this Policyfile.

```
Policyfile.lock.json
1 {
2 "revision_id": "bd6c581e1a16a3e317f043dd24f4b4f55f08352e8df83a9f5290aef0ae4",
3 "name": "myis",
4 "run_list": [
5 "recipe[myis::default]"
6],
7 "included_policy_locks": [
8],
9 "cookbook_locks": {
10 "myis": {
11 "version": "0.2.1",
12 "identifier": "17ddbd9d2c05e62af009d39b21f041e2d01cf7b",
13 "dotted_decimal_identifier": "6717730919810534.12085874017378800.724424",
14 "source": "..",
15 "cache_key": null,
16 "scm_info": {
17 "scm": "git",
18 "remote": null,
19 "revision": "d691971666b4079580636ca8958ea928cb1ca064",
20 "working_tree_clean": false,
21 "published": false,
22 "synchronized_remote_branches": [
23 ...
24],
25 },
26 "source_options": {
27 "path": "."
28 }
29 }
30 }
31 }
```



## GL: Generate a Policyfile

*In this group lab we'll practice creating a policyfile*

**Objective:**

- Create a Policyfile directory
- Generate a Policyfile for Apache, including a run list
- Generate a Policyfile.lock.json
- Push a Policyfile.lock.json to Chef Infra Server

## GL: Create a policyfiles Directory



```
cd ~/chef-repo
> mkdir policyfiles
```

## GL: Confirm the new policyfiles Directory Exists



```
> ls (or dir for Windows)
```

```
README.md cookbooks policyfiles roles
```

You can ignore the legacy roles directory.

## GL: Generate a Policyfile for Apache



```
cd ~/chef-repo/policyfiles>
> chef generate policyfile apache
```

```
Recipe: code_generator::policyfile
 * template[/Users/sdelfante/chef-repo/policyfiles/apache.rb] action create
 - create new file /Users/sdelfante/chef-repo/policyfiles/apache.rb
 - update content in file /Users/sdelfante/chef-repo/policyfiles/apache.rb
 from none to 65a9ac
 (diff output suppressed by config)
```

## GL: Confirm the new poliyfiles Exists



```
> ls (or dir for Windows)
```

```
apache.rb
```

## GL: View the Policyfile



```
> cat apache.rb
```

```
Policyfile.rb - Describe how you want Chef Infra Client to build your system.
...

A name that describes what the system you're building with Chef does.
name 'apache'

Where to find external cookbooks:
default_source :supermarket

run_list: chef-client will run these recipes in the order specified.
run_list 'apache::default'

Specify a custom source for a single cookbook:
cookbook 'example_cookbook', path: '../cookbooks/example_cookbook'
```

In the next step you'll need to add the the path to the apache cookbook.

## GL: Edit the New apache.rb Policyfile

~/chef-repo/policyfiles/apache.rb

```
#...skipping for brevity...
https://docs.chef.io/policyfile.html
A name that describes what the system you're building with Chef does.
name 'company_web'

Where to find external cookbooks:
default_source :supermarket

run_list: chef-client will run these recipes in the order specified.
run_list 'company_web::default'

Specify a custom source for a single cookbook:cookbook 'apache', path:
cookbook 'example_cookbook', path: '../cookbooks/example_cookbook'
cookbook 'apache', path: '../cookbooks/apache'
```

Add the code in green.

Notice how we need to specify the path to all dependent cookbooks.

```
A name that describes what the system you're building with
Chef does.
name 'company_web'

Where to find external cookbooks:
default_source :supermarket

run_list: chef-client will run these recipes in the order
specified.
run_list 'company_web::default'

Specify a custom source for a single cookbook (local, not
the Supermarket):
cookbook 'apache', path: '../cookbooks/apache'
```



## GL: Policyfile.rb and the Policyfile.lock.json

Now that we have our Policyfile.rb (apache.rb) we need to generate the Policyfile.lock.json (apache.lock.json) before we could upload the apache.lock.json to Chef Infra Server.

The `chef install policy_name` command creates the Policyfile.lock.json.

## GL: Generate the apache.lock.json



```
~/chef-repo/policyfiles> chef install apache.rb
```

```
Building policy apache
Expanded run list: recipe[apache::default]
Caching Cookbooks...
Installing apache 0.0.5
Installing apache2 8.3.0

Lockfile written to /Users/sdelfante/chef-repo/policyfiles/apache.lock.json
Policy revision id:
40edea4d9edf5f74e402e6d47d9e719750e451d2b9cb930a1725c28e918c17ec
```

## GL: Confirm the new apache.lock.json Exists



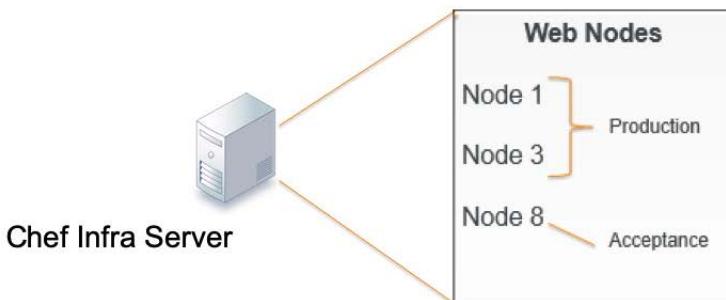
```
ls (or dir for Windows)
```

```
apache.lock.json apache.rb
```

In practice, at this point you could upload the apache.lock.json to Chef Infra Server but we need to define one more thing before we do.

## Policy Group = Environment

At the time when you upload the Policyfile.lock.json to the Chef Infra Server is when you specify a policy group, which can act like an environment such as **production** or **acceptance**. You could also think of policy group as a way to group like servers together. **Note:** Policy group replaces the legacy environments.

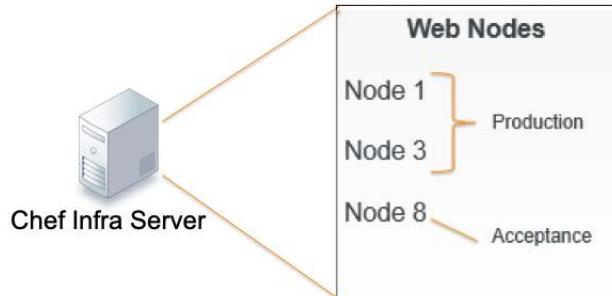


# Policy Group = Environment

A policy group can best be defined as a logical separation of nodes.

For example, let's say you have a Production environment and an Acceptance environment.

You could create a **prod** policy group and an **acceptance** policy group and push your policyfile to either one, depending where you want that policyfile's node to reside.



You likely are familiar with the concept of environments. An environment can best be defined as a logical separation of nodes that most often describe the life-cycle of an application. Each environment signifies different behaviors and policies to which a node adheres for a given application or platform.

For example, environments can be separated into 'acceptance' and 'production'. "Acceptance" would be where we may make allowances for constant change and updates and for applications to be deployed with each release. "Production" might be where we lock down our infrastructure and policies. Production would be what the outside world sees, and would remain unaffected by changes and upgrades until you specifically release them. Chef also has a concept of an environment. A Chef environment allows us to define a list of policies that we will allow by defining a cookbook.

# Policy Group

The first time you specify a policy group, that policy group name will be instantiated in Chef Infra Server. For example:

`chef push prod apache.lock.json` will create the policy group named **prod** and also upload the `apache.lock.json` (and its cookbooks) to the Chef Infra Server.

It will upload that `apache.lock.json` within the policy group named **prod**.

# Pushing a policyfile to Chef Infra Server

To push a policyfile to Chef Infra Server you need:

**Policy name**

and...

**Policy group**

These two items uniquely identify a single policyfile object on the Chef Infra Server.

## GL: Push Your apache.lock.json to prod on Chef Infra Server



```
~/chef-repo/policyfiles> chef push prod apache.lock.json
```

```
Uploading policy apache (450fb23145) to policy group prod
Uploaded apache 0.1.0 (1388ab3a)
```

## GL: Verify that Your Policy is on Chef Infra Server



```
> chef show-policy
```

```
apache
```

```
=====
```

```
* prod: 40edea4d9e
```

This output confirms that the policy is  
on the Chef infra Server.

## knife node policy set Command

In a moment you will apply the policyfile to a node.

To do so you will use the **knife node policy set** command.

Syntax:

```
knife node policy set NODE POLICY_GROUP POLICY_NAME
```

[https://docs.chef.io/workstation/knife\\_node/#policy-set](https://docs.chef.io/workstation/knife_node/#policy-set)

[https://docs.chef.io/workstation/knife\\_node/#policy-set](https://docs.chef.io/workstation/knife_node/#policy-set)

## GL: Apply the apache Policy to Your Linux (apache\_web) Node



```
> knife node policy set apache_web prod apache
```

```
Successfully set the policy on node apache_web
```

node name

policy\_group

policy\_name

'knife node policy set' takes 3 arguments:

- \* node name
- \* policy group
- \* Policy name

## GL: View Information About Your Linux Node



```
$ knife node show apache_web
```

```
Node Name: apache_web
Policy Name: apache
Policy Group: prod
FQDN: ip-172-31-90-167.ec2.internal
IP: 3.236.230.224
Run List: recipe[apache::default]
Recipes: apache::default
Platform: centos 7.6.1810
Tags:
```

You can see information about a particular node with the command 'knife node show apache\_web'. This will display a summary of the node information that the Chef Infra Server stores.

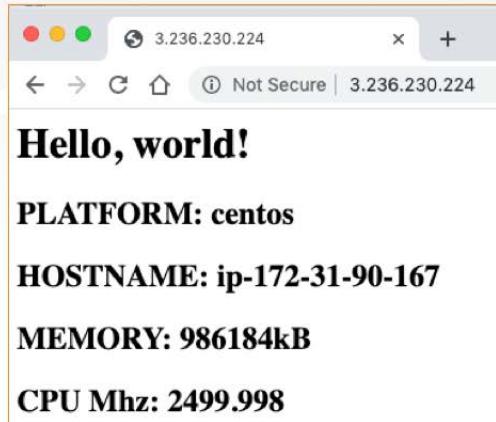
## GL: Converge the Linux Node with ssh



```
$ knife ssh IPADDRESS -m -x chef -P PWD 'sudo chef-client'

3.236.230.224 Starting Chef Infra Client, version 15.7.32
3.236.230.224 Using policy 'apache' at revision
'450fb23145c2b9635544042d413ccf25fc644964aa2f46df6114a5c1a087a94f'
3.236.230.224 resolving cookbooks for run list: ["apache::default@0.1.0 (1388ab3)"]
3.236.230.224 Synchronizing Cookbooks:
3.236.230.224 - apache (0.1.0)
3.236.230.224 Installing Cookbook Gems:
3.236.230.224 Compiling Cookbooks...
3.236.230.224 Converging 3 resources
3.236.230.224 Recipe: apache::server
3.236.230.224 * yum_package[httpd] action install
3.236.230.224 - install version 0:2.4.6-93.el7.centos.x86_64 of package httpd
3.236.230.224 * template[/var/www/html/index.html] action create
3.236.230.224 - create new file /var/www/html/index.html
...
3.236.230.224 Chef Infra Client finished, 4/4 resources updated in 06 seconds
```

## GL: Verify that the Linux Node Serves the Page



Verify that the node serves up the default html by pointing a web browser to the IP address of your Windows (apache\_web) node.



## Review Questions

1. What do policyfiles typically contain?
2. What can a policyfile's policy\_name be used for?
3. What is a policy group?

1. A policyfile contains information about a node's source for fetching cookbooks and for setting run lists.
2. To reflect the purpose of the machines where the policy will run. In other words, policy\_name can set the "role" for a node.
3. A policy group can best be defined as a logical separation of nodes into specific environments, such as production or acceptance environments.



## Q&A

What questions can we answer for you?



**CHEF**<sup>TM</sup>

---

©2020 Chef Software Inc.

## Cookbook Attributes, Attribute Files and Dependencies

Setting Attributes within a Cookbook and Applying the company\_web Cookbook to Your Nodes

# Objectives

After completing this module, you should be able to

- Explain where cookbook attributes reside
- Create a wrapper cookbook
- Configure dependencies between cookbooks
- Generate a new policyfile and policyfile.lock.json
- Upload the new policyfiles to Chef Infra server and converge the nodes

In this module you will learn how to set node attributes from with cookbooks, create wrapper cookbooks and set dependencies between cookbooks.



## Attribute Files

The Node object contains many automatic attributes generated by OHAI.

You can also maintain attributes within a cookbook.

These are like variables or parameters for your cookbook and allow recipes to be data driven.

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

While the node object that is generated by OHAI contains a large amount of data about the node, there will be situations where you want to create and set your own node attributes that can be used throughout your cookbooks and overridden when necessary. These user defined node attributes are much like variables or parameters that can allow for your cookbooks to be data driven.



## Best Practices

- ❖ Well-written cookbooks change behavior based on attributes.
- ❖ Ideally, you don't have to modify the contents of a cookbook to use it for your specific use case.
- ❖ Look at the attributes directory for things you can override through roles to affect behavior of the cookbook.
- ❖ Of course, well written cookbooks have sane defaults, and a README to describe all this.

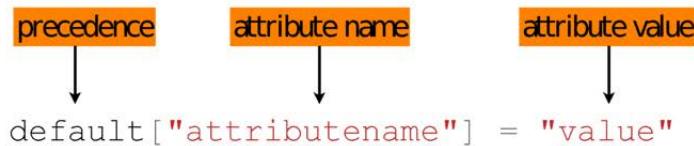
Ideally you will never have to manually edit the contents of a cookbook to be able to use it in a specific use case. The way that we can insure this is be creating cookbooks that are data driven and change their behavior based on attributes that can be overridden through the use of roles and environments. When creating node attributes within your cookbook, it is a good idea to document the purpose of this within your README file.

## Setting Attributes in Attribute Files

Cookbook attributes are set in the attributes file

`./cookbooks/<cookbook>/attributes/default.rb`

Format is:



We'll look at precedence later.

Cookbook attributes are defined in an attribute file found in an 'attributes' directory of the cookbook. You first give a precedence for the attribute which will generally be default if you are first creating this attribute. We will talk more about attribute precedence later on in the course. This is then followed by the name of the attribute that you feel describes its purpose and set this equal to some value which can later be overridden.

## Example: Setting package name to an attribute

```
cookbooks/apache/attributes/default.rb
```

```
default['apache']['package_name'] = 'httpd'
```

```
cookbooks/apache/recipes/default.rb
```

```
package node['apache']['package_name'] do
 action :install
end
```

We can set the name of a particular package to an attribute and then call that attribute within a recipe

In this example we see that we can instead of hard coding the name of the package resource, we can set this to a node attribute within the attribute file. This will resolve to 'httpd' when chef-client is run.

## Example: Setting package name to an attribute

```
cookbooks/apache/attributes/default.rb
```

```
case node['platform']
when 'ubuntu'
 default['apache']['package_name'] = 'apache2'
else
 default['apache']['package_name'] = 'httpd'
end
```

Implementing conditional statements allows us to alter the control flow permitting our cookbooks to be data driven.

If we desire for our cookbook to be data driven, we can change what the `package_name` node attribute will resolve to based on platform information. If this code is executed on a ubuntu machine, the `package_name` will resolve to 'apache2'. Otherwise, like on a Red Hat OS, this will resolve to httpd.



## Reconfigure Welcome Message

Currently a welcome message is hard coded in both web server cookbooks.

What if we wanted to display a message that includes our company name utilizing a node attribute?

How could we implement this node attribute within both our 'myiis' and 'apache' cookbooks?

Let's consider this scenario. We might want have to have our company name displayed in our welcome message. However, instead of simply hard coding this in we might set this as a node attribute in case something changes and we could simply override this if necessary. But how do we implement this in both the apache and myiis cookbooks?



## GL: Reconfigure Welcome Message

*So we want both our web server cookbooks to display  
our company name...*

**Objective:**

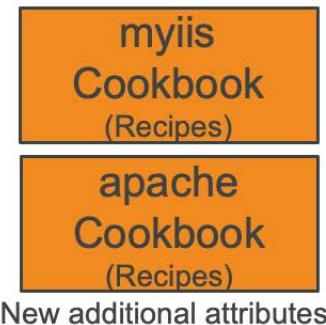
- Create a 'company\_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- Create a node attribute that contains your company name
- Implement the edit\_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- Create Policyfile and lock.
- Upload Policyfile.lock to the Chef Infra Server
- Converge the nodes

# Wrapper Cookbooks

A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook(s).

It can access all of the recipes, cookbook components, and attributes found in the original cookbook(s) and implement them in new ways.

## Wrapper Cookbook



<https://docs.chef.io/supermarket.html#wrapper-cookbooks>

<https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/>

A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook but allows us to define new default values for the recipes.

This is a common method for overriding cookbooks because it allows us to leave the original cookbook untouched. We simply provide new default values that we want and then include the recipes that we want to run.

Let's generate our wrapper cookbook named company\_web.

## GL: Generate the Wrapper Cookbook



```
$ cd ~/chef-repo
$ chef generate cookbook cookbooks/company_web
```

```
Generating cookbook company_web
- Ensuring correct cookbook content
```

```
Your cookbook is ready. To setup the pipeline, type `cd
cookbooks/company_web`, then run `delivery init`
```

Do this on your laptop

Go ahead and generate a cookbook named 'company\_web'. This will act as our wrapper cookbook that creates dependencies on both 'myiis' and 'apache'.

## GL: Create Dependency on apache and myiis

```
~/chef-repo/cookbooks/company_web/metadata.rb
```

```
name 'company_web'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures company_web'
long_description 'Installs/Configures company_web'
version '0.1.0'
chef_version '>= 14.0'
depends 'myiis'
depends 'apache'
```

We set our dependencies within the metadata.rb file of the wrapper cookbook. By doing this we now have access to all of the code base found in both 'myiis' as well as 'apache'.

## GL: Include Recipe Based on Platform

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

```

Cookbook:: company_web
Recipe:: default

Copyright:: 2020, The Authors, All Rights Reserved.

case node['platform']
when 'windows'
 include_recipe 'myiis::default'
else
 include_recipe 'apache::default'
end
```

Because we have set dependencies for the 'myiis' and 'apache' cookbooks we may now include the default recipes from these cookbooks. By using a case statement that checks the platform of the node, when this is executed on a windows machine it will apply the 'myiis' cookbook's default recipe. Else, as on a Linux machine, it will apply the 'apache' default recipe.



## GL: Reconfigure Welcome Message

*So we want both our web server cookbooks to display our company name...*

**Objective:**

- ✓ Create a 'company\_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- Create a node attribute that contains your company name
- Implement the edit\_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- Upload cookbooks to the Chef Infra Server via a policyfile
- Update the run list of the iis\_web node to use the default recipe of the company\_web cookbook and converge the node

## GL: Generate the default Attribute File

```
└─ $ cd ~/chef-repo
└─ $ chef generate --help
```

```
Available generators:
 app Generate an application repo
 cookbook Generate a single cookbook
 recipe Generate a new recipe
 attribute Generate an attributes file
 template Generate a file template
 file Generate a cookbook file
```

The chef generate command is also capable of creating an attribute file with the corresponding 'attributes' directory if it doesn't already exist.

## GL: Generate the default Attribute File



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

```
Usage: chef generate attribute [path/to/cookbook] NAME [options]
 -C, --copyright COPYRIGHT Name of the copyright holder - defaults
 to 'The Authors'
 -m, --email EMAIL Email address of the author - defaults to
 'you@example.com'
 -a, --generator-arg KEY=VALUE Use to set arbitrary attribute KEY to
 VALUE in the code_generator cookbook
 -h, --help Show this message
```

## GL: Generate the default Attribute File



```
$ chef generate attribute cookbooks/company_web default
```

Recipe: code\_generator::attribute

- \* directory[cookbooks/company\_web/attributes] action create
  - create new directory cookbooks/company\_web/attributes
- \* template[cookbooks/company\_web/attributes/default.rb] action create
  - create new file cookbooks/company\_web/attributes/default.rb
  - update content in file cookbooks/company\_web/attributes/default.rb from none to e3b0c4

Let's generate a default attribute file where we can set our own node attributes.

## GL: Set the Company Name as an Attribute



`cookbooks/company_web/attributes/default.rb`

```
default['company_web']['company_name'] = 'Your Company Name'
```

You can replace the 'Your Company Name' value with **Chef** or any name you like.

Now that we have our default attribute file, let's define a node attribute called 'company\_name' and set this to whatever value you like.



## GL: Reconfigure Welcome Message

*So we want both our web server cookbooks to display  
our company name...*

- Objective:**
- ✓ Create a 'company\_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
  - ✓ Create a node attribute that contains your company name
  - ❑ Implement the edit\_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
  - ❑ Create Policyfile and lock.
  - ❑ Upload Policyfile.lock to the Chef Infra Server
  - ❑ Converge the nodes



## Using the company\_name Attribute

We are now able to apply a different default recipe based on whether the node's platform is Windows or Centos, but how do we update the respective template file to display the company\_name attribute for both the 'myiis' and 'apache' cookbooks?

We are able to have access to this new node attribute within our cookbook, but how do we utilize it from both the 'apache' and 'myiis' cookbooks considering that they use different template resources to create the welcome page?



## **edit\_resource**

A recipe can find a resource in the resource collection, and then edit it by using the `edit_resource` method. If a resource block with the same name exists in the resource collection, it will be updated with the contents of the resource block.

[https://docs.chef.io/dsl\\_recipe.html#edit-resource](https://docs.chef.io/dsl_recipe.html#edit-resource)

One way we might go about changing the template resource is using the `edit_resource` method. When you apply a run list to a node, it creates what's known as a resource collection. This resource collection is a list of all the resources that will be applied to the node and using the `edit_resource` method we can change some of the properties of these resources. Perhaps we could change the source for the template resource?

## GL: View the server Recipes

```
/myiis/recipes/server.rb
```

```
powershell_script 'Install IIS' do
 code 'Add-WindowsFeature Web-Server'
end

template 'c:\inetpub\wwwroot\Default.htm' do
 source 'Default.htm.erb'
end

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

```
/apache/recipes/server.rb
```

```
package 'httpd'

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

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



We want to use a new source for the template resource for both our cookbooks

The 'myiis' and 'apache' cookbooks use different ERB templates for the creation of the homepage. We want to change this and use a new template that uses our new 'company\_name' node attribute.

## GL: Edit the Template resource for myiis

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

case node['platform']
when 'windows'
 include_recipe 'myiis::default'

 edit_resource(:template, 'c:\inetpub\wwwroot\Default.htm') do
 source 'homepage.erb'
 cookbook 'company_web'
 end
#Else Statement...
```

Here we add our `edit_resource` method to the 'windows' case statement. We will edit the template resource named as so and provide a new source for the template which will be found in the 'company\_web' cookbook.

## GL: Edit the Template resource for apache

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

#When Statement...
else
 include_recipe 'apache::default'

 edit_resource(:template, '/var/www/html/index.html') do
 source 'homepage.erb'
 cookbook 'company_web'
 end
end
```

Within the else statement we edit the template resource for the apache default recipe as well providing the same source for the resource.

## GL: View the default Recipe

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

case node['platform']
when 'windows'
 include_recipe 'myiis::default'

 edit_resource(:template, 'c:\inetpub\wwwroot\Default.htm') do
 source 'homepage.erb'
 cookbook 'company_web'
 end

else
 include_recipe 'apache::default'
 edit_resource(:template, '/var/www/html/index.html') do
 source 'homepage.erb'
 cookbook 'company_web'
 end
end
```

This is what the code will look like in completion for the default recipe.

```
case node['platform']
when 'windows'
 include_recipe 'myiis::default'

 edit_resource(:template, 'c:\inetpub\wwwroot\Default.htm') do
 source 'homepage.erb'
 cookbook 'company_web'
 end

else
 include_recipe 'apache::default'
 edit_resource(:template, '/var/www/html/index.html') do
 source 'homepage.erb'
 cookbook 'company_web'
 end
end
```

## GL: Generate the Template file



```
$ chef generate template cookbooks/company_web homepage
```

```
Recipe: code_generator::template
 * directory[cookbooks/company_web/templates] action create
 - create new directory cookbooks/company_web/templates
 * template[cookbooks/company_web/templates/homepage.erb] action create
 - create new file cookbooks/company_web/templates/homepage.erb
 - update content in file cookbooks/company_web/templates/homepage.erb from
 none to e3b0c4
 (diff output suppressed by config)
```

Use '**chef generate template**' to create a template in the company\_web cookbook found in the cookbooks/apache directory and the file we want to create is named homepage.erb

## GL: Update the Template File

```
~/chef-repo/cookbooks/company_web/templates/homepage.erb
```

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

Note: We are adding all this code but we are also highlighting the ['company\_web']['company\_name'] attributes for the discussion below.

Now at last we use our 'company\_name' node attribute. This is essentially the same html page that was generated from the apache and myiis cookbooks with the exception that we now have a new welcome message making use of our 'company\_name' node attribute.

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



## GL: Reconfigure Welcome Message

*So we want both our web server cookbooks to display  
our company name...*

- Objective:**
- ✓ Create a 'company\_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
  - ✓ Create a node attribute that contains your company name
  - ✓ Implement the edit\_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
  - Create Policyfile and lock.
  - Upload Policyfile.lock to the Chef Infra Server
  - Converge the nodes



## Policyfile.rb and the Policyfile.lock.json

Now that we have our `company_web` cookbook in our chef-repo, we can create our `Policyfile.rb` and then generate our `Policyfile.lock.json` as we discussed in the previous module.

We'll name our Policyfile **company\_web**.

## GL: Create the policyfiles Directory

```
💻 > cd ~/chef-repo
> mkdir policyfiles
```

## GL: Generate the Policyfile and Name it company\_web



```
> chef generate policyfile policyfiles/company_web
```

```
* template[/Users/sdelfante/chef-repo/company_web.rb] action create
 - create new file /Users/sdelfante/chef-repo/company_web.rb
 - update content in file /Users/sdelfante/chef-repo/company_web.rb from
none to 32a368
```

## GL: Verify that the Policyfile Exists



```
> ls policyfiles (or dir policyfiles for Windows)
```

company\_web.rb

## GL: Edit the New company\_web.rb Policyfile

~/chef-repo/policyfiles/company\_web.rb

```
#...skipping for brevity...
https://docs.chef.io/policyfile.html
A name that describes what the system you're building with Chef does.
name 'company_web'

Where to find external cookbooks:
default_source :supermarket

run_list: chef-client will run these recipes in the order specified.
run_list 'company_web::default'

Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
```

Add the code in code in green.

Notice how we need to specify the path to all dependent cookbooks too (myiis and apache).

```
Policyfile.rb - Describe how you want Chef Infra Client to
build your system.
#
For more information on the Policyfile feature, visit

A name that describes what the system you're building with
Chef does.
name 'company_web'

Where to find external cookbooks:
default_source :supermarket

run_list: chef-client will run these recipes in the order
specified.
run_list 'company_web::default'
```

```
Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
```

## GL: Generate the company\_web.lock.json



```
~/chef-repo> chef install policyfiles/company_web.rb

Expanded run list: recipe[company_web::default]
Caching Cookbooks...
Installing company_web >= 0.0.0 from path
Installing myiis >= 0.0.0 from path
Installing apache >= 0.0.0 from path

Lockfile written to /Users/sdelfante/chef-
repo/policyfiles/company_web.lock.json
Policy revision id:
3a1887234029070819533470d82616674e38b200413d82296af7383602b2bf0d
```

## GL: Verify that the `company_web.lock.json` Exists



```
> ls policyfiles(or dir for Windows)
```

```
company_web.lock.json company_web.rb
```

## GL: Move to the policyfiles directory



```
~/chef-repo> cd policyfiles
```

## GL: Push the company\_web.lock.json to Chef Infra Server



```
~/chef-repo/policyfiles> chef push prod
company_web.lock.json
```

```
Uploading policy company_web (3a18872340) to policy group prod
Uploaded apache 0.1.0 (1388ab3a)
Uploaded company_web 0.1.0 (b9969c7c)
Uploaded myiis 0.2.1 (cd0db3ed)
```

## GL: Verify the Policyfile.lock.json is on Chef Infra Server



```
~/chef-repo> chef show-policy
```

```
company_web
```

```
=====
```

```
* prod: 3a18872340
```

Here we can see that the **company\_web** policy has been uploaded to Chef Infra Server and is in the **prod** policy group.



## GL: Reconfigure Welcome Message

*So we want both our web server cookbooks to display  
our company name...*

**Objective:**

- ✓ Create a 'company\_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
- ✓ Create a node attribute that contains your company name
- ✓ Implement the edit\_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
- ✓ Create Policyfile and lock.
- ✓ Upload Policyfile.lock to the Chef Infra Server
- ❑ Converge the nodes

## GL: Apply the company\_web Policy to Your Windows Node



```
> knife node policy set iis_web prod company_web
```

Successfully set the policy on node iis\_web

node name

policy\_group

policy\_name

'knife node policy set' takes 3 arguments:

- \* node name
- \* policy group
- \* Policy name

## GL: View Information About Your Windows Node



```
$ knife node show iis_web
```

```
Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 3.88.178.251
Run List:
Recipes:
Platform: windows 6.3.9600
Tags:
```

You can see more information about a particular node with the command 'knife node show iis\_web'. This will display a summary of the node information that the Chef Infra Server stores.

## GL: Apply the company\_web Policy to Your Linux Node



```
> knife node policy set apache_web prod company_web
```

```
Successfully set the policy on node apache_web
```

node name

policy\_group

policy\_name

'knife node policy set' takes 3 arguments:

- \* node name
- \* policy group
- \* Policy name

## GL: View Information About Your Linux Node



```
$ knife node show apache_web
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-62-68.ec2.internal
IP: 18.206.64.141
Run List:
Recipes:
Platform: centos 7.6.1810
Tags:
```

You can see information about a particular node with the command 'knife node show apache\_web'. This will display a summary of the node information that the Chef Infra Server stores.

## GL: Converge your Windows Node via winrm



```
$ knife winrm IPADDRESS -m -x Administrator -P PASSWORD "chef-client"

...
Synchronizing Cookbooks:
- myiis (0.2.1)
3.88.178.251 - apache (0.1.0)
3.88.178.251
3.88.178.251 - company_web (0.1.0)
3.88.178.251 Installing Cookbook Gems:
3.88.178.251 Compiling Cookbooks...
* windows_service[w3svc] action start (up to date)
...
3.88.178.251 Running handlers:
3.88.178.251 Running handlers complete
3.88.178.251 Chef Infra Client finished, 2/4 resources updated in 48 seconds
```

So if you want to execute "chef-client" run for your iis\_web node, you should write out this command. You would need to provide the user name to log into the system, the password for that system, and then finally the command to execute. For more security, you should likely use API or ssh keys and forego specifying a username and password

If you have a cookbook error and need to update and re-push your policyfile:

```
chef update company_web.rb
chef push prod company_web.lock.json
```

## GL: Verify that the Windows Node Serves the Page



Verify that the node serves up the default html by pointing a web browser to the IP address of your Windows (iis\_web) node.

## GL: Converge the Linux Node with ssh



```
$ knife ssh IPADDRESS -m -x chef -P PWD 'sudo chef-client'
```

```
...
Synchronizing Cookbooks:
- myiis (0.2.1)
3.88.178.251 - apache (0.1.0)
3.88.178.251
3.88.178.251 - company_web (0.1.0)
3.88.178.251 Installing Cookbook Gems:
3.88.178.251 Compiling Cookbooks...
* windows_service[w3svc] action start (up to date)
...
3.88.178.251 Running handlers:
3.88.178.251 Running handlers complete
3.88.178.251 Chef Infra Client finished, 2/4 resources updated in 48 seconds
```

## GL: Verify that the Linux Node Serves the Page



Verify that the node serves up the default html by pointing a web browser to the IP address of your Windows (apache\_web) node.



## Review

Attributes are like parameters to your cookbook, not hard-coded values in recipes or templates. Can you think of some other parameters that you might want to create attributes for?

Can you imagine in complex topologies where you could have multiple levels of dependencies between cookbooks?



## GL: Reconfigure Welcome Message

*So we want both our web server cookbooks to display  
our company name...*

- Objective:**
- ✓ Create a 'company\_web' wrapper cookbook that can apply either the 'myiis' or 'apache' default recipe based on platform
  - ✓ Create a node attribute that contains your company name
  - ✓ Implement the edit\_resource method to update the template resource for both the 'myiis' and 'apache' server recipes
  - ✓ Create Policyfile and lock.
  - ✓ Upload Policyfile.lock to the Chef Infra Server
  - ✓ Converge the node



## 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.



©2020 Chef Software Inc.

## Community Cookbooks

Find, Explore and View Chef Cookbooks

1L

©2020 Chef Software Inc.

13-1



Instructor Note: The "1L" means you will need 1 new Linux node for each student in this module. It will be bootstrapped and used as a load balancer. Use a new fresh Linux instance for this...not the Linux node used previously in this course.

# Objectives

After completing this module, you should be able to

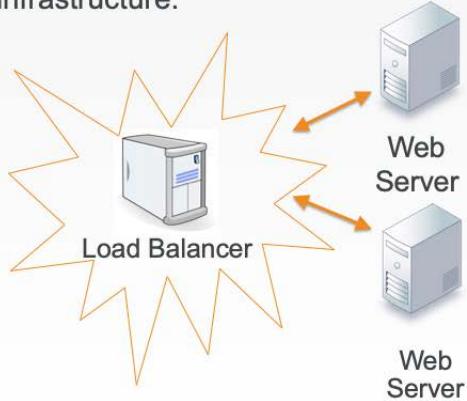
- Find cookbooks on the Chef Supermarket
- Create a wrapper cookbook for a community cookbook
- Utilize Custom Resources
- Create and upload a Policyfile to Chef Infra Server
- Bootstrap a new node that runs the Policyfile's cookbook
- Test the load balancer

In this module you will learn how to find cookbooks on the Chef Supermarket, create a wrapper cookbook, replace the existing default values, upload a cookbook to Chef Server, and bootstrap a new node that runs the cookbook

# Load Balancer

Adding a load balancer will allow us to better grow our infrastructure.

Receives requests and relays them to other systems.



With two web servers running with our organization, it's now time to talk about the next goal to tackle. We need to setup a load balancer.

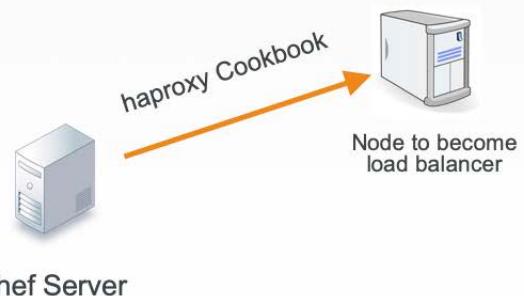
A load balancer is able to receive requests and relay them to other systems. In our case, we specifically want to use the load balancer to balance the entire traffic load between two or more systems.

This means we will need to establish a new node within our organization, install the necessary software to make the node a load balancer, and configure it so that it will relay requests to our existing nodes running apache and iis and to future nodes.

# Load Balancer

Work that needs to be accomplished to setup a load balancer within our infrastructure:

- Write a haproxy (load balancer) cookbook.
- We will need to establish a new node within our organization to which we apply that cookbook.



Similar to how we installed and configured apache on our first node, we could do the same thing here with a load balancer. We could learn the package name for the application 'haproxy', learn which file manages the configuration, learn how to compose the configuration with custom values, and then manage the service.

Package, Template and Service are the core of configuration management. Nearly all the recipes you write for an application will center on using these three resources. We could spend some time focused on composing the cookbook recipe and testing it on our platform with our custom configuration.



## Community Cookbooks

Someone already wrote that cookbook?

Available through the community site called the Chef Supermarket

<https://supermarket.chef.io>

But what if we told you someone already wrote that cookbook?

Someone already has and that cookbook is available through the community site called Supermarket. Supermarket is a public repository of all the cookbooks shared by other developers, teams, and companies who want to share their knowledge and hard work with you to save you time.



## Types of Community Cookbooks

Chef Community Cookbooks fall into two broad categories: **Resource Cookbooks** and **Recipe Cookbooks**.

There is no strict naming convention, and the patterns can be mixed. Generally Resource Cookbooks provide extensions for Chef Infra by defining new Chef Resources, and Recipe Cookbooks use these resources to declare how a system should be configured.



## Types of Community Cookbooks

So far this class has focused on writing Recipe Cookbooks. We have written Recipes and declared Resources within those Recipes to specify how a system should be configured.

But you can extend Chef Infra by writing your own Resources! Now we will examine a Community Cookbook that provides an extension to Chef Infra by defining a new type of resource we can use inside of our Recipes.



## Group Lab: Load Balancer

*Adding a load balancer will allow us to better grow our infrastructure.*

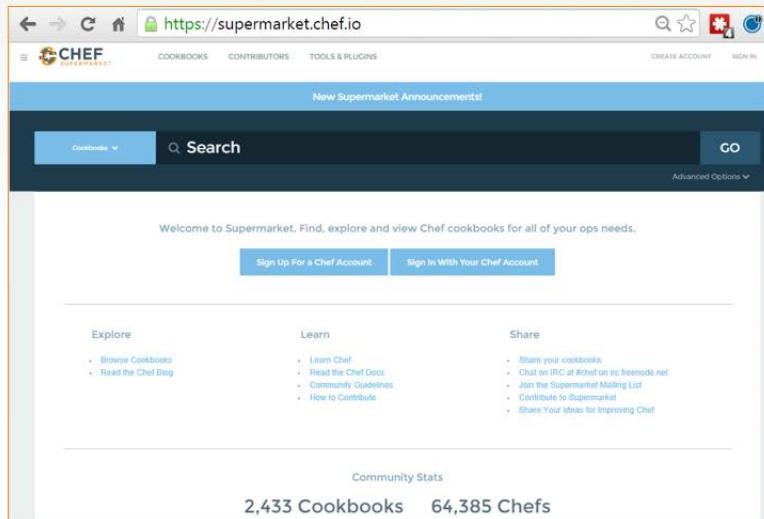
### **Objective:**

- Find a Cookbook on the Chef Supermarket to Manage a load balancer
- Configure the load balancer to send traffic to the iis\_web and apache\_web nodes
- Create myhaproxy Policyfile
- Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Let's find the haproxy (load balancer) cookbook within the community site to learn more about it.

# GL: Community Cookbooks

- ❖ Community cookbooks are managed by individuals.
- ❖ Chef does not verify or approve cookbooks in the Supermarket.
- ❖ Cookbooks may not work for various reasons.
- ❖ Still, there are real benefits to community cookbooks.



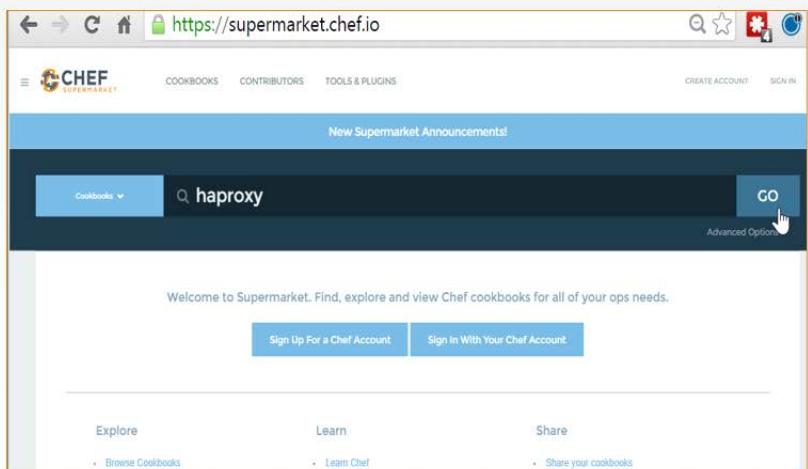
An important thing to remember is that on the community site are cookbooks managed by individuals. Chef does not verify or approve the cookbooks found in the Supermarket. These cookbooks solved problems for the original authors and then they decided to share them. This means that the cookbooks you find in the Supermarket may not be built or designed for your platform. It may not take into special consideration your needs and requirements. It may no longer be actively maintained.

Even if the cookbook does not work as a whole, there is still value in reading and understand the source code and extracting the pieces you need when creating your own. With all that said, there is a real benefit to the community site. When you find a cookbook that helps you deliver value quickly, it can be a tremendous boon to your productivity. This is what we are going to take advantage of with the haproxy cookbook.

# GL: Searching in the Supermarket

## STEPS

1. Visit [supermarket.chef.io](https://supermarket.chef.io)
2. Select the search field and type in haproxy in the search field. Then click the GO button.
3. Click the resulting [haproxy](#) link.



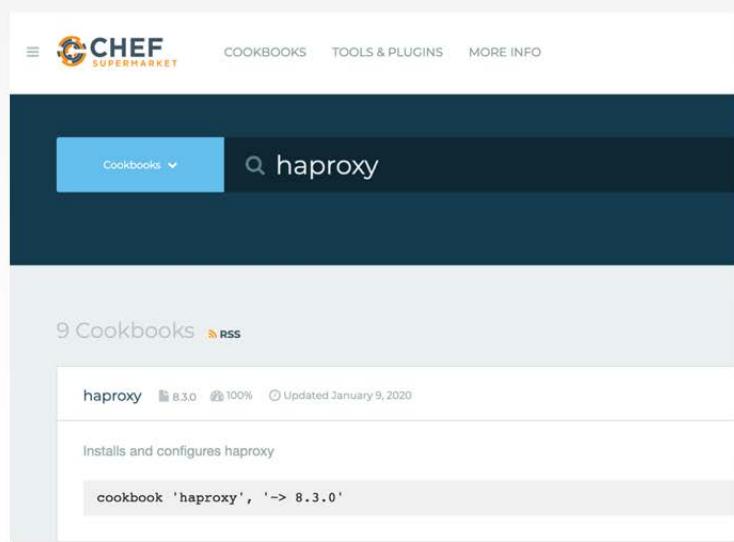
From the Supermarket main page type the search term "haproxy" and the click the GO button.

Below the search term will show us all the matching cookbooks. The haproxy cookbook is in that result set.

# GL: Searching in the Supermarket

## STEPS

1. Visit [supermarket.chef.io](https://supermarket.chef.io)
2. Select the search field and type in haproxy in the search field. Then click the GO button.
3. Click the resulting **haproxy** link.



Cookbooks usually map one-to-one to a piece of software and usually are named after the piece of software that they manage. Select the cookbook named haproxy from the search results.

# GL: Supermarket Cookbooks

On the left, we are presented with the various ways we can install the cookbook...

On the right side we can see the individuals that maintain the cookbook...

The screenshot shows the Chef Supermarket interface for the 'haproxy' cookbook. On the left, there's a sidebar with links like 'RSS', '(63) Versions 8.3.0', 'Follow 164', 'Berkshelf', 'Policyfile', 'Knife', and a search bar containing 'cookbook \'haproxy\', \'-> 8.3.0\''. Below this are tabs for 'README', 'Dependencies', 'Changelog', and 'Quality 100%'. The main content area has a title 'haproxy Cookbook' and a description 'Installs and configures HAProxy.'. It includes sections for 'build passing', 'cookbook v8.3.0', 'backers 8', 'sponsors 2', and 'License Apache 2.0'. On the right, there's a 'DETAILS' section with a 'View Source' and 'View Issues' button, updated on 'JANUARY 9, 2020'. It lists 'PLATFORMS' (Ubuntu, RHEL, CentOS, Debian, etc.) and 'LICENSE' (Apache 2.0). A 'Download Cookbook' button is at the bottom.

At this point you are presented with information that describes the cookbook. Starting on the right-hand side we see the individuals that maintain the cookbook, a link to view the source details, last updated date, supported platforms, licensing, and a link to download the cookbook.

On the left, we are presented with the various ways we can install the cookbook, the README that describes information about the cookbook, any cookbooks that this cookbook may depend on, a history of the changes, and its food critic rating--which is a code evaluator for best practices.

# GL: Supermarket Cookbooks

The area to focus most of your attention from the beginning is the README.

Reading and understanding the README at a glance is difficult. It is a skill that comes with time.

The screenshot shows the Chef Supermarket page for the 'haproxy' cookbook. At the top, there are tabs for 'README', 'Dependencies', 'Changelog', and 'Quality 100%'. Below the tabs, the title 'haproxy Cookbook' is displayed. Underneath the title, there are several status indicators: 'build passing', 'cookbook v8.3.0', 'backers 8', 'sponsors 2', 'License Apache 2.0'. A brief description follows: 'Installs and configures HAProxy.' Below this is the 'Maintainers' section, which states: 'This cookbook is maintained by the Sous Chefs. The Sous Chefs are a community of Chef cookbook maintainers working together to maintain important cookbooks. If you'd like to know more please visit [sous-chefs.org](#) or come chat with us on the Chef Community Slack in #sous-chefs.' The 'Requirements' section lists: '■ HAProxy stable or LTS ■ Chef 13.9+'. The 'Platforms' section lists supported platforms: '■ debian: 8 & 9 ■ ubuntu: 16.04 & 18.04 ■ centos: 7 ■ amazonlinux: 2'. The page footer contains the Chef logo.

©2020 Chef Software Inc.

13-13



The area to focus most of your attention from the beginning is the README. The README describes the various attributes that are defined within the cookbook and the purpose of the recipe. This is the same README file found in the cookbooks we currently have within our organization. This one, however, has had far more details added to give new users like us the ability to understand more quickly what the cookbook does and how it does it.

Reading and understanding the README at a glance is difficult. It is a skill that comes with time. For the haproxy cookbook, notice the version of the chef-client that is required to run the latest version of the cookbook. If you're not using the latest major version of the client you may need to investigate older versions of the cookbook.

# GL: Supermarket Cookbooks

The README defines a number of new Resources that can be used if this Cookbook is included in a node's run-list.

These Custom Resources are defined with a Cookbook's resources/directory. A well-written README will define the actions and properties a Custom Resource accepts.

## Common Resource Features

HAProxy has many configurable options available, this cookbook makes the most popular options available as resource properties.

If you wish to use a HAProxy property that is not listed the `extra_options` hash is available to take in any number of additional values.

For example, the ability to disable listeners is not provided out of the box. Further examples can be found in either `test/fixtures/recipes` or `spec/test/recipes`. If you have questions on how this works or would like to add more examples so it is easier to understand, please come talk to us on the [Chef Community Slack](#) on the `#sous-chefs` channel.

```
haproxy_listen 'disabled' do
 bind '0.0.0.0:1337'
 mode 'http'
 extra_options('disabled': '')
end
```

[https://docs.chef.io/custom\\_resources.html](https://docs.chef.io/custom_resources.html)

A custom resource functions just like any other resource, using a name, actions and properties.

```
custom_resource "name" do
 action :action_name
 property 'property_value'
end
```

Often, common examples of using the Custom Resources will be shown in the `text/fixtures/recipes` directory.

# GL: Supermarket Cookbooks

Further down in the README you'll see the full list of Custom Resources the haproxy Cookbooks provides.

We'll use of a number of these resources to easily configure an haproxy server and forward traffic to our web servers. This pattern provides an easy interface for consumers of the cookbook to change the way haproxy is deployed, without having to write all the logic yourself.

## Resources

- [haproxy\\_acl](#)
- [haproxy\\_backend](#)
- [haproxy\\_cache](#)
- [haproxy\\_config\\_defaults](#)
- [haproxy\\_config\\_global](#)
- [haproxy\\_fastcgi](#)
- [haproxy\\_frontend](#)
- [haproxy\\_install](#)
- [haproxy\\_listen](#)
- [haproxy\\_mailer](#)
- [haproxy\\_peer](#)
- [haproxy\\_resolver](#)
- [haproxy\\_service](#)
- [haproxy\\_use\\_backend](#)
- [haproxy\\_userlist](#)

You may use a limited set of resources to define a simple implementation, and then add more resources as your needs grow or change. We'll use just four of these resources in our recipe to define a load balancer, but the others exist to fine-tune your implementation.



## Using Community Cookbooks

Chef Community Cookbooks can be used as-is but in most cases you will want to use them as a foundation as you write your own.

Don't use forked community cookbooks in production, or you will miss out on upstream changes, and will have to rebase. Instead use **wrapper cookbooks**.

Because we want to be able to get upstream updates to these community cookbooks, rather than simply forking them we will create another wrapper cookbook.

# GL: Supermarket Cookbooks

**Reminder:** A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook.

It can define new default values for the recipes.

## Wrapper Cookbook

haproxy  
Cookbook  
(Attributes)

(New additional attributes)

<https://docs.chef.io/supermarket.html#wrapper-cookbooks>

<https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/>

A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook but allows us to define new default values for the recipes.

This is a common method for overriding cookbooks because it allows us to leave the original cookbook untouched. We simply provide new default values that we want and then include the recipes that we want to run.

Let's generate our wrapper cookbook named myhaproxy. Traditionally we would name the cookbook with a prefix of the name of our company and then follow it by the cookbook name 'company-cookbook'.

<https://docs.chef.io/supermarket.html#wrapper-cookbooks>

<https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/>



## Group Lab: Load Balancer

*Adding a load balancer will allow us to better grow our infrastructure.*

### **Objective:**

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- Configure the load balancer to send traffic to the iis\_web and apache\_web nodes
- Create myhaproxy Policyfile
- Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Let's find the haproxy (load balancer) cookbook within the community site to learn more about it.

Instructor Note:

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

Because the student must provide the IP and hostname of their own web server, this must be manually added if they download a copy of the 'myhaproxy' cookbook.

## GL: Returning to the Chef Repository Directory



```
$ cd ~/chef-repo
```

Change to your chef-repo directory

Instructor Note: The generate cookbook command needs the path in the name of the cookbook here to ensure that it generates the cookbook in the cookbooks directory. Without it the cookbook will be generated within the root of the chef repository. If that happens simply have the learner move the newly generated cookbook into the cookbooks directory.

# GL: Generating a New Cookbook



```
$ chef generate cookbook cookbooks/myhaproxy
```

```
Generating cookbook myhaproxy
- Ensuring correct cookbook content
- Committing cookbook files to git
```

```
Your cookbook is ready. To setup the pipeline, type `cd cookbooks/myhaproxy`, then
run `delivery init`
```

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: Creating a Dependency in the Cookbook

```
~/chef-repo/cookbooks/myhaproxy/metadata.rb

name 'myhaproxy'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures myhaproxy'
long_description 'Installs/Configures myhaproxy'
version '0.1.0'
chef_version '>= 14.0
depends 'haproxy', '~> 8.3.0'
```

Set up a dependency within your haproxy cookbook. Establishing this dependency informs the Chef Server that whenever you deliver this cookbook to a node, you should also deliver with it the mentioned dependent cookbooks.

This is important because your cookbook is simply going to set up new default values and then execute the recipes defined in the original cookbook.



## Custom Resources

A recipe can call any recipes or custom resources from a dependency that's defined with 'depends' in the metadata.rb file.

You can call a Custom Resource from any recipe in your wrapper cookbook.

We want to use our custom resources from the haproxy community cookbook within our myhaproxy cookbook. Now that we've added a dependency, we can do this within any recipe in our wrapper cookbook.

## GL: Include the haproxy's manual recipe in default recipe

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

#
Cookbook Name:: myhaproxy
Recipe:: default
#
Copyright (c) 2020 The Authors, All Rights Reserved.

haproxy_install 'package'

haproxy_frontend 'http-in' do
 bind '*:80'
 default_backend 'servers'
end
...
```

First, within the myhaproxy cookbook's default recipe we will use the `haproxy_install` resource to install haproxy from a package (as opposed to from source).

Next, we use the `haproxy_frontend` resource, giving it a generic name like "http-in". This will be the proxy server that receives our web traffic. Using the "bind" property we can tell it to push it's backend to port 80. We also define the name of our backend with the "default\_backend" property, which we'll name "servers" for now. We will continue writing this recipe later in this module, but first we need to define the backend server ip's we want to forward to our frontend.

## GL: Viewing Help on the Node Show Subcommand



```
$ knife node show --help
```

```
knife node show NODE (options)
 -a ATTR1 [--attribute ATTR2] , Show one or more attributes
 --attribute

 -s, --server-url URL Chef Server URL
 --chef-zero-host HOST Host to start chef-zero on
 --chef-zero-port PORT Port (or port range) to start chef-zero on.

Port ranges
 -k, --key KEY API Client Key
 --[no-]color Use colored output, defaults to false on
Windows, true
 -c, --config CONFIG The configuration file to use
 --defaults Accept default values for all questions
 -d, --disable-editing Do not open EDITOR, just accept the data as is
```

This new default value for the haproxy members needs to define the information about the webserver node. So you need to capture the node's public host name and public IP address.

The 'knife node show' command will display information about the node. You can ask to see a specific attribute on a node with the -a flag or the --attribute flag.

## Demo: Viewing the Node's IP Address



```
$ knife node show iis_web -a ipaddress
```

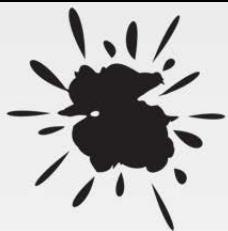
```
iis_web:
ipaddress: 172.31.8.68
```

This method of retrieving the IP address is not useful if you need the external IP address. We'll show you another way in a moment.

You can display the IP address of `iis_web` with the '`-a`' flag and specifying the attribute '`ipaddress`'.

With cloud providers that generate machines for you often assign internal IP addresses, those values may not work properly.

Instructor Note: The IP addresses of the nodes that were used during the creation of this training were based on Amazon Web Services (AWS). The address reported by Ohai is often the private, internal address.



## Amazon EC2 Instances

The IP address and host name are unfortunately not how we can address these nodes within our recipes.

The reason you may need to ask the node for a different set of attributes is that we are using Amazon as a cloud provider for our instances. These instances are displaying the internal IP address when we ask for the `ipaddress` attribute.

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.

## GL: Viewing the Node's Cloud Details



```
$ knife node show iis_web -a cloud
```

```
iis_web:
 cloud:
 local_hostname: ip-172-31-8-68.ec2.internal
 local_ipv4: 172.31.8.68
 private_ips: 172.31.8.68
 provider: ec2
 public_hostname: ec2-54-175-46-24.compute-1.amazonaws.com
 public_ips: 54.175.46.24
 public_ipv4: 54.175.46.24
```

You'll need this information for the next task.

If you use 'knife node show' to display the 'cloud' attribute for iis\_web, you will see the local, private, and public connection information.

Capture and write down the public hostname and the public ipv4 address of iis\_web. You will need this in the recipe you are going to write.

## GL: Viewing the Node's Cloud Details



```
$ knife node show apache_web -a cloud
```

```
apache_web:
```

```
 cloud:
```

```
 local_hostname: ip-172-31-57-169.ec2.internal
```

```
 local_ipv4: 172.31.57.169
```

```
 private_ips: 172.31.57.169
```

```
 provider: ec2
```

```
 public_hostname: ec2-34-196-10-17.compute-1.amazonaws.com
```

```
 public_ips: 34.196.10.17
```

```
 public_ipv4: 34.196.10.17
```

You'll need this information for the next task.

If you use 'knife node show' to display the 'cloud' attribute for apache\_web, you will see the local, private, and public connection information.

Capture and write down the public hostname and the public ipv4 address of apache\_web. You will need this in the recipe you are going to write.

## GL: Inserting Real Node Data into the Attributes

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

```
...
haproxy_backend 'servers' do
 server [
 'ec2-54-175-46-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
 'ec2-34-196-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
]
end

haproxy_service 'haproxy'
```

Replace the hostname value and IP address values with your `iis_web` and `apache_web` node's public host name and public IP address.

Finally, we complete the default recipe by declaring the `haproxy_backend` custom resource. It accepts a property called "server", which we define as an array of webserver addresses.

Each address follows the pattern:

'HOSTNAME PUBLIC\_IP\_ADDRESS:80 maxconn 32'

Make sure there's a comma after the first server (in the slide this is `iis_web`)

Lastly, we add the `haproxy_service` resource, which will automatically accept notifications from other services. Although the default action will be "nothing", we will see it automatically restarting the service when changes to the `haproxy_backend` are made due to the way the custom resource was implemented. This resource needs to be in our resource collection so it can receive notifications from the other custom resources.

Although the notifications aren't directly written into this recipe, you can learn more about notifications here:

[https://docs.chef.io/resource\\_common.html#notifications](https://docs.chef.io/resource_common.html#notifications)

# GL: Viewing the Complete Recipe

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

haproxy_install 'package'

haproxy_frontend 'http-in' do
 bind '*:80'
 default_backend 'servers'
end

haproxy_backend 'servers' do
 server [
 'ec2-54-175-46-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
 'ec2-34-196-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
]
end

haproxy_service 'haproxy'
```

The final default recipe for the wrapper cookbook 'myhaproxy' looks like the above but with the hostname value and the ipaddress value of your [iis\\_web](#) node.

The haproxy action of :reload will run every time chef-client executes, in case the backend list of servers is changed. This isn't ideal, and in a later module we'll use a notification to make this better.

Save your recipe file.



## Group Lab: Load Balancer

*Adding a load balancer will allow us to better grow our infrastructure.*

### **Objective:**

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the iis\_web and apache\_web nodes
- Create myhaproxy Policyfile
- Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

Let's find the haproxy (load balancer) cookbook within the community site to learn more about it.

Instructor Note:

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

Because the student must provide the IP and hostname of their own web server, this must be manually added if they download a copy of the 'myhaproxy' cookbook.

# CONCEPT



## Policyfile.rb and the Policyfile.lock.json

Now that we have our myhaproxy cookbook in our chef-repo, we can create our Policyfile.rb and then generate our Policyfile.lock.json as we discussed in previous modules.

This time we'll name our Policyfile **myhaproxy**.

## GL: Generate the Policyfile and Name it myhaproxy



```
> cd ~/chef-repo
> chef generate policyfile policyfiles/myhaproxy
```

```
Recipe: code_generator::policyfile
* template[/Users/sdelfante/chef-repo/policyfiles/myhaproxy.rb] action create
 - create new file /Users/sdelfante/chef-repo/policyfiles/myhaproxy.rb
 - update content in file /Users/sdelfante/chef-repo/policyfiles/myhaproxy.rb
from none to 2ae2aa
(diff output suppressed by config)
```

## GL: Verify that the Policyfile Exists



```
>ls policyfiles/ (or dir for Windows)
```

```
company_web.rb
```

```
company_web.lock.json
```

```
myhaproxy.rb
```

## GL: Edit the New myhaproxy.rb Policyfile

~/.chef-repo/policyfiles/myhaproxy.rb

```
#...skipping for brevity...
https://docs.chef.io/policyfile.html
A name that describes what the system you're building with Chef does.
name 'myhaproxy'

Where to find external cookbook
default_source :supermarket

run_list: chef-client will run these recipes in the order specified.
run_list 'myhaproxy::default'

Specify a custom source for a single cookbook:
cookbook 'myhaproxy', path: '../cookbooks/myhaproxy'
```

Replace the contents of the myhaproxy.rb below the `#https://docs.chef.io/policyfile.html` line with the code in green.

Notice how we need to specify the path to all dependent cookbooks too (myiis and apache).

```
Policyfile.rb - Describe how you want Chef Infra Client to build your system.
#
For more information on the Policyfile feature, visit...
#
A name that describes what the system you're building with Chef does.
name 'myhaproxy'

Where to find external cookbooks:
default_source :supermarket

run_list: chef-client will run these recipes in the order specified.
run_list 'myhaproxy::default'

Specify a custom source for a single cookbook:
cookbook 'myhaproxy', path: '../cookbooks/myhaproxy'
```



## Group Lab: Load Balancer

*Adding a load balancer will allow us to better grow our infrastructure.*

### **Objective:**

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the iis\_web and apache\_web nodes
- ✓ Create myhaproxy Policyfile
  - ❑ Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
  - ❑ Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

## GL: Generate the myhaproxy.lock.json



```
~/chef-repo> chef install policyfiles/myhaproxy.rb
```

```
Building policy myhaproxy
Expanded run list: recipe[myhaproxy::default]
Caching Cookbooks...
Installing myhaproxy >= 0.0.0 from path
Using haproxy 8.3.0
Using build-essential 8.2.1
Using yum-epel 3.3.0
Using seven_zip 3.1.2
Using mingw 2.1.0
Using windows 6.0.1

Lockfile written to /Users/robin/chef-repo/policyfiles/myhaproxy.lock.json
Policy revision id: 6e8a60de56e67ff84a7b7d8468c8ae63effd2f4d72afcc3480295954a24e0cdc
```

## GL: Verify that the myhaproxy.lock.json Exists



```
> ls policyfiles/ (or dir for Windows)
```

```
company_web.rb company_web.lock.json myhaproxy.rb
myhaproxy.lock.json
```

## GL: Push the myhaproxy.lock.json to Chef Infra Server



```
~/chef-repo> chef push prod policyfiles/myhaproxy.lock.json
```

```
Uploading policy myhaproxy (6e8a60de56) to policy group prod
Using build-essential 8.2.1 (4b9d5c72)
Using haproxy 8.3.0 (1a4f7607)
Using mingw 2.1.0 (9f5d572c)
Using seven_zip 3.1.2 (0e1fed3b)
Using windows 6.0.1 (042f3380)
Using yum-epel 3.3.0 (187c02d6)
Uploaded myhaproxy 0.1.0 (520e62bf)
```

## GL: Verify the myhaproxy Policy is on Chef Infra Server



```
~/chef-repo> chef show-policy
```

```
company_web
=====

* prod: 55529dbd15

myhaproxy
=====

* prod: e46185fa40
```

Here we can see that the **myhaproxy** policy has been uploaded to Chef Infra Server and is in the **prod** policy group.

Also notice the policy name that was derived from the contents of the **myhaproxy.lock.json**.



## Group Lab: Load Balancer

*Adding a load balancer will allow us to better grow our infrastructure.*

### **Objective:**

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the iis\_web and apache\_web nodes
- ✓ Create myhaproxy Policyfile
- ✓ Upload myhaproxy Policyfile.lock to the Chef Infra Server (uploads cookbooks)
- Bootstrap a new node that runs the myhaproxy (load balancer) cookbook

## GL: Bootstrap a New Linux Node



```
$ knife bootstrap IPADDRESS -U USER -P PWD --sudo -N lb
--policy-name myhaproxy --policy-group prod
```

```
[34.196.50.77] Starting Chef Infra Client, version 15.7.32
[34.196.50.77] Using policy 'myhaproxy' at revision
'ff07bcb7f5f57ac1cd6c2eb3f7c7785f8d5312c035725e3c54fb0d84292c6670'
[34.196.50.77] resolving cookbooks for run list: ["myhaproxy::default@1.0.0 (60)"]
[34.196.50.77] Synchronizing Cookbooks:
[34.196.50.77]
[34.196.50.77] - build-essential (8.2.1)
[34.196.50.77] - haproxy (8.3.0)
[34.196.50.77] - mingw (2.1.0)
[34.196.50.77] - myhaproxy (0.1.0)
[34.196.50.77] - seven_zip (3.1.2)
[34.196.50.77] - yum-epel (3.3.0)
[34.196.50.77] - windows (6.0.1)
...
Running handlers:
[34.196.50.77]
[34.196.50.77] Running handlers complete [34.196.50.77] Chef Infra Client finished, 16/29
resources updated in 29 seconds
```

node name

policy\_name

policy\_group

We are bootstrapping a new fresh Linux node here.

If you get the following prompt, type Y.

Connecting to 54.146.182.136

The authenticity of host '54.146.182.136 ()' can't be established.  
fingerprint is SHA256:q7AVSJ3Ai6fNFGGr8u/uwnUGOMP1MZo7QQrG8KwLSUml.

Are you sure you want to continue connecting  
? (Y/N)

## GL: Validate the Run List Has Been Set

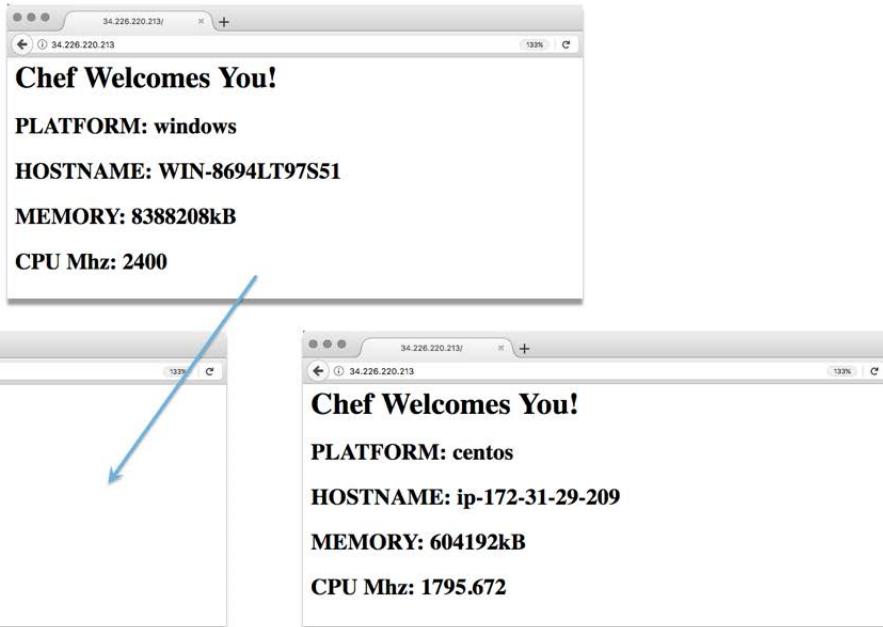


```
$ knife node show lb
```

```
Node Name: lb
Policy Name: myhaproxy
Policy Group: prod
FQDN: ip-172-31-22-163.ec2.internal
IP: 34.196.50.77
Run List: recipe[myhaproxy::default]
Recipes: myhaproxy::default, yum-epel::default
Platform: centos 7.6.1810
Tags:
```

Ensure the run list has been set correctly for lb.

# Lab: Test the Load Balancer



©2020 Chef Software Inc.

13-44



Point a web browser to the URL of your load balancer/proxy server and then click Refresh a few times. You should see each of your web server's HTML page as the Proxy server switches between each web server.

Note: The haproxy shared cache could take a few minutes before it refreshes, so you may not see the "Chef Welcomes You!" page update immediately. You may need to wait 20 or more seconds between refreshes.

To confirm haproxy is configured correctly, you could check the content of /etc/haproxy/haproxy.cfg on the load balancer node and confirm the servers are included in the pool. If you are curious, you could configure caching using the haproxy\_cache custom resource: [https://github.com/sous-chefs/haproxy/blob/master/documentation/haproxy\\_cache.md](https://github.com/sous-chefs/haproxy/blob/master/documentation/haproxy_cache.md)

# Lab: Test the Load Balancer



Point a web browser to the URL of your load balancer/proxy server and then click Refresh a few times. You should see each of your web server's HTML page as the Proxy server switches between each web server.



## GL: Load Balancer

*Adding a load balancer will allow us to better grow our infrastructure.*

**Objective:**

- ✓ Find a Cookbook on the Chef Supermarket to Manage a load balancer
- ✓ Configure the load balancer to send traffic to the iis\_web node
- ✓ Create Policyfile and lock.
- ✓ Upload Policyfile.lock to the Chef server
- ✓ Bootstrap a new node that runs the haproxy (load balancer) cookbook
- ✓ Converge the node



## Review Questions

1. What are the benefits of the Chef Super Market? And what are the drawbacks?
2. Why do you use a wrapper cookbook?
3. When might you decide to not wrap the cookbook?

1. Benefits: You can use existing cookbooks instead of writing all of them.  
Drawbacks: The cookbooks you find in the Supermarket may not be built or designed for your platform. They may not take into special consideration your needs and requirements or they may no longer be actively maintained.
2. When you want to use a community cookbook but you need to override some attributes or functionality in a wrapper instead of modifying the community cookbook.
3. Possible answers: Whenever you're using a cookbook that isn't directly owned by your team because it means you have no control over changes. Whenever you're using a cookbook that might act differently in different scenarios or setups.



## Q&A

What questions can we help you answer?

- Chef Supermarket
- Wrapper Cookbooks
- Node Attributes
- knife ssh

What questions can we help you answer?

In general or about specifically about Chef Super Market, wrapper cookbooks, node attributes, the 'knife ssh' command.



**CHEF**<sup>TM</sup>

---

©2020 Chef Software Inc.

## Using policy\_name to Define a Role for Nodes

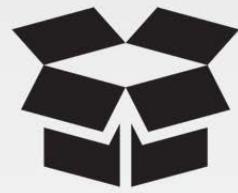
Using policy\_name instead of role objects

# Objectives

After completing this brief module, you should be able to

- Explain how policy\_name replaces the legacy Role object
- Use knife search to display nodes

In this brief module you will ensure your web nodes have the same policy\_name to better describe them and so you can configure them in a similar manner.



## policy\_name

Chef users sometimes used **role** objects to describe a run list of recipes that are executed on the node and to use for node searches.

Nowadays we use the `policy_name` for those purposes.

For example, all nodes that possess the **company\_web** policy name would be configured in a similar or identical manner.

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



## policy\_name

When you assign a common policy\_name to a group of nodes, each node will receive the same cookbooks.

When these nodes perform a Chef Client run, they utilize recipes specified in the Policyfile run list.

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



## GL: Verify that All Web Nodes Use the Same policy\_name

*Give your nodes a policy\_name to better describe them and so we can configure them in a similar manner.*

### Objective:

- Confirm our iis\_web node has the same policy\_name (company\_web) as the apache\_web node
- use `knife search` to list all or specific nodes

This is particularly powerful because we will no longer have to manage each of these nodes individually. Instead we can make changes to the policy file named **company\_web** (and its cookbooks) and all of the nodes that have this policy file named **company\_web** will update accordingly.

## company\_web Policy Name



In a previous module we set the `iis_web` node to use the `company_web` policy name with this command:

```
knife node policy set iis_web prod company_web
```

Now it is possible to update the cookbooks that are associated to the `company_web` policy name and nodes with the `company_web` policy name will be configured the same.

We will also be able to easily search for all nodes that have the `company_web` policy name.

**Note:** In the next module you will learn more about using search.

## GL: List Your Nodes



```
$ knife node list
```

```
apache_web
iis_web
lb
```

## GL: Searching for All Nodes with knife



```
$ knife search node "*:*"
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-57-169.ec2.internal
IP: 34.196.104.17
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 34.195.38.226
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:

Node Name: lb
Policy Name: myhaproxy
Policy Group: prod
FQDN: ip-172-31-22-163.ec2.internal
IP: 34.196.50.77
Run List: recipe[myhaproxy::default] ...
```

Here we can see for each node the policy\_name value, and the policy\_group each is assigned to.

## GL: Searching for Nodes With a Specific policy\_name



```
$ knife search node policy_name:company_web
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-57-169.ec2.internal
IP: 34.196.104.17
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 34.195.38.226
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
```



## GL: Verify that All Web Nodes Use the Same policy\_name

*We will give our nodes a role to better describe them and so we can configure them in a similar manner.*

### Objective:

- ✓ Confirm our iis\_web node has the same policy\_name (company\_web) as the apache\_web node
- ✓ Use `knife search` to list all or specific nodes

## company\_web Policy Name



In the next module you will learn more about using search and you will put the company\_web policy name to work for you.

## Q&A

What questions can we help you answer?





**CHEF**<sup>TM</sup>

---

©2020 Chef Software Inc.

# Search

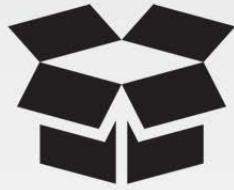
Update a Cookbook to Dynamically Use Nodes with the company\_web policy name

# Objectives

After completing this module, you should be able to

- Describe the query syntax used in search
- Build a search into your recipe code
- Create a Ruby String and Array dynamically
- Test that your load balancer is still balancing traffic

In this module you will learn how to describe the query syntax used in search, build a search into your recipe code, create a ruby array and ruby hash, and update the myhaproxy wrapper cookbook to dynamically use nodes with the company\_web policy.



## Search

To add new servers as load balancer members, we would need to bootstrap a new web server and then update our load balancer's myhaproxy cookbook recipe.

That seems inefficient to have to update a cookbook recipe.

To add new servers as load balancer members, we would need to bootstrap a new web server and then update our myhaproxy cookbook to include that new web server. But that seems dramatically inefficient to have to update a cookbook recipe.

A more ideal solution would be for the recipe to instead discover all of the web servers within our organization and automatically add them to a list of available members for our load balancer.

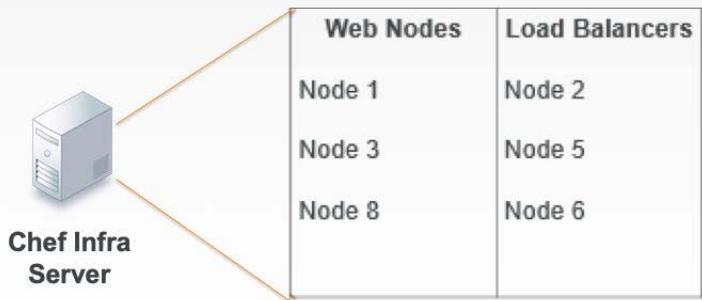
# The Chef Infra Server and Search

Chef Infra Server maintains a representation of all the nodes within our infrastructure that can be searched on.

Search is a service discovery tool that allows us to query the Chef Infra Server.

[https://docs.chef.io/chef\\_search.html](https://docs.chef.io/chef_search.html)

[https://docs.chef.io/chef\\_search.html#search-indexes](https://docs.chef.io/chef_search.html#search-indexes)

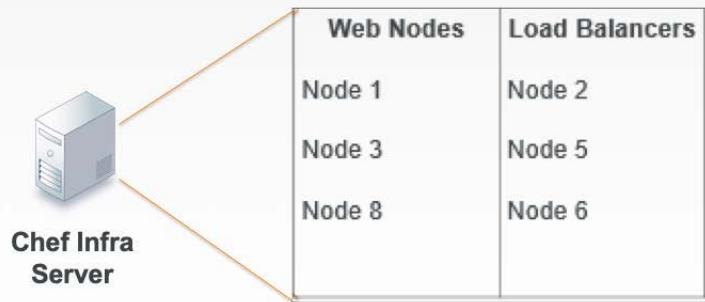


The Chef Infra Server maintains a representation of all the nodes within an infrastructure and provides a way for us to discover these systems through Search.

Search is a service discovery tool that allows us to query the Chef Infra Server across a few indexes. One such index is on our nodes.

## The Chef Infra Server and Search

We can ask the Chef Infra Server to return all the nodes or a subset of nodes based on the query syntax that we provide it through `knife search` or within our recipes through `search`.



We can ask the Chef Infra Server to return back to us all the nodes or a subset of nodes based on the query syntax that we provide it through the knife command `knife search` or within our recipes through the `search` method.

# Search Syntax

A search query is comprised of two parts: the key and the search pattern. A search query has the following syntax:

**key:search\_pattern**

...where key is a field name that is found in the JSON description of an indexable object on the Chef Infra Server and search\_pattern defines what will be searched for,

A search query is comprised of two parts: the key and the search pattern. A search query has the following syntax:

**key:search\_pattern**

...where key is a field name that is found in the JSON description of an indexable object on the Chef Infra Server (a role, node, client, environment, or data bag) and search\_pattern defines what will be searched for.

# Search Criteria

We may use wildcards within search so a search criteria that we could use is: `"*:*"`

However, querying and returning every node is not what we need to solve our current problem.



Scenario: We want only to return a subset of our nodes... only the nodes that are web servers.

Querying and returning every node is not exactly what we need to solve our current problem. Scenario: We want only to return a subset of our nodes--only the nodes that are webservers.

Let's examine the search criteria more so we can understand how it works and how we can use it to find a subset of the nodes--only the nodes that are webservers.

## Demo: View Information for All Nodes



```
> knife search node "*:*"
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-57-169.ec2.internal
IP: 34.196.104.17
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFOCUFHDCP.ec2.internal
IP: 34.195.38.226
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:

Node Name: lb
Policy Name: myhaproxy
Policy Group: prod
FQDN: ip-172-31-22-163.ec2.internal
IP: 34.196.50.77
Run List: recipe[myhaproxy::default]
Recipes: myhaproxy::default, haproxy::manual, haproxy::install_package
Platform: centos 7.6.1810
```

knife search node "\*:\*" will return see all nodes.

However, querying and returning every node is not exactly what we need to solve our current problem. In our scenario we want only to return a subset of our nodes (only the nodes that are webservers) and also we only want the IP address of those nodes, not any the other attributes returned.

Let's examine the search criteria more so we can understand how it works and how we can use it to find a subset of the nodes, and only the info we want for those nodes.

## Demo: View Information for All Server Nodes



```
> knife search node policy_name:company_web
```

```
Node Name: apache_web
Policy Name: company_web
Policy Group: prod
FQDN: ip-172-31-57-169.ec2.internal
IP: 34.196.104.17
Run List: recipe[company_web::default]
Recipes: company_web::default, apache::default, apache::server
Platform: centos 7.6.1810
Tags:

Node Name: iis_web
Policy Name: company_web
Policy Group: prod
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 34.195.38.226
Run List: recipe[company_web::default]
Recipes: company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```

We can query the Chef Infra Server for specific nodes such as those with the assigned `policy_name` of `company_web`.

## Demo: Return Public Hostname for Servers



```
> knife search node policy_name:company_web -a cloud.public_hostname
```

```
2 items found
```

```
apache_web:
 cloud.public_hostname: ec2-34-196-104-17.compute-1.amazonaws.com

iis_web:
 cloud.public_hostname: ec2-34-195-38-226.compute-1.amazonaws.com
```

The ‘-a’ flag allows you to specify a particular attribute from those nodes.

## Demo: Return Public Hostname and IP for Servers



```
> knife search node policy_name:company_web -a cloud
```

```
apache_web:
 cloud:
 local_hostname: ip-172-31-57-169.ec2.internal
 local_ipv4: 172.31.57.169
 local_ipv4_addrs: 172.31.57.169
 provider: ec2
 public_hostname: ec2-34-196-104-17.compute-1.amazonaws.com
 public_ipv4: 34.196.104.17
 public_ipv4_addrs: 34.196.104.17

iis_web:
 cloud:
 local_hostname: ip-172-31-62-51.ec2.internal
 local_ipv4: 172.31.62.51
 local_ipv4_addrs: 172.31.62.51
 provider: ec2
 public_hostname: ec2-34-195-38-226.compute-1.amazonaws.com
 public_ipv4: 34.195.38.226
 public_ipv4_addrs: 34.195.38.226
```

The '-a' flag allows you to specify a particular attribute from those nodes.

# Search Syntax within a Recipe

```
web_nodes = search('node', 'policy_name:company_web')
```

creates and names a variable

assigns the value of the operation on the right into the variable on the left

the index or items to search

the search criteria - key:value

invokes the search method

The search syntax within a recipe differs from the search syntax when using `knife search` from the command line.

```
web_nodes = search('node','policy_name:company_web')
```

Search within a recipe is done through a `search` method that is available within the recipe.

The `search` method accepts two arguments. The first argument is a string or variable that contains the index or item to search on the Chef Infra Server. These are: nodes; policy\_name; and policy\_group (a.k.a. environments). The second argument is a string or variable that contains the search criteria to scope the results. This is using the notation 'key:value'.

The result of the search method is stored in a local variable that is named 'web\_nodes'. Variables within Ruby are created immediately when you assign them.

## Search Syntax within a Recipe

```
web_nodes = search('node', 'policy_name:company_web')
```

Search the Chef Infra Server for all node objects that have the policy\_name equal to 'company\_web' and store the results into a local variable named "web\_nodes".

This example syntax could be translated to mean: Search the Chef Infra Server for all node objects that have the role equal to 'web\_server' and store the results into a local variable named 'web\_nodes'.

## Hard Coding Example

```
haproxy_install 'package'

haproxy_frontend 'http-in' do
 bind '*:80'
 default_backend 'servers'
end

haproxy_backend 'servers' do
 server [
 'ec2-54-175-46-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
 'ec2-34-196-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
]
end

haproxy_service 'haproxy'
```

Previously, we had been hard coding the hostname and ipaddress values in our wrapped myhaproxy recipe. We can request these values from the Chef Infra Server through the `knife node show` command. The hostname and ipaddress values are captured by Ohai and sent to the Chef Infra Server. On the Chef Infra Server we can query those values when we ask about a specific attribute about the node. We do that by providing the `-a` flag with the name of the attribute. Because the nodes that we manage are hosted in the cloud, these attributes are stored under a parent attribute named 'cloud'.



## GL: Dynamic Web Load Balancer

*Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!*

### Objective:

- Update the myhaproxy cookbook to dynamically use nodes with the company\_web policy\_name.
- Update the major version of the myhaproxy cookbook
- Upload the Cookbook
- Run chef-client on the load balancer node
- Verify the load balancer node relays requests to both web nodes

In this section we'll update the load balancer's myhaproxy cookbook to dynamically use nodes with the company\_web policy.

## GL: Remove the Hard-Coded Members

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

haproxy_install 'package'

haproxy_frontend 'http-in' do
 bind '*:80'
 default_backend 'servers'
end

haproxy_backend 'servers' do
 server [
 'ec2-54-175-46-24.compute-1.amazonaws.com 54.175.46.24:80 maxconn 32',
 'ec2-34-196-10-17.compute-1.amazonaws.com 34.196.10.17:80 maxconn 32'
]
end

haproxy_service 'haproxy'
```

Edit the 'myhaproxy' cookbook's default recipe and remove the current default recipe where you hard-coded the members.

## GL: Use Search to Find the Web Servers

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

```
...
web_nodes = search('node','policy_name:company_web')

haproxy_service 'haproxy'
```

**Note:** We will provide the final recipe in a moment.

Replace it with an updated recipe that searches for all nodes that have the company\_web policy name defined.

The search method's first parameter is asking the Chef Infra Server to look at all the nodes within our organization.

The search method's second parameter is asking the Chef Infra Server to only return the nodes that have been assigned the role web\_server.

All of those nodes are stored in a local variable named `all\_web\_nodes`. This is an array of node objects. It may contain zero or more nodes that match the search criteria.

## GL: Create an Array to Store the Converted Members

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

```
...
web_nodes = search('node', 'policy_name:company_web')

servers = []

TODO: Convert all found nodes into an array with format:
'PUBLIC_HOSTNAME PUBLIC_IP_ADDRESS:80 maxconn 32'

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy'
```

Unfortunately we cannot simply assign our array of web nodes into the haproxy's members attributes because it needs a string that contains the webserver's "PUBLIC\_HOSTNAME" and "PUBLIC\_IPADDRESS". We will need to convert each of the web node objects into a structure that the haproxy\_backend custom resource expects.

First we create an empty array and assign that empty array into a local variable named `servers`. `servers` is an array that we will populated with the hashes we will create later; until then we will write a TODO for us. Then we will pass that array to the "server" property of the haproxy\_backend resource.

## GL: Create an Array to Store the Converted Members

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

...
web_nodes = search('node', 'policy_name:company_web')

servers = []

web_nodes.each do |web_node|
 server = ""
 # TODO: Populate the array with each webserver's hostname and ipaddress
end

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy'
```

So we need to loop through the array of all the web nodes stored in `web\_nodes`. We do that through a method available on every array object named 'each'. With the each method a block of code is provided -- you see it here from the first 'do' right after the each to the 'end' later in the file.

A block of code is an operation that you want perform on every item in the array. In our case we want to take each of the node objects and grab the public hostname and ipaddress returned from the search. This information comes from the node objects stored on the Chef Infra Server.

So every member of the array is visited and every member of the array runs through the block of code.

## GL: Create an Array to Store the Converted Members

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

...
web_nodes = search('node', 'policy_name:company_web')

servers = []

web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']}:#{
 web_node['cloud']['public_ipv4']}:80 maxconn 32"
 servers.push(server)
end

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy'
```

Between the pipes we see a local variable that we are defining that exists only in the block `web\_node`. This local variable, `web\_node`, is a name we came up with to refer to each node in our array of `web\_nodes`. When inside the block of code `web\_nodes.each` (this is called a "loop") it is referred to as `web\_node`. Inside the block the first thing that is created is another local variable named "server" which is assigned a string that contains the webserver's public hostname and ipaddress. Then the local variable "server" is pushed into the servers array. This adds the `web\_node` to the end of the array. When we are done looping through every web node the servers array contains a list of all these string values, one for each `web\_node` returned from the search.

# GL: The Final Recipe

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

```
haproxy_install 'package'

haproxy_frontend 'http-in' do
 bind '*:80'
 default_backend 'servers'
end

web_nodes = search('node', 'policy_name:company_web')

servers = []

web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']} #{web_node['cloud']['public_ipv4']}:80 maxconn 32"
 servers.push(server)
end

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy'
```

```
haproxy_install 'package'
```

```
haproxy_frontend 'http-in' do
 bind '*:80'
 default_backend 'servers'
end
```

```
web_nodes = search('node', 'policy_name:company_web')
```

```
servers = []
```

```
web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']}"
 "#{web_node['cloud']['public_ipv4']}:80 maxconn 32"
 servers.push(server)
end
```

```
haproxy_backend 'servers' do
 server servers
```

```
end
```

```
haproxy_service 'haproxy'
```



## GL: Dynamic Web Load Balancer

*Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!*

### Objective:

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company\_web policy\_name.
  - ❑ Update the major version of the myhaproxy cookbook
  - ❑ Upload the Cookbook
  - ❑ Run chef-client on the load balancer node
  - ❑ Verify the load balancer node relays requests to both web nodes

## GL: Updating the Cookbook's Version Number

```
~/chef-repo/cookbooks/myhaproxy/metadata.rb

name 'myhaproxy'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures myhaproxy'
long_description 'Installs/Configures myhaproxy'
version '1.0.0'
chef_version '>= 14.0' if respond_to?(:chef_version)

depends 'haproxy', '~> 8.3.0'
```

First we update the version to the next major release. We set the version number to 1.0.0.



## Dynamic Web Load Balancer

*Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!*

### Objective:

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company\_web policy\_name.
- ✓ Update the major version of the myhaproxy cookbook
- Update and push the Policyfile
- Run chef-client on the load balancer node
- Verify the load balancer node relays requests to both web nodes

## GL: Ensure You are in the chef-repo



```
$ cd ~/chef-repo
```

## GL: Update the Policyfile



```
$ chef update policyfiles/myhaproxy.rb
```

```
Attributes already up to date
Building policy myhaproxy
Expanded run list: recipe[myhaproxy::default]
Caching Cookbooks...
Installing myhaproxy >= 0.0.0 from path
Using haproxy 8.3.0
Using build-essential 8.2.1
Using yum-epel 3.3.0
Using seven_zip 3.1.2
Using mingw 2.1.0
Using windows 6.0.1

Lockfile written to /Users/sdelfante/chef-repo/policyfiles/myhaproxy.lock.json
Policy revision id:
52a2244578f86c6b4066c7bf087292e514a819b3466d6fa3387a462858a038b4
```

If you ever need to update and re-push your policyfile, use the `chef update` command. For example:

```
chef update myhaproxy.rb
```

```
chef push prod myhaproxy.lock.json
```

## GL: Push the myhaproxy.lock.json to Chef Infra Server



```
$ chef push prod policyfiles/myhaproxy.lock.json
```

```
Uploading policy myhaproxy (08c39ccc8f) to policy group prod
Uploading policy myhaproxy (c0cb162cdb) to policy group prod
Using build-essential 8.2.1 (4b9d5c72)
Using haproxy 8.3.0 (1a4f7607)
Using mingw 2.1.0 (9f5d572c)
Using myhaproxy 1.0.0 (1a9d7377)
Using seven_zip 3.1.2 (0elfed3b)
Using windows 6.0.1 (042f3380)
Using yum-epel 3.3.0 (187c02d6)
```

## GL: Converging the Load Balancer Node



```
$ knife ssh 'name:lb' -x chef -P PASSWORD 'sudo chef-client'
```

```
ec2-35-170-33-199.compute-1.amazonaws.com Starting Chef Infra Client, version 15.7.32
ec2-35-170-33-199.compute-1.amazonaws.com Using policy 'myhaproxy' at revision
'9da82055ea2c960cd680d131de8e92ba773703538575e4e429a52ca13f01fbba'
ec2-35-170-33-199.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy::default@1.0.0 (e9a41c2)"]
ec2-35-170-33-199.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-35-170-33-199.compute-1.amazonaws.com - build-essential (8.2.1)
ec2-35-170-33-199.compute-1.amazonaws.com - haproxy (8.3.0)
ec2-35-170-33-199.compute-1.amazonaws.com - mingw (2.1.0)
ec2-35-170-33-199.compute-1.amazonaws.com - seven_zip (3.1.2)
ec2-35-170-33-199.compute-1.amazonaws.com - windows (6.0.1)
ec2-35-170-33-199.compute-1.amazonaws.com - yum-epel (3.3.0)
ec2-35-170-33-199.compute-1.amazonaws.com - myhaproxy (1.0.0)
...
...
ec2-35-170-33-199.compute-1.amazonaws.com Chef Infra Client finished, 6/26 resources
updated in 05 seconds
```

Converge the cookbook by logging into that node and running 'sudo chef-client' or remotely administer the node with the 'knife ssh' command as shown here.

Within the output you should see the haproxy configuration file will update with a new entry that contains the information of the second member (`apache_web`).



## Dynamic Web Load Balancer

*Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!*

### Objective:

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company\_web policy\_name.
- ✓ Update the major version of the myhaproxy cookbook
- ✓ Update and push the Policyfile
- ✓ Run chef-client on the load balancer node
- Verify the load balancer node relays requests to both web nodes

The screenshot shows a load balancer interface with three main components:

- A large central window titled "Chef Welcomes You!" displaying system information for a Windows host:
  - PLATFORM: windows
  - HOSTNAME: WIN-DQFQCUFHDCP
  - MEMORY: 1048176kB
  - CPU Mhz: 2400
- A smaller window on the left titled "Chef Welcomes You!" displaying the same system information for the same host.
- A smaller window on the right titled "Chef Welcomes You!" displaying system information for a CentOS host:
  - PLATFORM: centos
  - HOSTNAME: ip-172-31-26-186
  - MEMORY: 604192kB
  - CPU Mhz: 1799.999

A blue arrow points from the central window's URL bar (54.199.197.193) to the URL bar of the left window (54.199.197.193), indicating they are the same host. A large Chef logo icon is in the top right corner.

©2020 Chef Software Inc. 15-30 

Point a web browser to your load balancer and refresh a couple times.

Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.

The diagram illustrates a load balancing setup. Two separate web browser windows are shown, each displaying a "Chef Welcomes You!" page with system information. A blue arrow points from the left window to the right window, indicating traffic flow between them. In the top right corner, there is a large black silhouette of a chef's hat.

**Left Window (54.199.197.193):**

- Chef Welcomes You!
- PLATFORM: windows
- HOSTNAME: WIN-DQFQCUFHDCP
- MEMORY: 1048176kB
- CPU Mhz: 2400

**Right Window (184.73.96.131):**

- Chef Welcomes You!
- PLATFORM: centos
- HOSTNAME: ip-172-31-26-186
- MEMORY: 604192kB
- CPU Mhz: 1799.999

©2020 Chef Software Inc. 15-31 

Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.



## Let's Test that Our Code Really Works

To verify that our code is working, let's remove the load balancer configuration file, forcing our code to run.

## GL: Delete the haproxy.cfg File



```
$ knife ssh 'name:lb' -x chef -P PASSWORD "sudo rm
/etc/haproxy/haproxy.cfg"
```

knife ssh 'name:lb' -x chef -P PASSWORD "sudo rm -r /etc/haproxy/haproxy.cfg"

## GL: Converge the Load Balancer



```
$ knife ssh 'name:lb' -x chef -P PASSWORD 'sudo chef-client'

ec2-34-196-50-77.compute-1.amazonaws.com Starting Chef Infra Client, version 15.7.32
...
ec2-34-196-50-77.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy::default@1.0.0 (e9a41c2)"]
ec2-34-196-50-77.compute-1.amazonaws.com Synchronizing Cookbooks:
...
ec2-34-196-50-77.compute-1.amazonaws.com * template[/etc/haproxy/haproxy.cfg] action
create
ec2-34-196-50-77.compute-1.amazonaws.com - create new file /etc/haproxy/haproxy.cfg
ec2-34-196-50-77.compute-1.amazonaws.com - update content in file
/etc/haproxy/haproxy.cfg from none to 4334de
ec2-34-196-50-77.compute-1.amazonaws.com - suppressed sensitive resource
ec2-34-196-50-77.compute-1.amazonaws.com - change mode from '' to '0644'
ec2-34-196-50-77.compute-1.amazonaws.com - change owner from '' to 'haproxy'
ec2-34-196-50-77.compute-1.amazonaws.com - change group from '' to 'haproxy'
...
...
```

Run 'sudo chef-client' on your lb node to apply the 'myhaproxy' cookbook.

The screenshot shows a load balancer interface with three main components:

- A large central window titled "Chef Welcomes You!" displaying system information for a Windows host:
  - PLATFORM: windows
  - HOSTNAME: WIN-DQFQCUFHDCP
  - MEMORY: 1048176kB
  - CPU Mhz: 2400
- A smaller window on the left titled "Chef Welcomes You!" displaying the same system information for the same host.
- A smaller window on the right titled "Chef Welcomes You!" displaying system information for a CentOS host:
  - PLATFORM: centos
  - HOSTNAME: ip-172-31-26-186
  - MEMORY: 604192kB
  - CPU Mhz: 1799.999

A blue arrow points from the central window's URL bar (54.199.197.193) to the URL bar of the left window (54.199.197.193), indicating they are the same host. A large black chef's hat icon is positioned in the top right corner of the slide.

©2020 Chef Software Inc. 15-35 

Point a web browser to your load balancer and refresh a couple times.

Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.

The diagram illustrates a load balancing setup. Two separate web browser windows are shown, each displaying a "Chef Welcomes You!" page with system information. A blue arrow points from the left window to the right window, indicating traffic flow between them. In the top right corner, there is a large black silhouette of a chef's hat.

**Left Window (54.199.197.193):**

- Chef Welcomes You!**
- PLATFORM: windows**
- HOSTNAME: WIN-DQFQCUFHDCP**
- MEMORY: 1048176kB**
- CPU Mhz: 2400**

**Right Window (184.73.96.131):**

- Chef Welcomes You!**
- PLATFORM: centos**
- HOSTNAME: ip-172-31-26-186**
- MEMORY: 604192kB**
- CPU Mhz: 1799.999**

©2020 Chef Software Inc. 15-36 

Nothing should change externally. You may see some differences in the logs as the `load_balancer` configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.



## Dynamic Web Load Balancer

*Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!*

### **Objective:**

- ✓ Update the myhaproxy cookbook to dynamically use nodes with the company\_web policy\_name.
- ✓ Update the major version of the myhaproxy cookbook
- ✓ Update and push the Policyfile
- ✓ Run chef-client on the load balancer node
- ✓ Verify the load balancer node relays requests to both web nodes



## Review Questions

1. What is the great advantage of using the following dynamic search in the load balancer's default.rb?

```
web_nodes = search('node','policy_name:company_web')

servers = []

web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']} #{web_node['cloud']['public_ipv4']}:80
maxconn 32"
 servers.push(server)
end
...
```

1. You would not need to hard code the web nodes' hostnames and IP addresses each time you need to add a web node.



## Review Questions

2. What is the key item that tells the load balancer how to find the web servers its supposed to balance?

```
web_nodes = search('node','policy_name:company_web')

servers = []

web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']} #{web_node['cloud']['public_ipv4']}:80
maxconn 32"
 servers.push(server)
end
...
```

2. The policy name `company_web` shown here:

```
web_nodes = search('node' , 'policy_name:company_web')
```

## Q&A

What questions can we help you answer?





©2020 Chef Software Inc.

## Running chef-client as a Service

Introducing the chef-client cookbook

## Lesson Objectives

After completing this module, you should be able to:

- Generate a chef-client wrapper cookbook
- Update the company\_web policyfile to include the chef-client wrapper cookbook
- Update and push the company\_web Policyfile to Chef Infra Server
- Apply the updated Policyfile to our web nodes
- Run chef-client as a service/task on our web nodes

In this module we are setting up chef-client to run as a service on our Linux web node and a task on our windows web node.



## 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?



## GL: View How chef-client is Configured

In this group lab you will view how chef-client is configured.

## Demo: View chef-client config Directory (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD "ls -F /etc/chef"
```

```
ec2-34-196-104-17.compute-1.amazonaws.com accepted_licenses/ chef_guid client.pem
client.rb first-boot.json ohai/
ec2-34-196-50-77.compute-1.amazonaws.com accepted_licenses/ chef_guid 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.

## Demo: View chef-client config File (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD "cat /etc/chef/client.rb"
```

```
ec2-54-209-220-6.compute-1.amazonaws.com chef_server_url "https://api.chef.io/organizations/jane3"
ec2-54-209-220-6.compute-1.amazonaws.com validation_client_name "jane3-validator"
ec2-54-209-220-6.compute-1.amazonaws.com chef_license "accept"
ec2-54-209-220-6.compute-1.amazonaws.com log_location STDOUT
ec2-54-209-220-6.compute-1.amazonaws.com node_name "lb"
ec2-54-209-220-6.compute-1.amazonaws.com
ec2-18-206-64-141.compute-1.amazonaws.com chef_server_url "https://api.chef.io/organizations/jane3"
ec2-18-206-64-141.compute-1.amazonaws.com validation_client_name "jane3-validator"
ec2-18-206-64-141.compute-1.amazonaws.com chef_license "accept"
ec2-18-206-64-141.compute-1.amazonaws.com log_location STDOUT
ec2-18-206-64-141.compute-1.amazonaws.com node_name "apache_web"
ec2-18-206-64-141.compute-1.amazonaws.com
```

The configuration file for chef-client is '/etc/chef/client.rb'. This file contains the URL for the Chef Server that chef-client should communicate with, i.e. the API endpoint. The validation\_client\_name parameter is only used with older versions of Chef Server to define what key is used for the initial authentication with the Chef Server. The file also contains the name of the node, which is used to identify the node on the Chef Server, as well as chef-client's log level and log location.

'chef-client' on the node and 'knife' on the workstation are both API clients in that they both communicate with the Chef Server over the API - the file '/etc/chef/client.rb' is the equivalent to the 'knife.rb' file on the workstation.

## Demo: View chef-client config Directory (Windows)



```
$ knife winrm "os:windows" -x USER -P PWD
-a cloud.public_ipv4 "dir C:\chef"
```

```
3.88.178.251 Volume in drive C has no label.
3.88.178.251 Volume Serial Number is D4E5-7A7A
3.88.178.251 Directory of C:\chef
3.88.178.251 02/11/2020 07:01 PM <DIR> .
3.88.178.251 02/11/2020 07:01 PM <DIR> ..
3.88.178.251 02/11/2020 07:01 PM <DIR> backup
3.88.178.251 02/11/2020 05:42 PM <DIR> cache
3.88.178.251 02/10/2020 09:40 PM 36 chef_guid
3.88.178.251 02/10/2020 09:39 PM 1,706 client.pem
3.88.178.251 02/10/2020 09:39 PM 349 client.rb
3.88.178.251 02/10/2020 09:39 PM 17 first-boot.json
3.88.178.251 02/01/2016 06:41 AM <DIR> ohai
```

We can find similar information on our windows nodes. This will be found in the 'C:\chef' directory on Windows.

## Demo: View chef-client config File (Windows)



```
$ knife winrm "os/windows" -x USER -P PWD
-a cloud.public_ipv4 "more C:\chef\client.rb"
```

```
3.88.178.251 chef_server_url "https://api.chef.io/organizations/jane3"
3.88.178.251
3.88.178.251 validation_client_name "jane3-validator"
3.88.178.251 file_cache_path "c:/chef/cache"
3.88.178.251 file_backup_path "c:/chef/backup"
3.88.178.251 cache_options {(:path => "c:/chef/cache/checksums", :skip_expires => true)}
3.88.178.251 chef_license "accept"
3.88.178.251 node_name "iis_web"
3.88.178.251 log_level :auto
3.88.178.251 log_location STDOUT
```

more



## 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.



## GL: The chef-client Cookbook

*We will use the chef-client community cookbook to configure chef-client on each of our nodes to run periodically*

### **Objective:**

- Generate a chef-client wrapper cookbook
- Update the company\_web policyfile to include the chef-client wrapper cookbook
- Push the company\_web policyfile to Chef Infra Server
- Apply the update Policyfile to our web nodes

### Instructor Note:

The GitHub repo for the 'mychef\_client' cookbook can be found at:  
[https://github.com/chef-training/devops-cookbooks-mychef\\_client.git](https://github.com/chef-training/devops-cookbooks-mychef_client.git)



## 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 mychef\_client Wrapper Cookbook



```
$ cd ~/chef-repo
$ chef generate cookbook cookbooks/mychef_client
```

```
Generating cookbook mychef_client
- Ensuring correct cookbook content
- Committing cookbook files to git
```

```
Your cookbook is ready. To setup the pipeline, type `cd cookbooks/mychef_client`, then run
'delivery init'
```

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: Update mychef\_client metadata.rb

□ ~/chef-repo/cookbooks/mychef\_client/metadata.rb

```
name 'mychef_client'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures mychef_client'
long_description 'Installs/Configures mychef_client'
version '0.1.0'
chef_version '>= 12.1' if respond_to?(:chef_version)

depends 'chef-client'
```

## GL: Edit mychef\_client Default Recipe



`cookbooks/mychef_client/recipes/default.rb`

```

Cookbook:: mychef_client
Recipe:: default

Copyright:: 2020, The Authors, All Rights Reserved.

include_recipe 'chef-client::default'
```

This recipe just calls the recipe `chef-client::default`

## GL: Edit company\_web.rb Policyfile

```
~/chef-repo/policyfiles/company_web.rb
```

```
...# run_list: chef-client will run these recipes in the order
specified.

run_list 'mychef_client::default', 'company_web::default'

Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
cookbook 'mychef_client', path: '../cookbooks/mychef_client'
```

Add the `mychef_client` cookbook to the run list and cookbook paths.

You should add the '`mychef_client::default`' cookbook to the beginning of the run list because system-level configurations should be applied before any application cookbooks.

In this way, if an application cookbook fails during convergence, the chef-client configuration would still be set up.

## GL: Update the Policyfile



```
$ chef update policyfiles/company_web.rb
```

```
...
Expanded run list: recipe[mychef_client::default],
recipe[company_web::default]
Caching Cookbooks...
Installing company_web >= 0.0.0 from path
Installing myiis >= 0.0.0 from path
Installing apache >= 0.0.0 from path
Installing mychef_client >= 0.0.0 from path
Installing chef-client 11.5.0
Installing cron 6.2.2
Installing logrotate 2.2.2
Lockfile written to /Users/sdelfante/chef-
repo/policyfiles/company_web.lock.json ...
```

## GL: Push the Policyfile



```
$ chef push prod policyfiles/company_web.lock.json
```

```
Uploading policy company_web (fd0a5ffelc) to policy group prod
Using apache 0.1.0 (1388ab3a)
Using company_web 0.1.1 (085c5742)
Using myiis 0.2.2 (c7630da4)
Uploaded chef-client 11.5.0 (7cb128f1)
Uploaded cron 6.2.2 (602e43b3)
Uploaded logrotate 2.2.2 (bd20a5c5)
Uploaded mychef_client 0.1.0 (c7c59f9a)
```

## GL: Converge the Linux Web Node



```
$ knife ssh 'name:apache_web' -x chef -P PWD 'sudo chef-client'

ec2-18-206-64-141.compute-1.amazonaws.com Using policy 'company_web' at
revision 'fd0a5ffelcf4748f69bc66076de563fa2d2214ca16803d69ad89f7dbfc09fa39'
ec2-18-206-64-141.compute-1.amazonaws.com resolving cookbooks for run list:
...
ec2-18-206-64-141.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-18-206-64-141.compute-1.amazonaws.com - apache (0.1.0)
ec2-18-206-64-141.compute-1.amazonaws.com - company_web (0.1.0)
ec2-18-206-64-141.compute-1.amazonaws.com - chef-client (11.5.0)
ec2-18-206-64-141.compute-1.amazonaws.com - myiis (0.2.2)
ec2-18-206-64-141.compute-1.amazonaws.com - logrotate (2.2.2)
ec2-18-206-64-141.compute-1.amazonaws.com - cron (6.2.2)
ec2-18-206-64-141.compute-1.amazonaws.com - mychef_client (0.1.0)
```

## GL: Verify chef-client is Running (Linux)



```
$ knife ssh 'name:apache_web' -x chef -P PWD "ps awux | grep chef-client"
```

```
ec2-18-206-64-141.compute-1.amazonaws.com root 6462 0.9 6.1 299884 62692 ?
Ss 17:12 0:02 /opt/chef-workstation/embedded/bin/ruby --disable-gems
/usr/bin/chef-client -c /etc/chef/client.rb -i 1800 -s 300
ec2-18-206-64-141.compute-1.amazonaws.com chef 6530 0.0 0.1 113180 1604
pts/0 S+ 17:17 0:00 bash -c ps awux | grep chef-client
ec2-18-206-64-141.compute-1.amazonaws.com chef 6546 0.0 0.0 112708 972
pts/0 S+ 17:17 0:00 grep chef-client
```

Notice that the interval is set to 1800 seconds.

The output shows that chef-client is running as a service and that it is set to run every 1800 seconds.

## GL: Converge the iis\_web Node (Windows)



```
$ knife winrm 'name:iis_web' -x USER -P PWD
-a cloud.public_ipv4 "chef-client"
```

```
34.195.38.226 Using policy 'company_web' at revision
'7f50faaaaf9eceb35f72358c38a71b86c7fb7df8f49fee245827eb4613d919827'
34.195.38.226
34.195.38.226 resolving cookbooks for run list: ["mychef_client::default@0.1.0
(f5b035d)", "company_web::default@0.1.0 (c1b26cb)"]
34.195.38.226 Synchronizing Cookbooks:
34.195.38.226 - apache (0.1.0)
34.195.38.226 - company_web (0.1.0)
34.195.38.226 - cron (6.2.1)
34.195.38.226 - myiis (0.2.1)
34.195.38.226 - logrotate (2.2.0)
34.195.38.226 - mychef_client (0.1.0) ...
```

We need to apply this updated policy to our iis\_web node as well.

## GL: Verify chef-client is Running (Windows)



```
$ knife winrm 'name:iis_web' -x Administrator -P PWD -a
cloud.public_ipv4 'schtasks /Query | findstr /i "chef-client"'
```

3.88.178.251 chef-client

2/13/2020 5:52:00 PM Ready

And we verify that the chef-client task is running on our iis\_web server as well by querying the scheduled tasks searching for '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:

- ✓ Generate a chef-client wrapper cookbook
- ✓ Update the company\_web policyfile to include the chef-client wrapper cookbook
- ✓ Push the company\_web policyfile to Chef Infra Server
- ✓ Apply the update Policyfile to our web nodes

### Instructor Note:

The GitHub repo for the 'mychef\_client' cookbook can be found at:  
[https://github.com/chef-training/devops-cookbooks-mychef\\_client.git](https://github.com/chef-training/devops-cookbooks-mychef_client.git)



## GL: Change Default Settings

Wait!

There has just been a mandate that every web node in the infrastructure must run chef-client every 5 minutes (300 seconds).

We'll accomplish this change by creating an attribute file which will override the default setting of 1800 seconds

## GL: Generate the default Attribute File



```
$ chef generate attribute cookbooks/mychef_client default

Recipe: code_generator::attribute
 * directory[cookbooks/mychef_client/attributes] action create
 - create new directory cookbooks/mychef_client/attributes
 * template[cookbooks/mychef_client/attributes/default.rb] action
create
 - create new file
cookbooks/mychef_client/attributes/default.rb
 - update content in file
cookbooks/mychef_client/attributes/default.rb from none to e3b0c4
 (diff output suppressed by config)
```

Let's generate a default attribute file where we can set our own node attributes.

## GL: Set the chef\_client Interval



cookbooks/mychef\_client/attributes/default.rb

```
default['chef_client']['interval'] = '300'
```

Now that we have our default attribute file, we will set the chef-client run interval to 300 seconds.

## Node Attribute Precedence

	Attribute Files	Node/Recipe	Policy File
default	1	2	3
force_default	4	5	
normal	6	7	
override	8	9	10
force_override	11	12	
automatic from ohai highest priority			

We must set the precedence of node attributes which will allow them to be overridden when chef-client is executed. There are three locations where we can set a node attribute: Attribute files, recipes, and Policyfile (aka roles).

With each location we assign a precedence like default, force\_default, normal, override, and force\_override. With each location and precedence level there is an assigned value. The higher the value, the higher the precedence, and whatever has the highest value will win out when chef-client is executed and the associated value for the node attribute will be used. At the top level we have automatic. Automatic node attribute precedence is reserved for node attributes collected by Ohai and cannot be overridden.

## GL: Update the Policyfile



```
$ chef update policyfiles/company_web.rb
```

```
...
Expanded run list: recipe[mychef_client::default],
recipe[company_web::default]
Caching Cookbooks...
Installing company_web >= 0.0.0 from path
Installing myiis >= 0.0.0 from path
Installing apache >= 0.0.0 from path
Installing mychef_client >= 0.0.0 from path
Using chef-client 11.2.0
Using cron 6.2.1
Using logrotate 2.2.0
Lockfile written to /Users/sdelfante/chef-repo/company_web.lock.json
Policy revision id
```

## GL: Push the Policyfile to Prod



```
$ chef push prod policyfiles/company_web.lock.json
```

```
Uploading policy company_web (1e97a11553) to policy group prod
Using apache 0.1.0 (1388ab3a)
Using chef-client 11.5.0 (7cb128f1)
Using company_web 0.1.0 (085c5742)
Using cron 6.2.2 (602e43b3)
Using logrotate 2.2.2 (bd20a5c5)
Using myiis 0.2.2 (c7630da4)
Uploaded mychef_client 0.1.0 (f79fa661)
```

## GL: Converge the Linux Web Node



```
$ knife ssh 'name:apache_web' -x chef -P PWD 'sudo chef-client'

...
ec2-34-196-104-17.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-34-196-104-17.compute-1.amazonaws.com - apache (0.1.0)
ec2-34-196-104-17.compute-1.amazonaws.com - chef-client (11.2.0)
ec2-34-196-104-17.compute-1.amazonaws.com - cron (6.2.1)
ec2-34-196-104-17.compute-1.amazonaws.com - myiis (0.2.1)
ec2-34-196-104-17.compute-1.amazonaws.com - logrotate (2.2.0)
ec2-34-196-104-17.compute-1.amazonaws.com - company_web (0.1.0)
ec2-34-196-104-17.compute-1.amazonaws.com - mychef_client (0.1.0)

...
* template[/etc/sysconfig/chef-client] action create
ec2-34-196-104-17.compute-1.amazonaws.com - update content in file /etc/sysconfig/chef-
client from ec7de1 to 4251d6
ec2-34-196-104-17.compute-1.amazonaws.com --- /etc/sysconfig/chef-client 2020-07-31
16:12:34.501755564 +0000
```

## GL: Verify chef-client is Running (Linux)



```
$ knife ssh 'name:apache_web' -x chef -P PWD "ps awux | grep chef-client"
```

```
ec2-18-206-64-141.compute-1.amazonaws.com root 6864 5.2 6.1 299356 62120 ?
Ss 17:32 0:02 /opt/chef-workstation/embedded/bin/ruby --disable-gems
/usr/bin/chef-client -c /etc/chef/client.rb -i 300 -s 300
ec2-18-206-64-141.compute-1.amazonaws.com chef 6884 0.0 0.1 113180 1600
pts/0 S+ 17:32 0:00 bash -c ps awux | grep chef-client
ec2-18-206-64-141.compute-1.amazonaws.com chef 6900 0.0 0.0 112708 972
pts/0 S+ 17:32 0:00 grep chef-client
```

Notice that the interval is now 300 seconds instead of 1800 like before.

The output shows that chef-client is running as a service and that it is set to run every 300 seconds (5 minutes).

## GL: Converge the iis\_web Node (Windows)



```
$ knife winrm 'name:iis_web' -x USER -P PWD
-a cloud.public_ipv4 "chef-client"
```

```
3.88.178.251 Starting Chef Infra Client, version 15.8.23
3.88.178.251 Using policy 'company_web' at revision
'1e97a11553265cefa813fab2898e718040939c5787f9f1a18c3bf2714f82898b'
3.88.178.251
3.88.178.251 resolving cookbooks for run list: ["mychef_client::default@0.1.0
(f79fa66)", "company_web::default@0.1.1 (085c574)"]
...
3.88.178.251 * windows_service[w3svc] action enable (up to date)
3.88.178.251 * windows_service[w3svc] action start (up to date)
3.88.178.251 Running handlers:
3.88.178.251 Running handlers complete
3.88.178.251 Chef Infra Client finished, 1/10 resources updated in 06 seconds
```

We need to apply this updated policy to our iis\_web node as well.

## GL: Verify chef-client is Running (Windows)



```
$ knife winrm 'name:iis_web' -x Administrator -P PWD -a
cloud.public_ipv4 'schtasks /Query | findstr /i "chef-client"'
```

3.88.178.251 chef-client	2/13/2020 5:52:00 PM	Ready
[REDACTED]		

And we verify that the chef-client task is running on our iis\_web server as well by querying the scheduled tasks searching for 'chef-client'.



## Q&A

What questions can we help you answer?

- Chef Supermarket
- Wrapper Cookbooks
- chef-client Cookbook

What questions can we help you answer?

In general or about specifically about Chef Super Market, wrapper cookbooks, node attributes, the 'knife ssh' command.



©2020 Chef Software Inc.

## Using Policy Groups to Reflect Environments

Separating your nodes with policy\_group

# Objectives

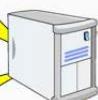
After completing this module, you should be able to

- Deploy a node to an environment via policy\_group
- Update the load balancer's search query
- Test your load balancer to confirm that policy\_group is separating your node from a group of nodes.

In this section, you will learn how to deploy a node to an policy\_group environment and update a search query.

# Keeping Your Infrastructure Current

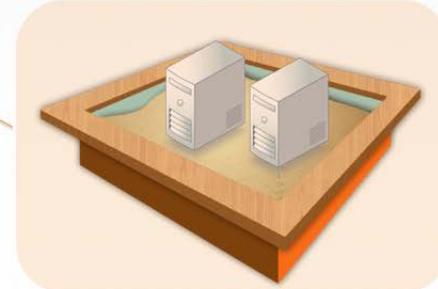
Changing Needs  
Changing Software  
Growing Organization  
Increased Website Popularity



Production



Acceptance



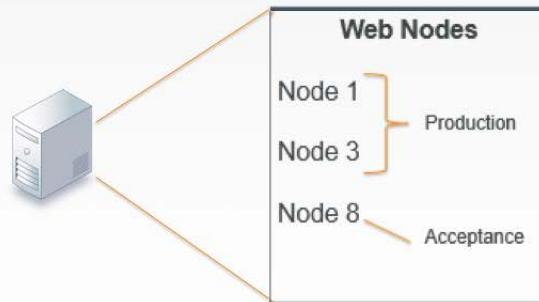
So, we have updated our load balancer's myhaproxy cookbook to dynamically search for and update nodes. Everything is as it should be. But our system is like a living, breathing thing that must grow and be updated to fit our changing needs. We need to find a way to update and test new tools, features and settings without impacting our current production system.

Of course, we have local testing tools like Test Kitchen to help us verify that our individual cookbooks work before we upload them to the Chef Server. But, that is not always enough. We may want to build, test, and release new features to our cookbooks but we do not immediately want all of our nodes to immediately use them. For example, what if we had a requirement to update our apache cookbook with a new front page for our application? The release date of our new service with the sign up page does not go live for a week. So, we want to build, test, and upload that cookbook to the Chef Server without actually applying the cookbook until the release date. How would we accomplish that?

This is where `policy_group` environments are useful.

# **policy\_group Environments**

Environments can define different functions of nodes that live 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.

## Assigning a Node to an Environment

```
knife node policy set iis_web prod company_web
```



Assigning a node to an environment is as simple as specifying a `policy_group` in the `knife node policy set...` command.

In this example we assigned the `iis_web` node to the `prod` (production) environment. In this module, you will move your `iis_web` node to a new environment called `acceptance` and see the results.

## Assigning a Node to an Environment

```
knife node policy set iis_web prod company_web
```



We will leave our load balancer and our **apache\_web** node in the **prod** environment so the load balancer will serve up only nodes in the **prod** environment.

**Reminder:** The first time you specify a policy group, that policy group name will be instantiated in Chef Infra Server. Then you can reuse it for other nodes.



## Group Lab: Using policy\_group

*Let's create an acceptance policy\_group environment for our nodes*

**Objective:**

- Test your current load balancer's behavior
- Assign the `iis_web` node to acceptance
- Update the load balancer's search criteria
- Converge the load balancer node
- Test your load balancer

**GL: Test the Load Balancer**

**Chef Welcomes You!**

**PLATFORM: windows**

**HOSTNAME: WIN-8694LT97S51**

**MEMORY: 8388208kB**

**CPU Mhz: 2400**

**Chef Welcomes You!**

**PLATFORM: windows**

**HOSTNAME: WIN-8694LT97S51**

**MEMORY: 8388208kB**

**CPU Mhz: 2400**

**Chef Welcomes You!**

**PLATFORM: centos**

**HOSTNAME: ip-172-31-29-209**

**MEMORY: 604192kB**

**CPU Mhz: 1795.672**

©2020 Chef Software Inc. 17-8 

First, point a web browser to the URL of your load balancer/proxy server and then click Refresh a few times. You should see each of your web server's HTML page as the Proxy server switches between each web server, just like in a previous module.

Reminder: The haproxy shared cache could take a few second to a couple minutes before it refreshes, so you may not see the "Chef Welcomes You!" page update immediately. You may need to wait 20 or more seconds between refreshes.

**GL: Test the Load Balancer**

**Chef Welcomes You!**

**PLATFORM: centos**

**HOSTNAME: ip-172-31-29-209**

**MEMORY: 604192kB**

**CPU Mhz: 1795.672**

**Chef Welcomes You!**

**PLATFORM: windows**

**HOSTNAME: WIN-8694LT97S51**

**MEMORY: 8388208kB**

**CPU Mhz: 2400**

**Chef Welcomes You!**

**PLATFORM: centos**

**HOSTNAME: ip-172-31-29-209**

**MEMORY: 604192kB**

**CPU Mhz: 1795.672**

©2020 Chef Software Inc. 17-9 

First, point a web browser to the URL of your load balancer/proxy server and then click Refresh a few times. You should see each of your web server's HTML page as the Proxy server switches between each web server, just like in a previous module.

**GL: Push the company\_web.lock.json to a new acceptance Environment**

```
~/chef-repo> chef push acceptance policyfiles/company_web.lock.json
```

```
Uploading policy company_web (1e97a11553) to policy group acceptance
Using apache 0.1.0 (1388ab3a)
Using chef-client 11.5.0 (7cb128f1)
Using company_web 0.1.0 (085c5742)
Using cron 6.2.2 (602e43b3)
Using logrotate 2.2.2 (bd20a5c5)
Using mychef_client 0.1.0 (f79fa661)
Using myiis 0.2.2 (c7630da4)
```

New  
policy\_group

## GL: Show the Policies on Chef Infra Server



```
~/chef-repo> chef show-policy
```

```
company_web
=====
* acceptance: 1e97a11553
* prod: 1e97a11553

myhaproxy
=====
* acceptance: *NOT APPLIED*
* prod: e633695adc
```

Here we can see that the **company\_web** policy has been uploaded to Chef Infra Server and is in the **acceptance** policy\_group.

## GL: Assign the iis\_web Node to acceptance



```
> knife node policy set iis_web acceptance company_web
```

Successfully set the policy on node iis\_web

node name

policy\_group

policy\_name

Here we are setting the iis\_web node to the acceptance policy\_group.

## GL: View Information About Your Node



```
$ knife node show iis_web
```

```
Node Name: iis_web
Policy Name: company_web
Policy Group: acceptance
FQDN: WIN-DQFQCUFHDCP.ec2.internal
IP: 3.88.178.251
Run List: recipe[mychef_client::default], recipe[company_web::default]
Recipes: mychef_client::default, company_web::default, chef-client::default, chef-client::task, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```

You can see more information about a particular node with the command 'knife node show iis\_web'. This will display a summary of the node information that the Chef Infra Server stores.

## GL: Modify the Load Balancer's Existing Search Criteria

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

```
...
web_nodes = search('node','policy_name:company_web')
servers = []
web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']}"
 "#{web_node['cloud']['public_ipv4']}":80 maxconn 32"
 servers.push(server)
end

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy'
```

Looking at the existing recipe in the load balancer's myhaproxy cookbook, we can review the original search syntax. Remove or comment the highlighted search criteria

## GL: Update the Search to Consider policy\_group

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

```
...
web_nodes = search('node',"policy_name:company_web AND policy_group:#{node.policy_group}")
servers = []
web_nodes.each do |web_node|
 server = "#{web_node['cloud']['public_hostname']} #{web_node['cloud']['public_ipv4']}:80"
 maxconn 32"
 servers.push(server)
end

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy'
```

**Note:** We are now using double quotes in the search string.

The double quotes are to interpolate the node.policy\_group variable.

Replace the old search criteria with the following:

```
all_web_nodes = search('node',"policy_name:company_web AND
policy_group:#{node.policy_group}")
```

## GL: Update the Search to Consider policy\_group

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

web_nodes = search('node', "policy_name:company_web AND policy_group:#{node.policy_group}")
servers = []
web_nodes.each do |web_node|

 server = "#{web_node['cloud']['public_hostname']} #{web_node['cloud']['public_ipv4']}:80
maxconn 32"
 servers.push(server)
end

haproxy_backend 'servers' do
 server servers
end

haproxy_service 'haproxy' do
 subscribes :reload, 'template[/etc/haproxy/haproxy.cfg]', :delayed
end
```

**Note:** We are forcing the haproxy service to reload because we've updated the web server pool.

Instructor Note: This is the first time we've used a notification. These come in the form of "notifies" and "subscribes". Any resource in the resource collection (all resources used during a chef-client run) can send and receive notifications. These allow resources to inform one another when their state changes.

In this example, we are telling the haproxy\_service resource to take the action of :reload when the template for the haproxy.cfg file changes on disk. This way, anytime the webserver pool is updated, the haproxy service will be reloaded.

You can have as many notifies or subscribes statements on a resource as you would like, and you can also set timers of :before, :immediately or :after to declare when the actions should take place.

[https://docs.chef.io/resource\\_common.html#notifications](https://docs.chef.io/resource_common.html#notifications)

[https://github.com/sous-chefs/haproxy/blob/master/documentation/haproxy\\_service.md](https://github.com/sous-chefs/haproxy/blob/master/documentation/haproxy_service.md)

## GL: Bump the myhaproxy Version in metadata.rb

~/chef-repo/cookbooks/myhaproxy/metadata.rb

```
name 'myhaproxy'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'All Rights Reserved'
description 'Installs/Configures myhaproxy'
version '1.1.0'
chef_version '>= 14.0'
depends 'haproxy', '~> 8.3.0'
```

## GL: Update the myhaproxy.rb Policy



```
chef update policyfiles/myhaproxy.rb
```

```
Attributes already up to date
Building policy myhaproxy
Expanded run list: recipe[myhaproxy::default]
Caching Cookbooks...
Installing myhaproxy >= 0.0.0 from path
Using haproxy 8.3.0
Using build-essential 8.2.1
Using yum-epel 3.3.0
Using seven_zip 3.1.2
Using mingw 2.1.0
Using windows 6.0.1
Lockfile written to /Users/sdelfante/chef-repo/policyfiles/myhaproxy.lock.json
...
```

We just updated the `myhaproxy/recipes/default.rb` so we need to update the `myhaproxy.rb` policy.

## GL: Push the myhaproxy.rb Policy



```
chef push prod policyfiles/myhaproxy.lock.json
```

```
Uploading policy myhaproxy (b44fabe708) to policy group prod
Using build-essential 8.2.1 (4b9d5c72)
Using haproxy 8.3.0 (1a4f7607)
Using mingw 2.1.0 (9f5d572c)
Using myhaproxy 1.1.0 (c30514f7)
Using seven_zip 3.1.2 (0elfed3b)
Using windows 6.0.1 (042f3380)
Using yum-epel 3.3.0 (187c02d6)
```

Now we need to push the myhaproxy.lock.json to Chef Infra Server.

## GL: Converge the Load Balancer node



```
$ knife ssh 'name:lb' -x chef -P PWD 'sudo chef-client'
```

```
...
ec2-54-209-220-6.compute-1.amazonaws.com - update content in file
/etc/haproxy/haproxy.cfg from 91c09e to 8f4138
ec2-54-209-220-6.compute-1.amazonaws.com - suppressed sensitive resource
ec2-54-209-220-6.compute-1.amazonaws.com * service[haproxy] action enable (up to
date)
ec2-54-209-220-6.compute-1.amazonaws.com * service[haproxy] action start (up to
date)
ec2-54-209-220-6.compute-1.amazonaws.com * haproxy_service[haproxy] action reload
ec2-54-209-220-6.compute-1.amazonaws.com * service[haproxy] action reload
ec2-54-209-220-6.compute-1.amazonaws.com - reload service service[haproxy]
ec2-54-209-220-6.compute-1.amazonaws.com (up to date) ...
ec2-54-209-220-6.compute-1.amazonaws.com Running handlers:
ec2-54-209-220-6.compute-1.amazonaws.com Running handlers complete
ec2-54-209-220-6.compute-1.amazonaws.com Chef Infra Client finished, 2/26 resources
```

## GL: Only the apache\_web Node is Being Proxyed

URL of load balancer.

Output from the iis\_web server.

Chef Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-57-169

MEMORY: 1013192kB

CPU Mhz: 2400.184

©2020 Chef Software Inc.

17-21

 CHEF

Point a web browser to the URL or public IP address of your load balancer. It should display the web page of the apache\_web server node that the load balancer is configured to serve. This is because our load balancer and our apache\_web server nodes are now in the prod environment policy\_group while the iis\_web node is in the acceptance environment policy\_group.



## Group Lab: Using policy\_group

*Let's create an acceptance environment for our nodes*

**Objective:**

- ✓ Test your current load balancer's behavior
- ✓ Assign the `iis_web` node to acceptance
- ✓ Update the load balancer's search criteria
- ✓ Converge the load balancer node
- ✓ Test your load balancer



## Review Questions

1. What is the benefit of constraining cookbooks to a particular environment?
2. What is the key item that defines an environment?
3. What does this bit of code in the load balancer do?

```
web_nodes = search('node',"policy_name:company_web AND policy_group:#{{node.policy_group}}")
```

1. So you can separate your nodes for specific purposes such as placing a node in an Acceptance environment while testing it.
2. policy\_group.
3. It searches for nodes that have the company\_web policy\_name AND that belong to the same policy\_group (environment) that the load balancer is in.

## Q&A

What questions can we help you answer?





**CHEF**<sup>TM</sup>

---

©2020 Chef Software Inc.

## Further Resources

Other Places to Talk About, Practice, and Learn Chef



## Going Forward

There are many Chef resources available to you outside this class. During this module we will talk about just a few of those resources.

*The best way to learn Chef is to use Chef*

You did it! You reached the end of DevOps Foundation, but where do we go from here? Remember that the best way to learn Chef is to use Chef and we are going to provide you with some resources to continue working with Chef.



## Practice Chef

First, let's talk about stuff you can read to help you learn Chef.

## Learn Chef Rally

Interactive learning for those new to Chef.

[learn.chef.io](https://learn.chef.io)





## Beyond the Basics

- What happens during a knife bootstrap?
- What happens during a chef-client run?
- What is the security model used by chef-client?
- Further explanation of Attribute Precedence

<https://learn.chef.io/modules/beyond-the-basics#/>

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.



## 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.



## 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.

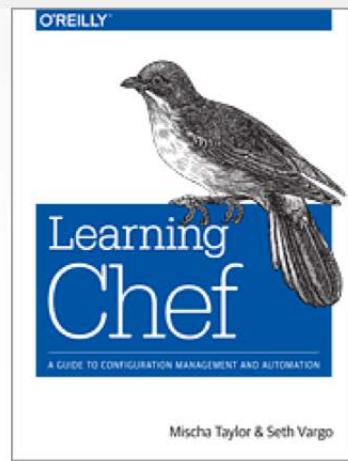
Any question you have, you probably will find the answer for on our Docs site.

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.

# Learning Chef

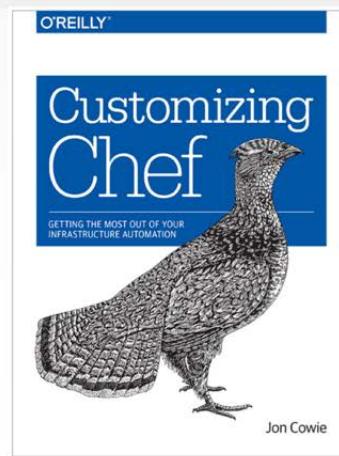
A Guide to Configuration Management and Automation



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



©2020 Chef Software Inc.

20-10

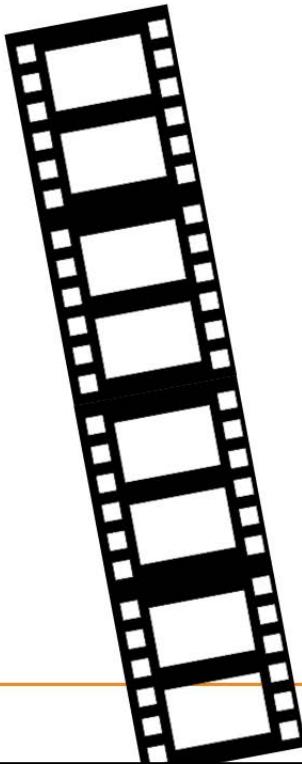


Additionally, you may want to read Jon Cowie's Getting the Most Out of Your Infrastructure Automation. It's also available on O'Reilly.

## 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. The webinars are highly recommended as well!

# Chef Developers' Slack Meetings

<https://github.com/chef/chef-community-slack-meetings>



Join members of the Chef Community in our community Slack channel. Have access to and start connecting with some of the best Chef's out there!

<https://github.com/chef/chef-community-slack-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://www.chef.io/feedback/>

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://www.chef.io/feedback/>



# CHEF CONF 2020

# CHEFCONF 2020 IS COMING!

Seattle | June 1-4   London | June 15-17

[chefconf.chef.io](http://chefconf.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 2020 will be held in Seattle June 1-4 and in London on June 15-17.

[chefconf.chef.io](http://chefconf.chef.io)



**CHEF**<sup>TM</sup>

**Thank You!**

---

©2020 Chef Software Inc.

And finally, thank you so much! Hope you enjoyed class!

## Appendix A Optional: Data Bags

Working with Custom Data Sets

Instructor Note: You can teach this optional module if time allows and if the audience would be interested in this topic.

# Objectives



After completing this module, you should be able to

- Explain how to use and manage a Data Bag
- Create and upload a Data Bag to Chef Server
- Query Data Bag information with the CLI
- Generate the 'myusers' cookbook
- Add the 'myusers' cookbook to company\_web policyfile's run list
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes

In this module we will be learning how to create and manage our own data bags. We can query this data bag from the CLI and use this information within our recipes as well.



## What about user data?

Where should we store data that each node might need access to?

There will come situations that we want to store data about our users. What would be the most effective way to do this?



## What about user data?

We could start by storing information about users as Node Attributes...

But this would duplicate a lot of information - every user in the company would be stored in every Node object!

If we created a node attribute for our users we would be adding a lot of information that may not be used by every node. This might not be the most efficient way to store this data.



## Data Bags

A data bag is a container for items that represent information about your infrastructure that is not tied to a single node.

Examples:

- Users
- Groups
- Application Release Information
- Passwords (in an encrypted data bag)

[https://docs.chef.io/data\\_bags.html](https://docs.chef.io/data_bags.html)

In this case, we would want to use what's known as a data bag. Data bags are containers for data and allow us to store information on the Chef Server rather than within the node objects themselves. In its essence a data bag is a user defined index on the Chef Server much like how we have the client, node, role and environment indexes. Whatever information we store within this data bag will be accessible to us from the command line as well as from within our recipes.



## Group Lab: Custom Data Sets

*We can store sets of JSON data on our Chef Server, accessible by a node with search*

**Objective:**

- Create “users” data bags
- Upload data bags to Chef Server
- Use CLI to query information about data bags

Let's go ahead and create our first data bag for our users.

## GL: What Can 'knife data bag' Do?



```
$ cd ~/chef-repo
$ knife data bag --help
** DATA BAG COMMANDS **
knife data bag create BAG [ITEM] (options)
knife data bag delete BAG [ITEM] (options)
knife data bag edit BAG ITEM (options)
knife data bag from file BAG FILE|FOLDER [FILE|FOLDER...] (options)
knife data bag list (options)
knife data bag show BAG [ITEM] (options)
```

Running 'knife data bag --help' will display for us the sub-options for 'knife data bag'. One of these options is a 'knife data bag list'.

## GL: Run 'knife data bag list'



```
$ knife data bag list
```



'knife data bag list' will display for us all of the data bags found on our Chef Server. At the moment this is an empty list.

## GL: Create a `data_bags` Directory



```
$ mkdir data_bags
```

We want to have a logical location for storing the .json files associated with our users so let's create a 'data\_bags' directory.

## GL: Create a `data_bags/users` Directory



```
$ mkdir data_bags/users
```

Within this 'data\_bags' directory we will create another directory for our users. This is where we will place all of the .json files associated with our users.

## GL: Create users Data Bag on Chef Server



```
$ knife data bag create users
```

```
Created data_bag[users]
```

For us to create the data bag for our users on the Chef Server itself, we run this command.

## GL: Create centos\_user.json

```
~/chef-repo/data_bags/users/centos_user.json
```

```
{
 "id": "centos_user",
 "comment": "I am a centos_user user",
 "platform": "centos"
}
```

Here we will define the data representing our centos\_user user. This is in json format which will use a series of key value pairs much like what we see with node attributes. Here we want to set the 'id', 'comment', and 'platform' keys for our user.

## GL: Create windows\_user.json

```
~/chef-repo/data_bags/users/windows_user.json
```

```
{
 "id": "windows_user",
 "comment": "I am a windows user",
 "platform": "windows"
}
```

We'll also create a windows\_user as well.



## Group Lab: Custom Data Sets

*We can store sets of JSON data on our Chef Server, accessible by a node with search*

**Objective:**

- Create “users” data bags
- Upload data bags to Chef Server
- Use CLI to query information about data bags

## GL: Upload Data Bag Items to Chef Server



```
$ knife data bag from file users data_bags/users/centos_user.json
data_bags/users/windows_user.json
```

```
Updated data_bag_item[users::centos_user]
Updated data_bag_item[users::windows_user]
```

You should still be in ~/chef-repo when running this command.

Now that we have defined our users in our .json files, we need to get this data up to the Chef Server. We do so with a 'knife data bag from file' and then specify the path the .json files that we wish to add to the data bag.

## GL: Validate Chef Server received items



```
$ knife data bag show users
```

```
centos_user
windows_user
```

We can see what is found within the 'users' data bag by querying the Chef Server from the command line.



## Group Lab:Custom Data Sets

*We can store sets of JSON data on our Chef Server,  
accessible by a node with search*

### **Objective:**

- Create “users” data bags
- Upload data bags to Chef Server
- Use CLI to query information about data bags

## GL: View Details of centos\_user



```
$ knife data bag show users centos_user
```

```
comment: I am a centos_user user
id: centos_user
platform: centos
```

If we wish to find out the specifics regarding our centos\_user, we can run a 'knife data bag show' command specifying that it is the 'centos\_user' within the 'users' data bag.

## GL: Search the users Index



```
$ knife search users "*:*"
```

```
2 items found

chef_type: data_bag_item
comment: I am a centos_user user
data_bag: users
id: centos_user
platform: centos

chef_type: data_bag_item
comment: I am a windows user
data_bag: users
id: windows_user
platform: windows
```

If there is a need to see everything within a data bag using wildcards will return everything found in the data bag.

## GL: Return Users with “platform:centos”



```
$ knife search users "platform:windows"
```

```
1 items found
```

```
chef_type: data_bag_item
comment: I am a windows user
data_bag: users
id: windows_user
platform: windows
```

Using this search criteria, we can return every user that has the platform of Windows.



## Group Lab: Custom Data Sets

*We can store sets of JSON data on our Chef Server,  
accessible by a node with search*

### Objective:

- ✓ Create “users” data bags
- ✓ Upload data bags to Chef Server
- ✓ Use CLI to query information about data bags



## GL: Create Users from a Data Bag

*Dynamically search through the Chef Server under the 'users' index to create users*

**Objective:**

- Generate the 'myusers' cookbook
- Create users based on data bag contents within default recipe
- Add the 'myusers' cookbook to company\_web policyfile's run list
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes

How might we go about creating users on our nodes dynamically using information stored within our 'users' data bag?



## Where are the users?

Because our users are now indexed on the Chef Server, we have a centralized source of truth for information regarding these users.

We can search through this information inside of our recipes.

Just like other information stored on the Chef Server, we can use the search method to dynamically use information within our recipes.

## GL: Generate the myusers Cookbook



```
$ cd ~/chef-repo
$ chef generate cookbook cookbooks/myusers
```

```
Generating cookbook myusers
- Ensuring correct cookbook content
- Committing cookbook files to git
```

```
Your cookbook is ready. To setup the pipeline, type `cd cookbooks/myusers`,
then run `delivery init`
```

Generate a 'myusers' cookbook which will be creating users on our nodes.



## GL: Create Users from a Data Bag

*Dynamically search through the Chef Server under the 'users' index to create users*

**Objective:**

- ✓ Generate the 'myusers' cookbook
- Create users based on data bag contents within the users recipe
- Add the 'myusers' cookbook to company\_web policyfile's run list
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes

## GL: Create users recipe

```
~/chef-repo/cookbooks/myusers/recipes/users.rb
```

```
system_users = search("users", "platform:#{{node['platform']}}")

system_users.each do |user_data|
 user user_data['id'] do
 comment user_data['comment']
 action :create
 end
end
```

Here we define a 'system\_users' variable and assign it to the results of the search method. The search method queries the 'users' data bag for every user that has the same platform as the node that is applying this code. So if this is a windows machine it will return our windows\_user. If this is a Centos machine our centos\_user will be returned. Once the 'system\_users' variable has been populated with each user that has the same platform as the node itself, a .each method is called that will loop through each user and is represented by the iteration variable 'user\_data'. Within the do end block we call the 'user' resource using the 'id' field of the user as the name. Calling the 'comment' property we add the user's comment to the newly created user as well.

## GL: Include users recipe within default recipe

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

```

Cookbook:: myusers
Recipe:: default

Copyright:: 2018, The Authors, All Rights Reserved.

include_recipe 'myusers::users'
```

Let's include this new 'users' recipe within the 'default' recipe.



## GL: Create Users from a Data Bag

*Dynamically search through the Chef Server under the 'users' index to create users*

**Objective:**

- ✓ Generate the 'myusers' cookbook
- Create users based on data bag contents within the users recipe
- Add the 'myusers' cookbook to company\_web policyfile's run list
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes

## GL: Add myusers Cookbook to the company\_web Policyfile

```
~/chef-repo/policyfiles/company_web.rb

...skipping...

run_list: chef-client will run these recipes in the order specified.
run_list 'mychef_client::default','company_web::default','myusers::default'

Specify a custom source for a single cookbook:
cookbook 'company_web', path: '../cookbooks/company_web'
cookbook 'myiis', path: '../cookbooks/myiis'
cookbook 'apache', path: '../cookbooks/apache'
cookbook 'mychef_client', path: '../cookbooks/mychef_client'
cookbook 'myusers', path: '../cookbooks/myusers'
```



## GL: Create Users from a Data Bag

*Dynamically search through the Chef Server under the 'users' index to create users*

**Objective:**

- ✓ Generate the 'myusers' cookbook
- ✓ Create users based on data bag contents within users recipe
- ✓ Add the 'myusers' cookbook to company\_web policyfile's run list
- ❑ Update and push the policyfile to the Chef Infra Server
- ❑ Converge web nodes

## GL: Ensure You are in chef-repo/policyfiles



```
$ cd ~/chef-repo/policyfiles
```

## GL: Update the Policyfile



```
$ chef update company_web.rb
```

```
Attributes already up to date
Building policy company_web
Expanded run list: recipe[mychef_client::default], recipe[company_web::default],
recipe[myusers::default]
Caching Cookbooks...
Installing company_web >= 0.0.0 from path
Installing myiis >= 0.0.0 from path
Installing apache >= 0.0.0 from path
Installing mychef_client >= 0.0.0 from path
Installing myusers >= 0.0.0 from path
Using chef-client 11.2.0
Using cron 6.2.1
Using logrotate 2.2.0

Lockfile written to /Users/sdelfante/chef-repo/policyfiles/company_web.lock.json
Policy revision id: 14a342b5d2134516984b878c1f6be83a51427e6f84594c4dd9e57456b1312582
```

## GL: Push the Policyfile to Chef Infra Server



```
$ chef push prod company_web.lock.json
```

```
Uploading policy company_web (14a342b5d2) to policy group prod
Using apache 0.1.0 (1388ab3a)
Using chef-client 11.2.0 (0b49a3a8)
Using company_web 0.1.0 (c1b26cb5)
Using cron 6.2.1 (08676b5c)
Using logrotate 2.2.0 (53e09234)
Using mychef_client 0.1.0 (10d082a4)
Using myiis 0.2.1 (cd0db3ed)
Uploaded myusers 0.1.0 (bebccce3)
```



## GL: Create Users from a Data Bag

*Dynamically search through the Chef Server under the 'users' index to create users*

**Objective:**

- ✓ Generate the 'myusers' cookbook
- ✓ Create users based on data bag contents within users recipe
- ✓ Add the 'myusers' cookbook to company\_web policyfile's run list
- ✓ Update and push the policyfile to the Chef Infra Server
- ❑ Converge web nodes

## GL: Converge All Web Nodes



```
$ knife ssh "name:apache_web" -x USER -P PWD "sudo chef-client"
```

```
$ knife winrm "name:iis_web" -a cloud.public_ipv4 -x USER -P PWD "chef-client"
```

## GL: Check Local Users for Apache Server



```
$ knife ssh "name:apache_web" -x USER -P PWD "cat /etc/passwd"
....
ec2-34-196-104-17.compute-1.amazonaws.com
chrony:x:998:995::/var/lib/chrony:/sbin/nologin
ec2-34-196-104-17.compute-1.amazonaws.com centos:x:1000:1000:Cloud
User:/home/centos:/bin/bash
ec2-34-196-104-17.compute-1.amazonaws.com chef:x:1001:1001:ChefDK
User:/home/chef:/bin/bash
ec2-34-196-104-17.compute-1.amazonaws.com tss:x:59:59:Account used by
the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
ec2-34-196-104-17.compute-1.amazonaws.com
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
ec2-34-196-104-17.compute-1.amazonaws.com centos_user:x:1002:1002:I am
a centos_user user:/home/centos_user:/bin/bash
```

We can verify that the centos\_user was created on our apache\_web server by looking at the passwd file.

## GL: Check Local Users for IIS Server



```
> knife winrm "name:iis_web" -a cloud.public_ipv4 -x USER -P PWD "net user windows_user"
```

34.195.38.226 User name	windows_user
34.195.38.226 Full Name	
34.195.38.226 Comment	I am a windows user
34.195.38.226 User's comment	
34.195.38.226 Country/region code	000 (System Default)
34.195.38.226 Account active	Yes
34.195.38.226 Account expires	Never
34.195.38.226	
34.195.38.226 Password last set	8/7/2020 5:45:27 PM
34.195.38.226 Password expires	Never
34.195.38.226 Password changeable	8/7/2020 5:45:27 PM
34.195.38.226 Password required	Yes
34.195.38.226 User may change password	Yes
34.195.38.226	
34.195.38.226 Workstations allowed	All

And we make sure that the windows\_user was created by running 'net user windows\_user'.



## GL: Create Users from a Data Bag

*Dynamically search through the Chef Server under the 'users' index to create users*

**Objective:**

- ✓ Generate the 'myusers' cookbook
- ✓ Create users based on data bag contents within users recipe
- ✓ Add the 'myusers' cookbook to company\_web policyfile's run list
- ✓ Update and push the policyfile to the Chef Infra Server
- ✓ Converge web nodes

Instructor Note:

The GitHub repo for the 'myusers' cookbook can be found at:

<https://github.com/chef-training/devops-cookbooks-myusers.git>



## Optional Lab: Managing Groups

- Create a 'groups' data bag on your Chef Server
- Create centos\_group.rb and windows\_group.rb files with: "id", "members", and "platform" fields and upload to the 'groups' data bag
- Create a 'groups.rb' recipe file that creates the group with corresponding members based on the node's platform and include this recipe in the default.rb recipe
- Update the metadata.rb file with a minor version change
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes
- Verify the new group on apache\_web with: `cat /etc/group`
- Verify the new group on iis\_web with: `net localgroup GROUP_NAME`

It's now your turn to create groups on our web servers as well.

Instructor Note: You can skip this Managing Groups lab if you need additional time. It's basically the same steps as we did for the Users section of this module.

## Lab: Create a `data_bags/groups` Directory



```
$ cd ~/chef-repo
$ mkdir data_bags/groups
```

Create a 'groups' directory for us to store our .json files associated with our groups.

## Lab: Create groups Data Bag on Chef Server



```
$ knife data bag create groups
```

```
Created data_bag[groups]
```

Create the 'groups' data bag on the Chef Server with this command.



## Optional Lab: Managing Groups

- ✓ Create a 'groups' data bag on your Chef Server
- Create centos\_group.json and windows\_group.json files with: "id", "members", and "platform" fields and upload to the 'groups' data bag
- Create a 'groups.rb' recipe file that creates the group with corresponding members based on the node's platform and include this recipe in the default.rb recipe
- Update the metadata.rb file with a minor version change
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes
- Verify the new group on apache\_web with: `cat /etc/group`
- Verify the new group on iis\_web with: `net localgroup GROUP_NAME`

## Lab: Create centos\_group.json



~/chef-repo/data\_bags/groups/centos\_group.json

```
{
 "id": "centos_group",
 "members": ["centos_user"],
 "platform": "centos"
}
```

Define a centos\_group that will include our centos\_user as a member. Note that this 'members' key uses an array for the values so if we had more users we would simply add them each to the array.

## Lab: Create windows\_group.json

```
└─ ~/chef-repo/data_bags/groups/windows_group.json
```

```
{
 "id": "windows_group",
 "members": ["windows_user"],
 "platform": "windows"
}
```

And do the same for a windows\_group with the windows\_user as a member.

## Lab: Upload Data Bag Items to Chef Server



```
$ knife data bag from file groups data_bags/groups/centos_group.json
data_bags/groups/windows_group.json
```

```
Updated data_bag_item[groups::centos_group]
Updated data_bag_item[groups::windows_group]
```

Upload these new data bag items to the chef server with a 'knife data bag from file'

knife data bag from file groups data\_bags/groups/centos\_group.json  
data\_bags/groups/windows\_group.json

## Lab: Validate Chef Server Received Items



```
$ knife data bag show groups
```

```
centos_group
windows_group
```

Running a 'knife data bag show' on our groups data bag will validate that our new groups have been added as data bag items.

## Lab: View details of centos\_group



```
$ knife data bag show groups centos_group
```

```
id: centos_group
members: centos_user
platform: centos
```

We again can take a closer look at a single data bag item by specifying the data bag and the specific item.

## Lab: Return groups with “platform:windows”



```
$ knife search groups "platform:windows"
```

```
1 items found
```

```
chef_type: data_bag_item
data_bag: groups
id: windows_group
members: windows_user
platform: windows
```

Search will return to us subsets of data based on what we use as search criteria. In this case everything with the platform of Windows.



## Optional Lab: Managing Groups

- Create a 'groups.rb' recipe file that creates the group with corresponding members based on the node's platform and include this recipe in the default.rb recipe
- Update the metadata.rb file with a minor version change
- Update and push the policyfile to the Chef Infra Server
- Converge web nodes
- Verify the new group on apache\_web with: `cat /etc/group`
- Verify the new group on iis\_web with: `net localgroup GROUP_NAME`

It's now your turn to create groups on our web servers as well.

Instructor Note: You can skip this Managing Groups lab if you need additional time. It's basically the same steps as we did for the Users section of this module.

## Lab: Create groups recipe

```
~/chef-repo/cookbooks/myusers/recipes/groups.rb

system_groups = search("groups", "platform:#{{node['platform']}}")

system_groups.each do |group_data|
 group group_data['id'] do
 members group_data['members']
 action :create
 end
end
```

Very similar to our 'users' recipe, we use the search method to return every group that has the same platform as the node itself and add it to a 'system\_groups' variable. Calling the .each method on 'system\_groups' loops through every group that has been added and creates a new group passing in the members to this new group.

## Lab: Update the default recipe

~/chef-repo/cookbooks/myusers/recipes/default.rb

```

Cookbook:: myusers
Recipe:: default

Copyright:: 2018, The Authors, All Rights Reserved.

include_recipe 'myusers::users'
include_recipe 'myusers::groups'
```

We include this 'groups' recipe within the default.rb recipe file.

## Lab: Version the myusers metadata.rb



~/chef-repo/cookbooks/myusers/metadata.rb

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

This is considered a minor change to the cookbook and is reflected in the metadata.rb file.



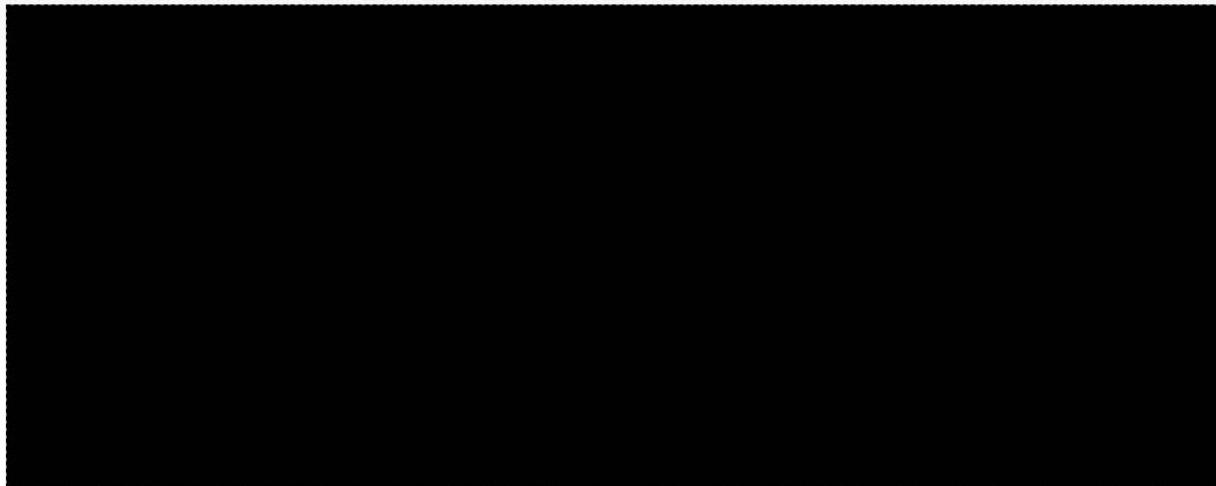
## Optional Lab: Managing Groups

- ✓ Create a 'groups.rb' recipe file that creates the group with corresponding members based on the node's platform and include this recipe in the default.rb recipe
- ✓ Update the metadata.rb file with a minor version change
  - Update and push the policyfile to the Chef Infra Server
  - Converge web nodes
  - Verify the new group on apache\_web with: `cat /etc/group`
  - Verify the new group on iis\_web with: `net localgroup GROUP_NAME`

## GL: Ensure You are in chef-repo/policyfiles



```
$ cd ~/chef-repo/policyfiles
```



## GL: Update the Policyfile



```
$ chef update company_web.rb
```

```
Attributes already up to date
Building policy company_web
Expanded run list: recipe[mychef_client::default], recipe[company_web::default],
recipe[myusers::default]
Caching Cookbooks...
Installing company_web >= 0.0.0 from path
Installing myiis >= 0.0.0 from path
Installing apache >= 0.0.0 from path
Installing mychef_client >= 0.0.0 from path
Installing myusers >= 0.0.0 from path
Using chef-client 11.2.0
Using cron 6.2.1
Using logrotate 2.2.0

Lockfile written to /Users/sdelfante/chef-repo/policyfiles/company_web.lock.json
Policy revision id: 2fb379d0bf8eedd1c43efcc3c57b32424b1dbd9fa27850e79daba3a71c5e672
```

## GL: Push the Policyfile to Chef Infra Server



```
$ chef push prod company_web.lock.json
```

```
Uploading policy company_web (2fb379d0bf) to policy group prod
Using apache 0.1.0 (1388ab3a)
Using chef-client 11.2.0 (0b49a3a8)
Using company_web 0.1.0 (c1b26cb5)
Using cron 6.2.1 (08676b5c)
Using logrotate 2.2.0 (53e09234)
Using mychef_client 0.1.0 (10d082a4)
Using myiis 0.2.1 (cd0db3ed)
Uploaded myusers 0.2.0 (0fb96da2)
```

## GL: Converge All Web Nodes



```
$ knife ssh "name:apache_web" -x USER -P PWD "sudo chef-client"
```

```
$ knife winrm "name:iis_web" -a cloud.public_ipv4 -x USER -P PWD "chef-client"
```

## GL: Check Local Users for Apache Server



```
$ knife ssh "name:apache_web" -x USER -P PWD "cat /etc/passwd"
....
ec2-34-196-104-17.compute-1.amazonaws.com
chrony:x:998:995::/var/lib/chrony:/sbin/nologin
ec2-34-196-104-17.compute-1.amazonaws.com centos:x:1000:1000:Cloud
User:/home/centos:/bin/bash
ec2-34-196-104-17.compute-1.amazonaws.com chef:x:1001:1001:ChefDK
User:/home/chef:/bin/bash
ec2-34-196-104-17.compute-1.amazonaws.com tss:x:59:59:Account used by
the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
ec2-34-196-104-17.compute-1.amazonaws.com
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
ec2-34-196-104-17.compute-1.amazonaws.com centos_user:x:1002:1002:I am
a centos_user user:/home/centos_user:/bin/bash
```

We can verify that the centos\_user was created on our apache\_web server by looking at the passwd file.

## GL: Check Local Users for IIS Server



```
> knife winrm "name:iis_web" -a cloud.public_ipv4 -x USER -P PWD "net user windows_user"
```

34.195.38.226 User name	windows_user
34.195.38.226 Full Name	
34.195.38.226 Comment	I am a windows user
34.195.38.226 User's comment	
34.195.38.226 Country/region code	000 (System Default)
34.195.38.226 Account active	Yes
34.195.38.226 Account expires	Never
34.195.38.226	
34.195.38.226 Password last set	8/7/2020 5:45:27 PM
34.195.38.226 Password expires	Never
34.195.38.226 Password changeable	8/7/2020 5:45:27 PM
34.195.38.226 Password required	Yes
34.195.38.226 User may change password	Yes
34.195.38.226	
34.195.38.226 Workstations allowed	All

And we make sure that the windows\_user was created by running 'net user windows\_user'.

## Check Local Groups (Linux)



```
$ knife ssh "name:apache_web" -x USER -P PWD "cat /etc/group"
```

```
....
ec2-52-90-49-196.compute-1.amazonaws.com sshd:x:74:
ec2-52-90-49-196.compute-1.amazonaws.com chef:x:500:
ec2-52-90-49-196.compute-1.amazonaws.com dbus:x:81:
ec2-52-90-49-196.compute-1.amazonaws.com cgrid:x:499:
ec2-52-90-49-196.compute-1.amazonaws.com dockerroot:x:498:chef
ec2-52-90-49-196.compute-1.amazonaws.com centos_group:x:501:centos_user
```

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.

## Check Local Groups (Windows)



```
> knife winrm "name:iis_web" -a cloud.public_ipv4 -x USER -P PWD "net localgroup windows_group"
```

```
52.90.49.196 Alias name windows_group
52.90.49.196 Comment
52.90.49.196
52.90.49.196 Members
52.90.49.196
52.90.49.196 -----

52.90.49.196 windows_user
52.90.49.196 The command completed successfully.
```

To verify that everything is working the same as before, run 'knife winrm' for both of these nodes. In this instance the query syntax is going to find all nodes with the role set to web.



## Optional Lab: Managing Groups

- ✓ Create a 'groups.rb' recipe file that creates the group with corresponding members based on the node's platform and include this recipe in the default.rb recipe
- ✓ Update the metadata.rb file with a minor version change
- ✓ Update and push the policyfile to the Chef Infra Server
- ✓ Converge web nodes
- ✓ Verify the new group on apache\_web with: `cat /etc/group`
- ✓ Verify the new group on iis\_web with: `net localgroup GROUP_NAME`



## Review Questions

1. When should we utilize data bags instead of node attributes?
2. When creating a new data bag, what index on the Chef server does the data bag get added to?

1. When we need to store information that is not tied to a single node, such as user names for all CentOS nodes in your infrastructure.
2. It depends on the data type. Users are in the users index...groups in the groups index.



## Q&A

What questions can we help you answer?



©2020 Chef Software Inc.