



CHEFTM

Chef Training Services

DevOps Foundation

Participant Guide



The slide features a blue and white abstract background with wavy patterns. In the top left corner, there's a graphic of overlapping blue and teal rectangles. The title "DevOps Foundation" is prominently displayed in orange at the top center. Below it, the word "Introduction" is written in a smaller, dark font. At the bottom center, the text "Course v1.1.2" is visible. A horizontal orange line separates the main content from the footer. The footer contains three items: a copyright notice "©2018 Chef Software Inc.", a page number "1-1", and the Chef logo (a circular icon with the word "CHEF" below it).

DevOps Foundation

Introduction

Course v1.1.2

©2018 Chef Software Inc.

1-1

CHEF

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

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

Instructor Note: Be sure to read Appendix Z for training lab set up notes and additional instructor notes



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.

Chef



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

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

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

Chef



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

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

A great way to learn Chef is to use Chef

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

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

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
- Add multiple nodes to be managed by a Chef Server
- Manage the deployment of cookbooks to nodes with Roles and Environments
- Manage user and group data with data bags

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

- ❖ What is DevOps?
- ❖ Using Chef Resources
- ❖ Building Chef Cookbooks
- ❖ Collecting details about the system
- ❖ Managing data with templates

Agenda: Day 2

- ❖ Set up an Apache web server
- ❖ Install the ChefDK and sign up for a Managed Chef account
- ❖ Communicate with a Chef Server
- ❖ Attribute Files and Dependencies
- ❖ Community cookbooks
- ❖ Manage multiple nodes

Agenda: Day 3

- ❖ Roles
- ❖ Use Search within a recipe
- ❖ Set up chef-client to run as a service/task
- ❖ Data Bags
- ❖ Environments
- ❖ Further Resources



Introductions

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

- ✓ Introduce ourselves
- ✓ Introduce this training experience



Pre-built Workstation

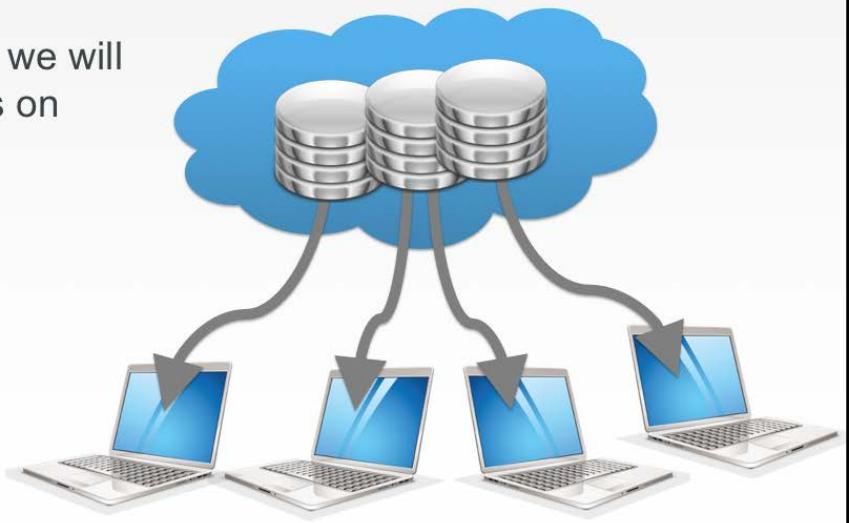
We will provide for you a workstation with all the tools installed.

- 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.

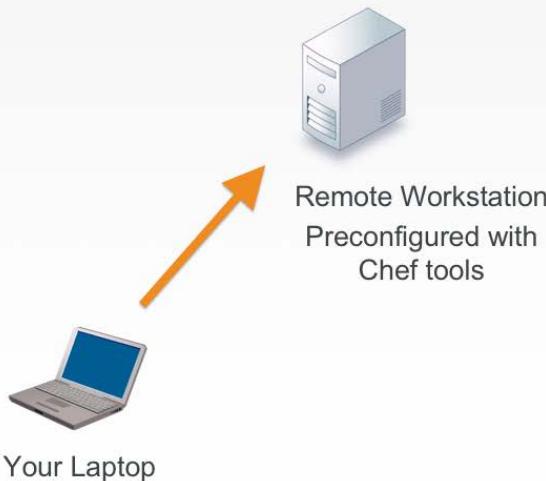
Setting up Your Workstation

At the beginning of 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: Cod3Can!

Pre-built Workstation



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

Objective:

- ✓ Login to the Remote Workstation



CHEFTM

©2018 Chef Software Inc.

What is DevOps?

Blending Skill Sets to Deploy more Quickly



What is DevOps?

- Breaks down the boundary between the group that develops code and applications from the group that deploys, manages and supports these applications.
- Manages and supports collaboration between developer and operations teams throughout the service lifecycle, hence the term DevOps

What is DevOps, not just as a coin-term but why is it valuable? The term DevOps means a breaking down of barriers between roles that were previously separate and siloed. Just as the Agile movement set smaller goals to deploy more quickly, DevOps aims at the same task, yet this time it's a blending of roles and skill sets. Specifically, it attempts to break down the barriers between the roles that have been traditionally assigned to developers and operations. It aims to create an environment where these teams collaborate with one another throughout the lifecycle of an application or service.



What is DevOps?

- A methodology and culture. It will require a change in mindset with new tools and skillsets.
- It is not a product or buzzword. It will take time and dedication to implement!

At its core DevOps is a methodology and a culture. It's not just a buzzword that a boss or innovator uses to persuade people to develop or move faster toward deployment. It's a change in the cultural norm of an organization and this will take time and a lot of energy to manifest.



The Pre-DevOps World

- **Siloed**
 - Not much communication or collaboration between development and operations teams.
- **Expensive / slow**
 - Releases take much longer and require more work
- **Trust Issues**
 - Difficult for developers to gain access to environments for testing of code

In the world before DevOps there were some serious obstacles to reaching the goal of promoting code to your production environment. Because communication and collaboration between development and operations teams was minimal at best, development of necessary software and having it work correctly in environments very far removed from the development team took an extremely long time. Releases might take months rather than days. Trust issues prevented developers from gaining access to environments to test their code and required lengthy procedures to only have a limited amount of time to test their code.



DevOps Understanding

*Let's better understand **why** the DevOps strategy is prevalent.*

Objective:

- Key Components and Benefits to DevOps
- Stages of DevOps
- Collaboration: Development & Operations

Let's take a look at a few key points surrounding DevOps

Key Components to DevOps



- Automation
 - QA (quality assurance) and testing automation
 - Infrastructure automation and management
- Continuous Integration
 - Quickly and efficiently integrate new code into the main body of code for release
- Continuous Delivery
 - Automate the process for deploying code
- Monitoring
 - Perpetual monitoring of changes being pushed into the production environment

A DevOps cultural shift within an organization happens most often from a top down approach. Its key aspect is to remove the boundaries between the group that writes and develops code and applications from the group that deploys, manages and supports the same code and applications during uptime. It doesn't make sense to have the two groups separated and not collaborating. Operations needs to know how code and applications are developed to help them manage issues as they arise. Developers need to know about integrations that are necessary in the operations context. Code or applications can not be pushed into production unless QA signs off, so developers need to know the criteria that need to be met when creating new features or functionality.



Benefits of DevOps Methodology

- Empowerment of developers to launch and destroy VMs for testing
- Separation from production environment to sandbox new feature/application
- Reduction of time needed for software development
- Operational understanding of how the application/code will affect the ecosystem

In an older modality, developers would work toward a product goal, some feature of code that needed to be created based on a manager's or client's needs. In DevOps, the goals haven't changed, but the manner in which they are reached has.

Provisioning a new machine takes time in a legacy organization, managerial approval, cross party monitoring, and a timeframe for the developer to test their finished code. This is not a well prescribed method for a quick, effective solution for developers to see the end result of their new feature. Having sandbox environments that are easily accessible for developers will reduce the amount of time for software development as well as facilitate an understanding of how your code will affect the ecosystem.



Benefits of DevOps Methodology

- Increased deployment frequencies
- More dependable releases
- Reduction of time needed for software development
- Increased business value

With a DevOps methodology you will see that you will have more frequent deployments that are more reliable. With this you will see an increased business value by getting your product to the customer faster with better reliability and uptime

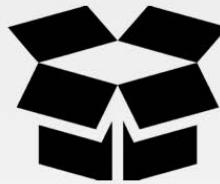


DevOps Understanding

*Let's better understand **why** the DevOps strategy is prevalent.*

Objective:

- ✓ Key Components and Benefits to DevOps
- ❑ Stages of DevOps
- ❑ Collaboration: Development & Operations



Stages of DevOps

- ❖ **Code:** development, review, management, merge (Git)
- ❖ **Build:** continuous integration, build status (Chef Automate, Jenkins)
- ❖ **Test:** continuous testing, automated testing (InSpec)
- ❖ **Release:** change management, automated release (Chef Automate, Jenkins)
- ❖ **Deploy:** Infrastructure as Code (Chef)
- ❖ **Monitor:** Performance and end-user experience (Sensu, Splunk)

There are some defined stages of DevOps that feed into one another in the lifecycle of an application. The 'code' stage is where developers collaborate on software projects and use version control systems such as Git to merge into the main body of code. Build is where we start applying our code in a stand alone form. Testing should be happening throughout but it is key to have automated tests running before the release stage where we can automate the release of the code we have developed. In the deploy stage we apply our code using a tool like Chef. Finally, we want to monitor our production environment with tools like Sensu and Splunk.



DevOps Understanding

*Let's better understand **why** the DevOps strategy is prevalent.*

Objective:

- ✓ Key Components and Benefits to DevOps
- ✓ Stages of DevOps
- ❑ Collaboration: Development & Operations

What does collaboration mean with a tool like Chef?



Infrastructure As Code

- Silos between operations and deployment are broken, and teams work seamlessly in a continuous automation cycle
- Chef takes the process of software deployment and converts it into code
- Infrastructure processes will be treated just like software: they will be stored in a central repository transparent to the entire team and are versionable

Traditionally, software development worked in cycles: coding, testing and deployment. Today's software development cycles are infinitely more complex. Updates to consumer-facing programs happen in real time and on a whole suite of different machines. Such a dramatic shift necessitates DevOps environments in which the silos between operations and deployment are broken, and teams work seamlessly in a continuous automation cycle. Enter infrastructure as code.

Chef takes the process of software deployment and converts it into code—a set of commands or “recipes” to be followed.

By using infrastructure as code, Chef simplifies the work of software deployment.

Each iteration of software deployment is defined as code and is therefore versionable and open for your team to track.



Who is Development?

Commonly the development team is considered to be the software developers creating the applications. However, we should include everyone involved in the development process such as:

- QA
- Solution Architects
- UI designers
- System Analysts

Much of the time the development team is considered to be those software developers coding away. But really the development team should be considered everyone involved in the development process. The Dev team are the 'makers' of the applications or services.



Who is Operations?

Operations acts as a blanket term for those involved in the use of the applications created by the development team. This includes:

- System Administrators
- Network Engineers
- Database Administrators
- Security Analysts

Operations could be considered the team who are utilizing the tools created by Dev.



Chef Requires Collaboration!

To adopt a DevOps methodology, it's important to know what is involved when using Chef for both development and operations teams



Development



Operations

To really become effective with Chef, there needs to be a collaborative effort between developer and operations teams. This is because we are describing our infrastructure as code. The realm that traditionally has been proscribed to operations has now come into the world of developers and we now manage our infrastructure needs through the Chef DSL. Although there's not always a clear cut line between the tasks accomplished by each team, as we go through the class we will be working through labs that might fall into the realms of development or operations and we will differentiate this with these symbols. We need to know what is involved with each others teams so that we can better collaborate with them. Spend a day in someone else's shoes so to say.



DevOps Understanding

*Let's better understand **why** the DevOps strategy is prevalent.*

Objective:

- ✓ Key Components and Benefits to DevOps
- ✓ Stages of DevOps
- ✓ Collaboration: Development & Operations



Discussion

Is your team utilizing or moving towards practicing a DevOps approach?

If so, what are some of the challenges that this has presented?



CHEFTM

©2018 Chef Software Inc.

Chef Resources

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.

Example: powershell_script

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

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

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

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.



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**



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.



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: Create and Open a Recipe File



> dir

| Mode | LastWriteTime | Length | Name |
|-------|-------------------|--------|-------------|
| d---- | 2/16/2017 6:31 PM | | .atom |
| d---- | 2/16/2017 6:20 PM | | .berkshelf |
| d---- | 2/16/2017 6:21 PM | | .chef |
| d---- | 2/16/2017 6:23 PM | | .kitchen |
| d---- | 2/16/2017 6:22 PM | | .VirtualBox |
| d---- | 2/16/2017 6:21 PM | | chef |
| d-r-- | 2/16/2017 6:16 PM | | Contacts |
| d---- | 2/16/2017 6:44 PM | | cookbooks |
| d-r-- | 2/16/2017 6:32 PM | | Desktop |
| d-r-- | 2/16/2017 6:21 PM | | Documents |
| d-r-- | 2/16/2017 6:16 PM | | Downloads |

When opening up Powershell you are dropped into the 'Administrator' directory. This is our working directory for today.

GL: Create and Open a Recipe File



```
> atom hello.rb
```



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
```

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



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

©2018 Chef Software Inc.

3-16



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.



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 --local-mode hello.rb
```

```
Converging 1 resources
Recipe: @recipe_files::C:/Users/Administrator/hello.rb
  * file[C:\hello.txt] action create[2017-10-30T19:23:52+00:00] INFO: Processing
    file[C:\hello.txt] action create (@reci
    pe_files::C:/Users/Administrator/hello.rb line 1)
[2017-10-30T19:23:52+00:00] INFO: file[C:\hello.txt] created file C:\hello.txt
  - create new file C:\hello.txt[2017-10-30T19:23:52+00:00] INFO:
    file[C:\hello.txt] updated file contents C:\hello.txt

  - update content in file C:\hello.txt from none to 4ae7c3
    --- C:\hello.txt      2017-10-30 19:23:52.000000000 +0000
    +++ C:\chef-hello20171030-756-1kj8npo.txt      2017-10-30 19:23:52.000000000
```



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!
```



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?



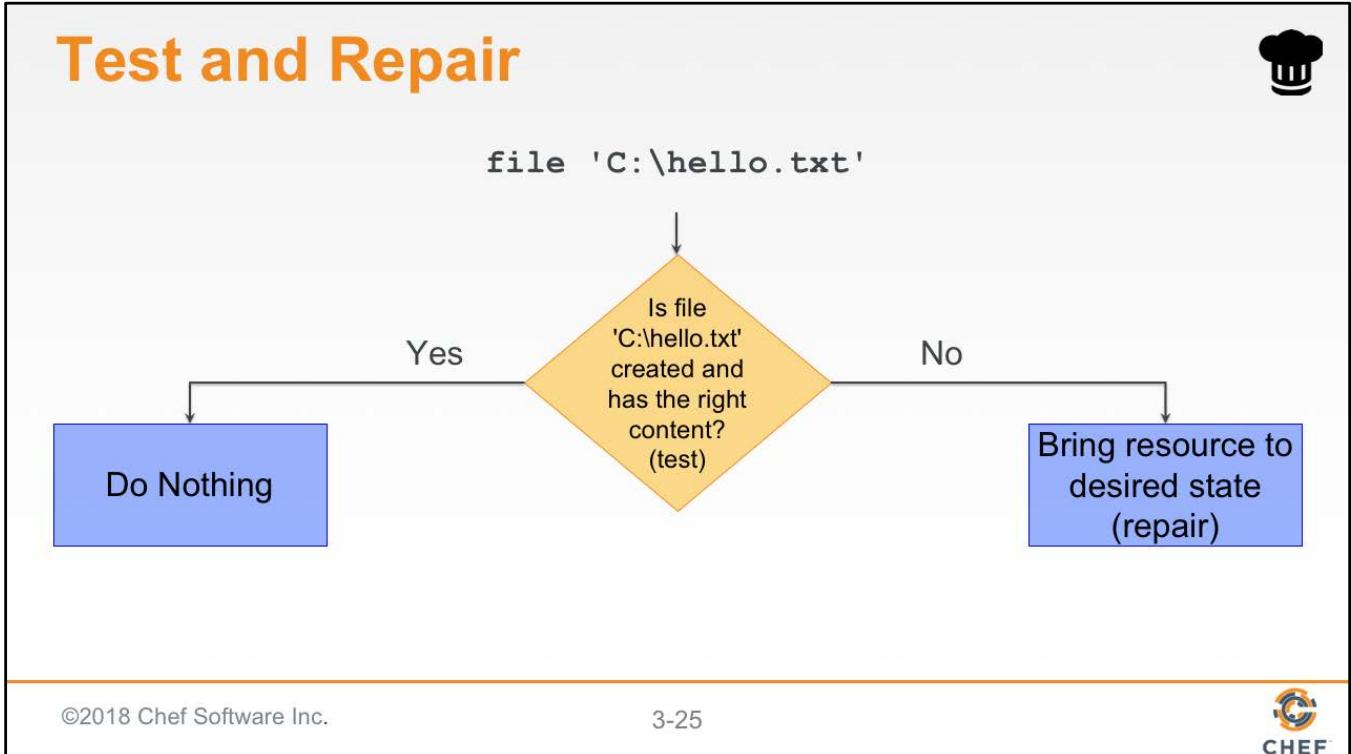
Test and Repair

`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: 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
```

```
[2017-10-30T19:31:13+00:00] INFO: file[C:\hello.txt] backed up to  
C:/Users/Administrator\.chef\local-mode-cache\backup\h  
ello.txt.chef-20171030193113.498382  
[2017-10-30T19:31:13+00:00] INFO: file[C:\hello.txt] deleted file at  
C:\hello.txt
```

```
    - delete file C:\hello.txt
```

```
[2017-10-30T19:31:13+00:00] INFO: Chef Run complete in 0.059047 seconds
```

```
Running handlers:
```

```
[2017-10-30T19:31:13+00:00] INFO: Running report handlers
```

```
Running handlers complete
```



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.

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
```

```
Converging 1 resources
Recipe: @recipe_files::C:/Users/Administrator/disable-uac.rb
  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]
action create[2017-10-30T19:41:15+00:00]
] INFO: Processing
registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System]
action create (@recipe_f
iles::C:/Users/Administrator/disable-uac.rb line 3)

  - set value {:name=>"EnableLUA", :type=>:dword, :data=>0}
[2017-10-30T19:41:15+00:00] INFO: Chef Run complete in 0.060001 seconds
Running handlers:
[2017-10-30T19:41:15+00:00] INFO: Running report handlers
Running handlers complete
```



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.

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
```

```
Recipe: @recipe_files::C:/Users/Administrator/disable-uac.rb
  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System] action create[2017-10-30T19:44:33+00:00]
    ] INFO: Processing
registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System] action create
(@recipe_files::C:/Users/Administrator/disable-uac.rb line 3)
  (up to date)
  * registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System] action create
create[2017-10-30T19:44:33+00:00]
    ] INFO: Processing
registry_key[HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System] action create
(@recipe_files::C:/Users/Administrator/disable-uac.rb line 12)
      - set value {:name=>"ConsentPromptBehaviorAdmin", :type=>:dword, :data=>0}
[2017-10-30T19:44:34+00:00] INFO: Chef Run complete in 0.063 seconds
```



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"



Discussion

What is a resource?

What are some other possible examples of resources?

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

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

Answer these four questions:

What is a resource?

What are some other possible examples of resources?

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

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

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.



Q&A

What questions can we answer for you?

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

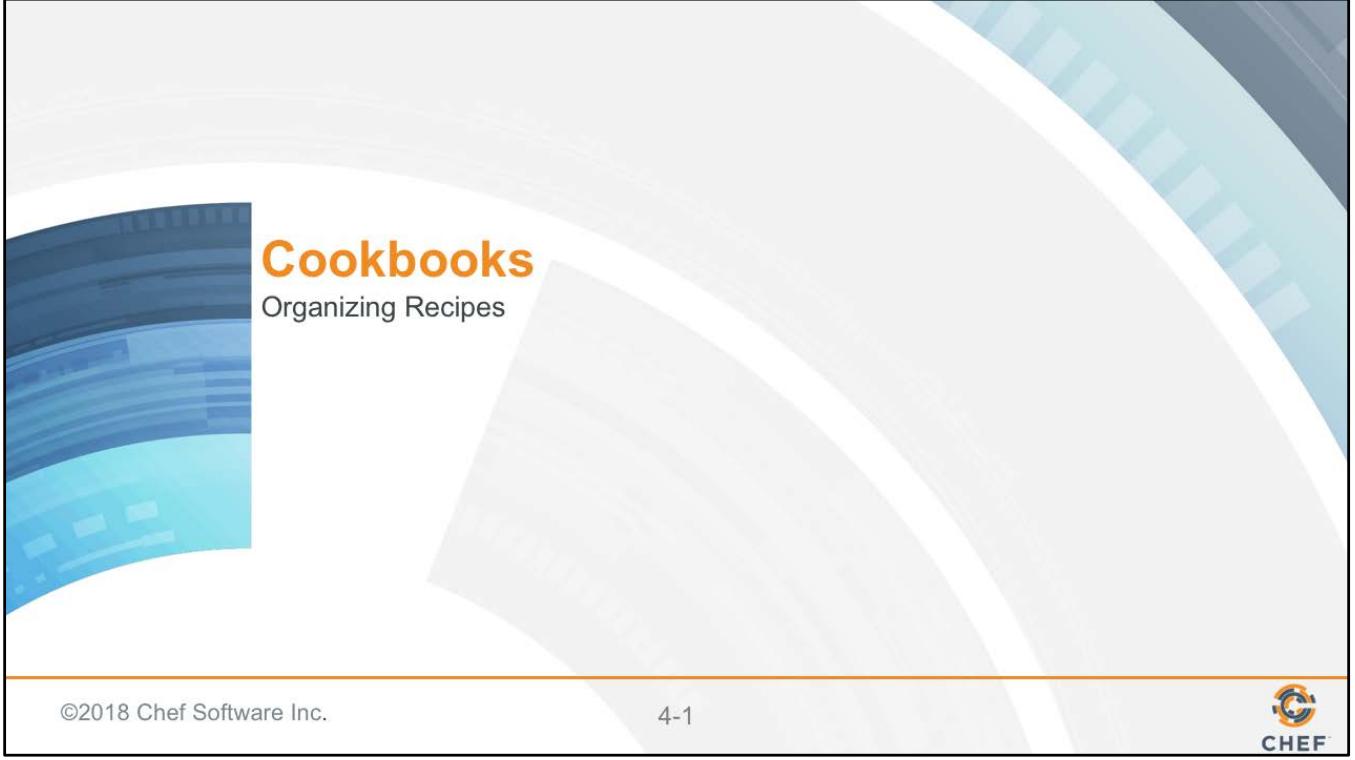
What questions can we answer for you?

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



CHEFTM

©2018 Chef Software Inc.



Cookbooks

Organizing Recipes

Objectives



After completing this module you should be able to:

- Generate a Chef cookbook
- Implement the include_recipe method
- Apply a run-list of recipes to a system
- Define a Chef cookbook that sets up a web server

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.

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

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

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
- 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--- | 3/22/2018 11:10 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  
  
Mode           LastWriteTime     Length Name  
----           -----          -----  
d----        10/23/2017  12:51 AM      .atom  
d----        2/16/2017   6:20 PM      .berkshelf  
d----        10/23/2017  12:48 AM      .chef  
d----        2/16/2017   6:23 PM      .kitchen  
d----        2/16/2017   6:22 PM      .VirtualBox  
d----        2/16/2017   6:21 PM      chef  
d-r--         2/16/2017   6:16 PM      Contacts  
d----        2/16/2017   6:44 PM      cookbooks  
d-r--         10/23/2017 12:40 AM      Desktop  
d-r--         2/16/2017   6:21 PM      Documents  
d-r--         2/16/2017   6:16 PM      Downloads
```



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:
```

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

```
(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,       Use GENERATOR_COOKBOOK_PATH for the
      --generator-cookbook
```



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

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

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

GL: Let's Create a CookbookGL:



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

```
Generating cookbook workstation
```

- Ensuring correct cookbook file content
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content

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

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

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

```
Why not start by writing a test? Tests for the default recipe are stored at:  
test/recipes/default_test.rb
```



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
```



©2018 Chef Software Inc.

4-13



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.rb
```

```
# 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
```



©2018 Chef Software Inc.

4-16



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

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



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
```

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



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

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



GL: 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

GL: Copy the Recipe into the Cookbook



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



From the Home directory, move your 'disable-uac.rb' recipe to the workstation cookbook and place it alongside our default recipe.



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]"
```

Synchronizing Cookbooks:

[2017-10-30T20:09:06+00:00] INFO: Storing updated cookbooks/workstation/recipes/disable-uac.rb in the cache.

- workstation (0.1.0)

Installing Cookbook Gems:

Compiling Cookbooks...

Converging 2 resources

Recipe: workstation::disable-uac

* registry_key[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System]
action create[2017-10-30T20:09:06+00:00]

] INFO: Processing registry_key[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System]



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



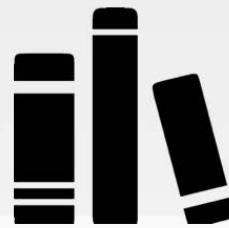
GL: Setting up a Workstation

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>

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

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]"
```

```
Starting Chef Client, version 12.5.1
resolving cookbooks for run list: ["workstation"]
Synchronizing Cookbooks:
  - workstation (0.1.0)
Compiling Cookbooks...
Converging 2 resources

Recipe: workstation::disable-uac
  * registry_key[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System]
    action create (up to date)
  * registry_key[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System]
    action create (up to date)

Running handlers:
```

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>

Lab: Create a Cookbook



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

```
Generating cookbook iis
```

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

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

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



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
```

```
Compiling Cookbooks...
Recipe: code_generator::recipe
  * directory[./cookbooks/iis/spec/unit/recipes] action create (up to date)
  * cookbook_file[./cookbooks/iis/spec/spec_helper.rb] action create_if_missing
    (up to date)
    * template[./cookbooks/iis/spec/unit/recipes/server_spec.rb] action
      create_if_missing
        - create new file ./cookbooks/iis/spec/unit/recipes/server_spec.rb
        - update content in file ./cookbooks/iis/spec/unit/recipes/server_spec.rb
          from none to 93f1e7
          (diff output suppressed by config)
    * template[./cookbooks/iis/recipes/server.rb] action create
      - create new file ./cookbooks/iis/recipes/server.rb
      - update content in file ./cookbooks/iis/recipes/server.rb from none to
```



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: Create 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 ~/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]"
```

```
Starting Chef Client, version 12.13.37
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Compiling Cookbooks...
[2016-03-28T21:20:12+00:00] WARN: Node WIN-14DV1I4A82F.ec2.internal has an empty run
list.
Converging 3 resources
Recipe: @recipe_files::C:/Users/Administrator/cookbooks/iis/recipes/server.rb
  * powershell_script[Install IIS] action run
    - execute "C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe" -NoLogo -
NonInteractive -NoProfile -ExecutionP
  olicy Bypass -InputFormat None -File "C:/Users/ADMINI~1/AppData/Local/Temp/2/chef-
script20160328-2492-q11lef.ps1"
    * file[C:\inetpub\wwwroot\Default.htm] action create
      - create new file C:\inetpub\wwwroot\Default.htm
```



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: 21
                  Content-Type: text/html
                  Date: Mon, 21 Dec 2016 20:59:13 GMT
                  ETag: "954c2066323cd11:0"
                  Last-Modified: Mon, 21 Dec 2016 20:58:52 GMT
                  Server...
```



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>



Discussion

What file would you read first when examining a cookbook? second?

What are the benefits of using the include_recipe method? What are the drawbacks?

What other recipes might you include in the myiis or workstation cookbooks?

How do resources accept multiple actions?

Answer these questions.

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



Q&A

What questions can we answer for you?

- Cookbooks
- Recipes
- Run-lists
- include_recipe method

What questions can we help you answer?

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



CHEFTM

©2018 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>



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.



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
```



©2018 Chef Software Inc.

5-17



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

GL: Running Ohai to Show the IP Address



```
> chai platform
```

```
[  
  "windows"  
]
```



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

GL: Running Ohai to Show the Hostname



```
> ohai hostname
```

```
[  
  "ip-172-31-57-153"  
]
```



GL: Running Ohai to Show the Memory



> ohai memory

```
{  
  "swap": {  
    "total": "0kB",  
    "free": "0kB"  
  },  
  "total": "604308kB",  
  "free": "297940kB"  
}
```



GL: Running Ohai to Show the Total Memory



```
> ohai memory/total
```

```
[  
  "604308kB"  
]
```



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": 4,  
    "vendor_id": "GenuineIntel",  
    "family": 1,  
    "model": "14857",  
    "stepping": "9",  
    "physical_id": "CPU0",  
    "model_name": "Intel64 Family 6 Model 58 Stepping 9",  
    "mhz": "3201"  
  },  
  "total": 4,
```



GL: Running Ohai to Show the First CPU



```
> ohai cpu/0
```

```
{
  "cores": 4,
  "vendor_id": "GenuineIntel",
  "family": 1,
  "model": "14857",
  "stepping": "9",
  "physical_id": "CPU0",
  "model_name": "Intel64 Family 6 Model 58 Stepping 9",
  "mhz": "3201"
}
```



GL: Running Ohai to Show the First CPU MHz



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

```
[  
  "3201"  
]
```





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.

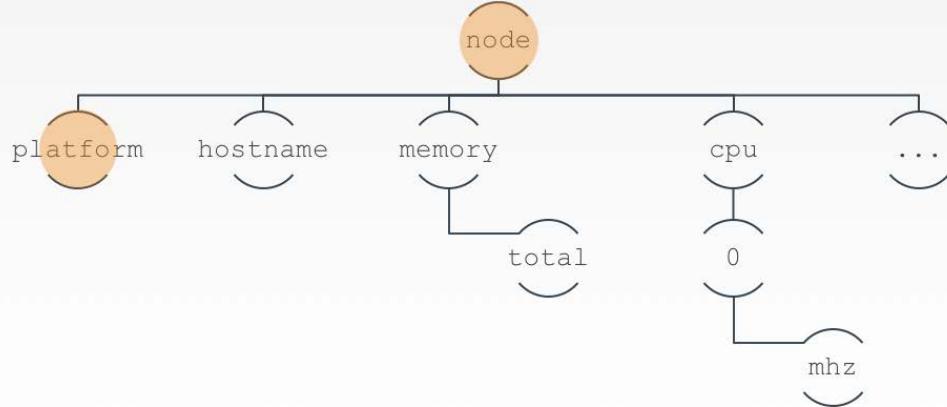
<http://docs.chef.io/nodes.html#attributes>

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.

The Node



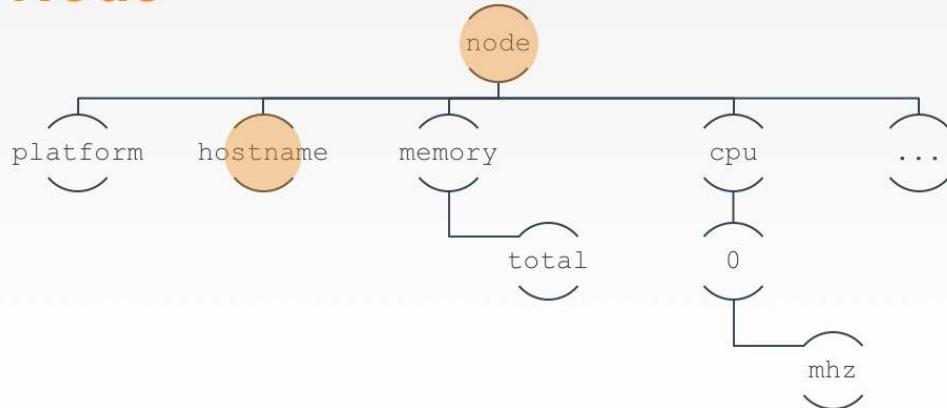
```
CLI: ohai platform
```

```
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



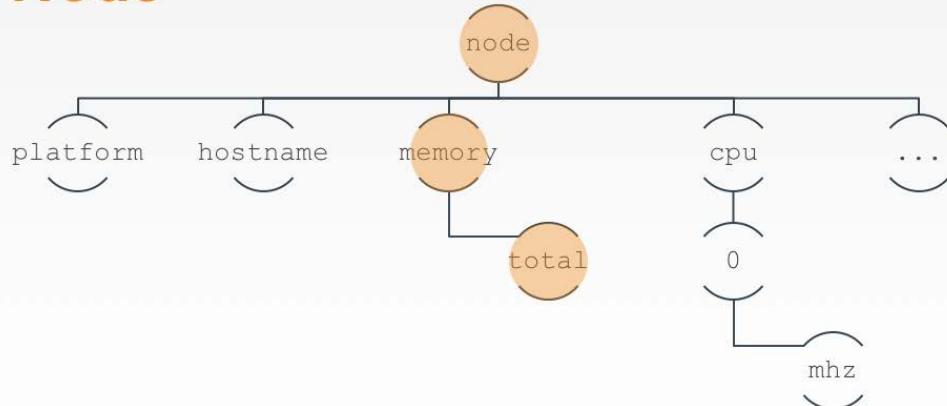
```
CLI: ohai hostname
```

```
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
```

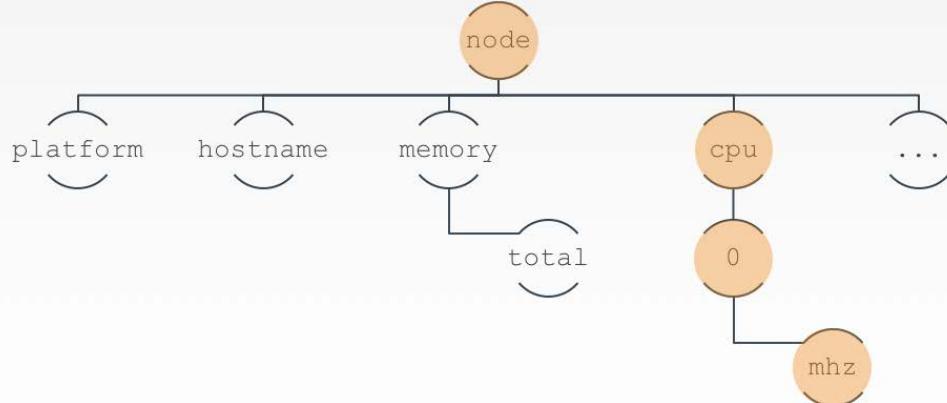
```
RECIPE: node['memory']['total']
```

```
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

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



String Interpolation

```
I have 4 apples
```

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

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

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



String Interpolation

```
I have 4 apples
```

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





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

©2018 Chef Software Inc.

5-36



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>

©2018 Chef Software Inc.

5-37



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?

©2018 Chef Software Inc.

5-38



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'
```



©2018 Chef Software Inc.

5-39



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]"
```

```
[2014-12-23T22:45:48+00:00] WARN: No config file found or specified on command  
line, using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["myiis"]  
Synchronizing Cookbooks:  
  - myiis (0.2.0)  
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
```



©2018 Chef Software Inc.

5-41



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
```



©2018 Chef Software Inc.

5-42



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>



Discussion

What is the node object and when is this generated?

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

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

Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.



Q&A

What questions can we help you answer?

- 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.



CHEFTM

©2018 Chef Software Inc.

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 how to understand when to use a template resource, create a template file, use ERB tags to display node data in a template, define a template resource.



Cleaner Recipes

In the last section we updated our ‘myiis’ web server cookbook to display information about our node.

We added this 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
```

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.

Let's talk about some ways in which this method of keeping the content within the recipe will be problematic as we make more changes.



Double Quotes Close Double Quotes

Double quoted strings are terminated by double quotes.

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



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

If any of the text that we paste into this content field has double quotes it is going to have to be escaped. This can possibly become a problem if we wanted to add some additional style or design to our splash page.



Backslash

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

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



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

Here is the fixed version where all the double-quotes within the string are escaped to ensure that Ruby will read this as a single string of characters.



Backslash

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

"Root Path: C:\\\"



That also brings up an issue with the backslash character. This is particularly important on Windows where the file separator is often represented with a backslash character. You will need to keep an eye out for backslash characters because backslash characters are now the escape character in a double-quoted string.

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



Backslash

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

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

Unexpected Formatting

```
file '/etc/motd' do
  content 'This is the first line of the file.
    This is the second line. If I try and line it up...
'
end
```

Indented; better for humans to read

Placing the terminating quote on a new line
makes it easier to find

This is the first line of the file.

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

Unexpected indentation in the resulting file

[

Unexpected new line in resulting file

©2018 Chef Software Inc.

6-9



It is important to note that the file content may have some important formatting that might be easily overlooked when working with the content in a recipe file. For some configuration files that may require particular formatting (e.g. tabs; spaces; certain indentation) this could cause your configuration files to be read incorrectly causing your applications depending on that configuration to fail.

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

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



Manual Editing

This process 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.



Cleaner Recipes

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

Objective:

- Decide which resource will help us address this issue

To solve this problem, we need to read up on the file resource more or see if Chef provides alternatives.



Let's Check the Docs...

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

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

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

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

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

Let's start with the `remote_file` resource.



cookbook_file

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

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

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

Demo: cookbook_file's Source Match Up

```
> tree /f cookbooks\myiis\files\default
Folder PATH listing
Volume serial number is B04A-119C
C:\users\Administrator\cookbooks\myiis\files\default
    Default.htm

No subfolders exist
```

```
cookbook_file 'C:\inetpub\wwwroot\Default.htm' do
  source 'Default.htm'
end
```

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

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



remote_file

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

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

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

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



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

Let's explore templates.

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

Demo: 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



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-myis.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
```

```
cookbooks/myiis/templates/
|   Default.htm.erb
|
└── default
```



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>

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 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>
```

©2018 Chef Software Inc.

6-34



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.



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

```
~\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
  source 'Default.htm.erb'
end
```



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



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]"
```

```
[2017-08-30T19:48:15+00:00] WARN: No config file found or specified on command line,  
using command line options.  
Starting Chef Client, version 12.13.37  
resolving cookbooks for run list: ["myiis"]  
Synchronizing Cookbooks:  
  - myiis (0.2.0)  
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 ...
```



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: 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>



Discussion

What is the benefit of using a template over defining the content within a recipe?

What are the drawbacks using a template over defining the content within a recipe?

What do each of the ERB tags accomplish?

Answer these questions.

With your answers, turn to another person and alternate asking each other these questions and sharing your answers.



Q&A

What questions can we help you answer?

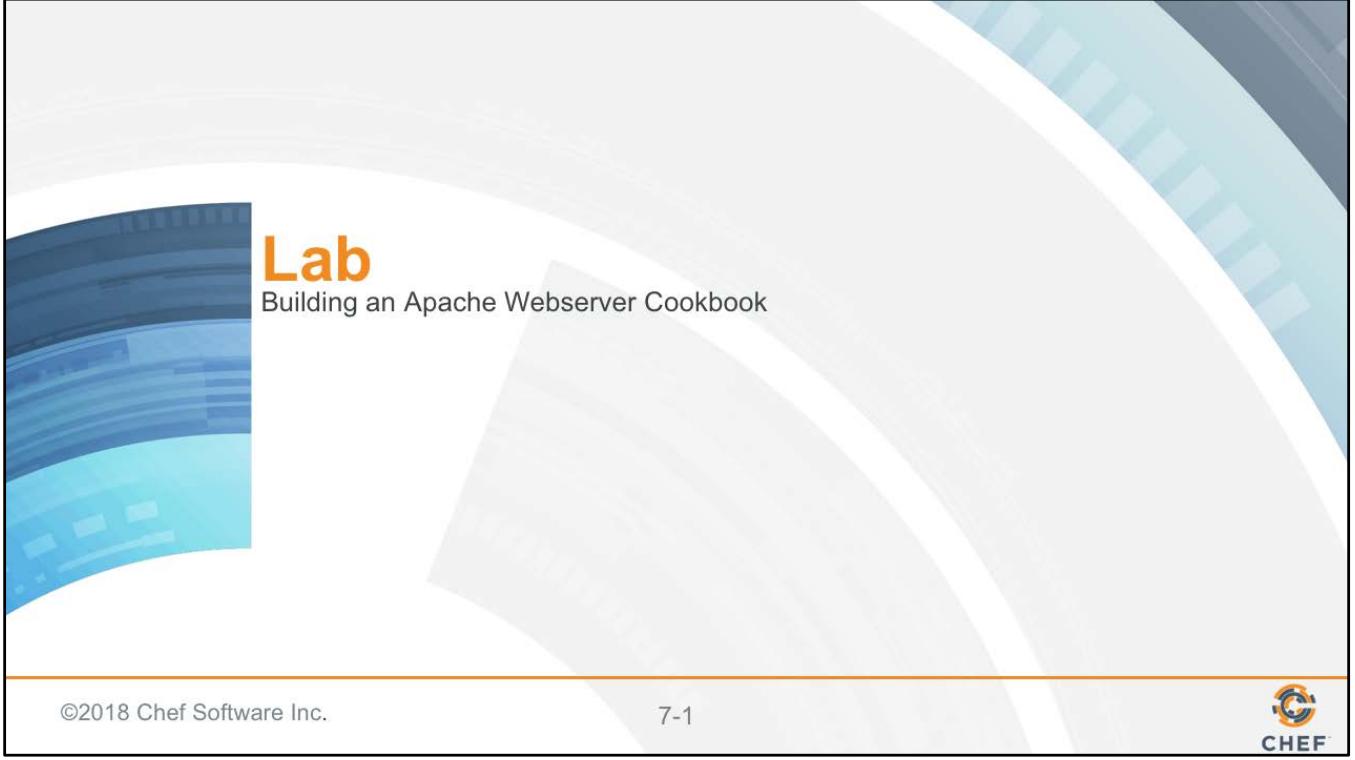
- Resources (file, cookbook_file, template, and remote_file)
- Templates
- ERB

What questions can we help you answer?

Generally or specifically about resources, templates, and ERB.



©2018 Chef Software Inc.



Lab

Building an Apache Webserver Cookbook

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



Group Lab: Pre-built Linux Node

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

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 ~]$
```

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!
```



Group 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.

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: Working within Home Directory

 \$ cd ~

DEV

©2018 Chef Software Inc.

7-7



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: Creating the apache Cookbook



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

Generating cookbook apache

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

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



Now create a cookbook called 'apache'



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
```



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



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.



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 Client, version 13.2.20  
resolving cookbooks for run list: ["apache"]  
Synchronizing Cookbooks:  
  - apache (0.1.0)  
Installing Cookbook Gems:  
Compiling Cookbooks...  
Converging 3 resources  
Recipe: apache::server  
  * yum_package[httpd] action install  
    - install version 2.2.15-60.el6.centos.6 of package httpd
```



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-172-31-29-209</h2>
    <h2>MEMORY: 604192kB</h2>
    <h2>CPU Mhz: 1800.000</h2>
  </body>
</html>
```



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?



CHEFTM

©2018 Chef Software Inc.

Workstation Installation

Configuring Your Laptop as a Workstation

©2018 Chef Software Inc.

8-1



Objectives

After completing this module, you should be able to

- Ensure that the ChefDK is installed 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 the ChefDK

Installing the tools on your system

Objective:

- Install the ChefDK
- 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 the Chef Development Kit (ChefDK). 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 Development Kit (ChefDK)

The ChefDK contains tools like chef, and chef-client.

The omnibus installer is used to set up the Chef development kit on a workstation, including an embedded version of Ruby, RubyGems, OpenSSL, key-value stores, parsers, libraries, command line utilities, and community tools such as Kitchen, Berkshelf, and ChefSpec.

<https://downloads.chef.io/chef-dk/>

Throughout this course we have been using a number of tools found within the ChefDK. The ChefDK 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.

You can download the ChefDK at <https://downloads.chef.io/chef-dk>.

Instructor Note: Prior to attending this course they may have received correspondence that informed them to setup the ChefDK on their systems. It is possible that they did not and this slides acts as a good reminder to ensure that they have the necessary tools before continuing on to the next section. **IMPORTANT:** This course was tested against ChefDK version 0.17.17. If you use a different version, the exercises and labs may not work properly.



Download the ChefDK

ChefDK is a tool chain built on top of the Ruby programming language.

The ChefDK installer does not install any particular graphical-user-interface—installs CLI instead

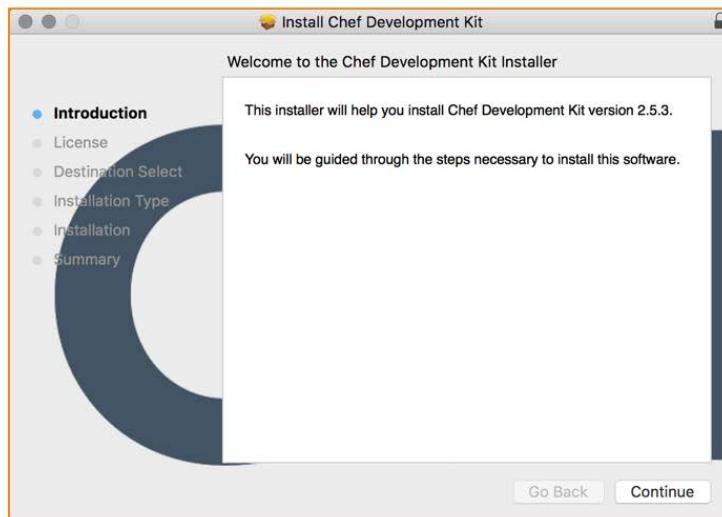
<https://downloads.chef.io/chef-dk/>

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

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

You may have already downloaded the ChefDK previously in this course.

Installing ChefDK



©2018 Chef Software Inc.

8-6



Follow the ChefDK installation wizard's instructions. It could take over 10 minutes to install ChefDK.

ChefDk will be installed into an opscode folder on your laptop.



Installing the ChefDK

Installing the tools on your system

Objective:

- Install the ChefDK
- Execute commands to ensure everything is installed
- Install a text editor (optional)

Verify installation of ChefDK



```
> chef --version
```

```
Chef Development Kit Version: 2.0.28
chef-client version: 13.2.20
delivery version: master (17c1b0fed9be4c70f69091a6d21a4cbf0df60a23)
berks version: 6.2.0
kitchen version: 1.16.0
inspec version: 1.31.1
```

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

We see some familiar tools like 'chef-client', and 'ohai', these are the ones that we have used on our remote workstation. Some of these tools you have not seen yet. Later in this course, we'll explore 'berks'.



Installing the ChefDK

Installing the tools on your system

Objective:

- ✓ Install the ChefDK
- ✓ 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 the ChefDK

Installing the tools on your system

Objective:

- ✓ Install the ChefDK
- ✓ Execute commands to ensure everything is installed
- ✓ Install a text editor (optional)

Q&A

What questions can we answer for you?





CHEFTM

©2018 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
- 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.

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 for 'CHEF 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 'Forgot your password?' link.

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'm not a robot



I agree to the [Terms of Service](#) and the [Master License and Services Agreement](#).

Get Started

Instructor Note: Learners that already have an account will login instead of creating an account. The learner's account may be tied to an production organization. The learner can create a new organization with their existing account. If there are still concerns they can create a new login with a unique email address.



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.

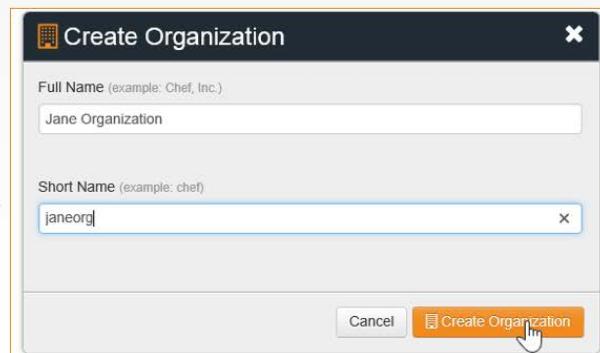


Instructor Note: Learners that already have an account will already have an organization. They are welcome to use that organization if it does not have any cookbooks or nodes from previous exercises or from their production systems. It is often easier to have them create a new organization for the purposes 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 your new organization to highlight it and then click Starter Kit.**
6. From the resulting window, click the Download Starter Kit button.

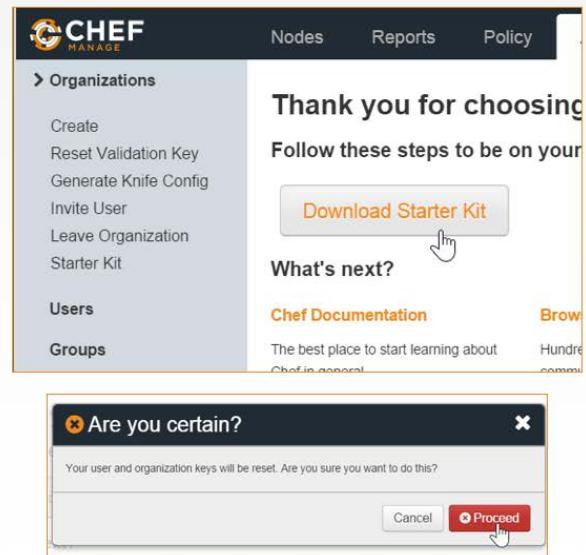


Instructor Note: By far the easiest way to get setup with Managed Chef Server is download the Starter Kit. The most important pieces of the starter kit are found in a hidden directory (".".chef") which contains the organization key, user key, and organization configuration file.

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
```

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?



CHEFTM

©2018 Chef Software Inc.

The Chef Server

A Hub for Configuration Data

Objectives



After completing this module, you should be able to

- Connect your local workstation (laptop) to a Chef Server
- Clone a GitHub repository
- Upload cookbooks to a Chef Server
- Bootstrap a node
- Manage a node via a Chef Server

In this module you will learn how to connect your local workstation to a Chef Server, upload cookbooks to a Chef Server, bootstrap a node, manage a node via a Chef Server.

Managing an Additional System



To manage another system, you would need to:

1. Provision a new node within your company or appropriate cloud provider with the appropriate access to login to administrate the system.
2. Install the Chef tools.
3. Transfer the apache or myiis cookbook.
4. Run chef-client on the new node to apply the apache or myiis cookbook's default recipe.

As an exercise, roughly estimate the time it would take to accomplish this series of steps of preparing another node.

A new system would require us to provision a new node within your company or appropriate cloud provider with the appropriate access to login to administrate the system.

Install the Chef tools.

Transfer the apache cookbook.

Run chef-client locally to apply the apache cookbook's default recipe.

Instructor Note: This exercise is to show the value of using a Chef Server with regard to managing multiple systems. It can be done with the group, with individuals, or done in pairs.

Managing Additional Systems



Installing the Chef tools, transferring the apache cookbook, and applying the run list is not terribly expensive.

- Chef provides a one-line curl install.
- You could use **git** to clone the repository from a common **git** repository.
- Applying the run list.

The cost of installing the Chef tools, transferring the apache cookbook, and applying the run list is not terribly expensive.

Chef provides a one-line curl install for the Chef Development Kit (ChefDK).

You could use git to clone the repository from a common git repository. Another option is to archive the cookbook and then using SCP to copy over the contents. A third might be to mount a file share. There are a myriad ways to transfer the cookbooks to the new instance.

Then applying the run list requires the execution of a command on that system.

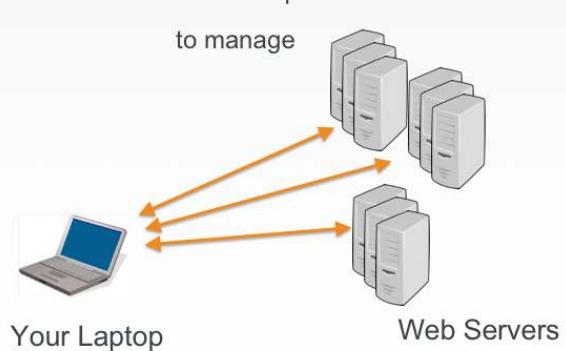


Managing Additional Systems

Now



Future



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

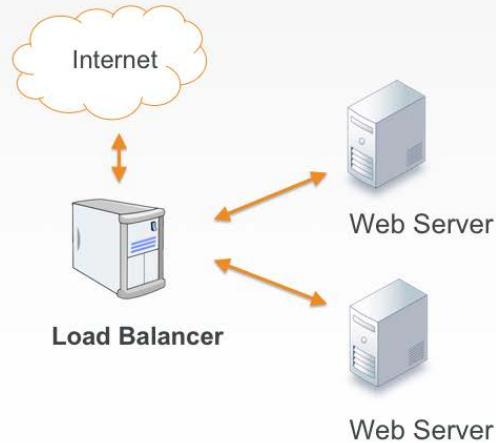
Some may think 10 minutes is not so bad. But what if there were 10 new nodes? 20 new nodes?

As the popularity of your site grows, one server will not be able to keep with all of the web requests. You will need to provision additional machines as demand increases.

Managing User Traffic



A load balancer can forward incoming user web requests to other nodes.



Let's change topics for a moment to managing user web traffic.

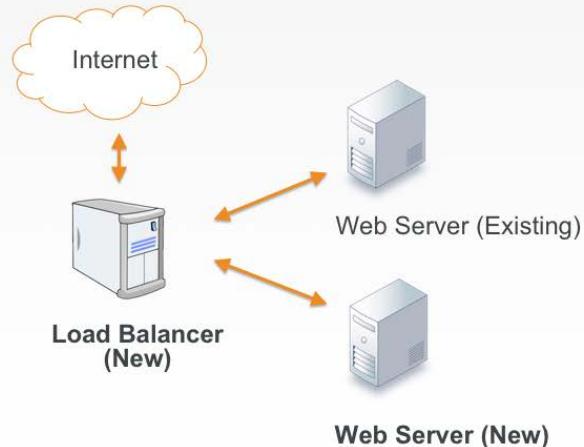
In addition to the complexities of configuring and managing multi-server infrastructure, such as web servers, you also need to develop a way to route incoming traffic to each of those web servers and other nodes. There are many ways that you can route the traffic from one node to a group of similar nodes. This can be done with services by some of the major cloud providers or it can be done with another instance running as a load balancer. A load balancer allows us to receive incoming requests and forward those requests to other nodes. A load balancer allows us to receive incoming requests and forward those requests to other nodes.

Instructor Note: The preceding three slides covered the complexity of configuring and managing multi-server infrastructure. This slide and page is now talking about managing user web traffic, via a load balancer.

Managing User Traffic



Today you will set up a new load balancer that will direct web requests to similarly-configured nodes.



Steps to Setup Load Balancer and Web Servers

Web Server

1. Provision the instance
2. Install Chef
3. Copy the web server cookbook
4. Apply the cookbook

Load Balancer

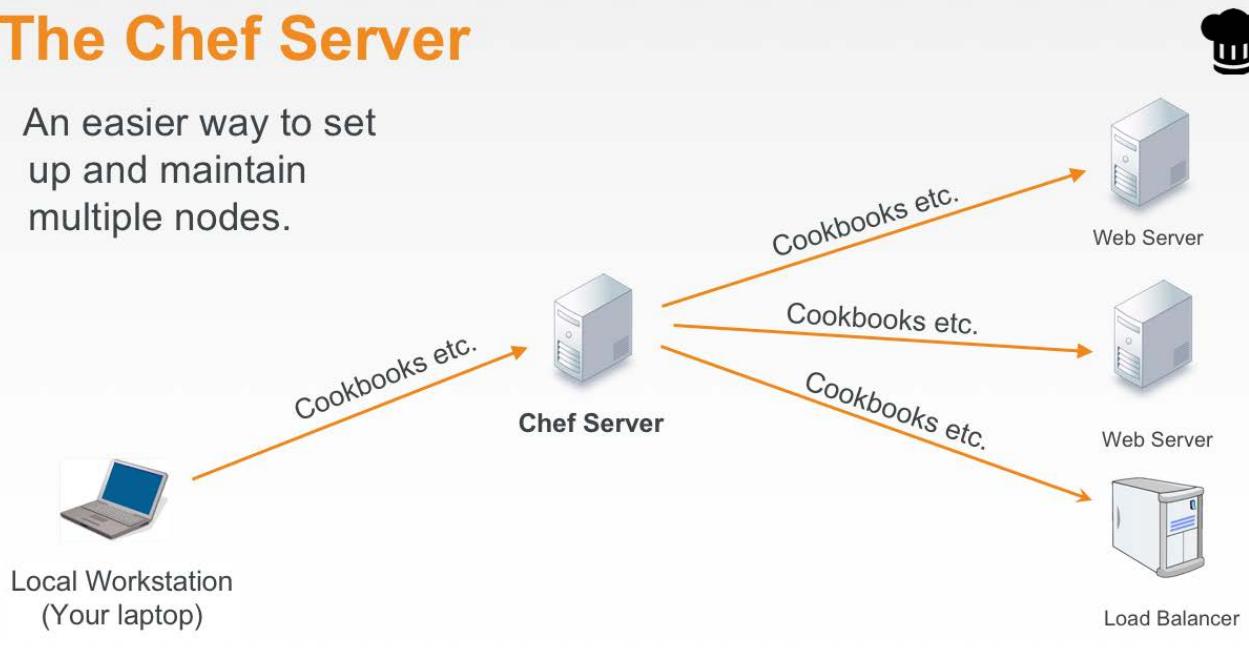
1. Create the haproxy (load balancer) cookbook
2. Provision the instance
3. Install Chef
4. Copy the haproxy cookbook
5. Apply the cookbook

Whether you tackle installing, configuring, or running a load balancer or recreate a second instance running the apache cookbook's default recipe, you will need to solve the problem of how you can manage multiple systems. Each system would need to have Chef installed, the cookbooks copied onto each system, and a run list of the recipes to apply to each system.

Instructor Note: The left side is setting up a new web server that is like the one they created yesterday. The right is new work that the learner will be accomplishing today. Note that the haproxy is actually a load balancer.

The Chef Server

An easier way to set up and maintain multiple nodes.



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

The Chef Server is designed to help us manage multiple nodes in this situation. The Chef Server acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by 'chef-client'. Nodes, such as web servers, load balancers, etc., use 'chef-client' to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server). In a production environment, the 'chef-client' runs in an automated mode—it polls the Chef Server for updates at set intervals and then applies any configuration changes. This scalable approach distributes the configuration effort throughout the organization.



GL: Managing Nodes with Hosted Chef

It will be easier to distribute the cookbooks we write to multiple nodes if we store them in a central repository.

Objective:

- Download and copy the required cookbooks to your local machine
- Upload the 'myiis' cookbook to the Chef server
- Verify the cookbook is found on the Chef server

We are going to need a copy of the myiis and apache cookbooks that we created on day one and get this up to the Chef 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/devops-foundations-repo>

©2018 Chef Software Inc.

10-11



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

chef-training / devops-foundations-repo

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

No description, website, or topics provided.

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

| SamMc87 Initial commit | Latest commit 61f237f a day ago |
|------------------------|---------------------------------|
| apache | Initial commit |
| myiis | Initial commit |
| .DS_Store | Initial commit |
| README.md | Initial commit |

©2018 Chef Software Inc.

10-12

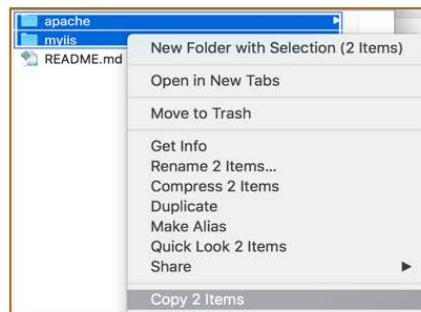


GL: Move Cookbooks to the Cookbooks Folder

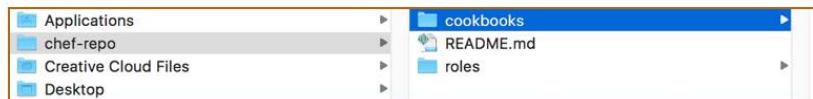
Steps:

1. Open the downloaded [chef-essentials-master.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.

chef-essentials-master



chef-repo/cookbooks





GL: Managing Nodes with Hosted Chef

It will be easier to distribute the cookbooks we write to multiple nodes if we store them in a central repository.

Objective:

- ✓ Download and copy the required cookbooks to your local machine
- ❑ Upload the 'myiis' cookbook to the Chef server
- ❑ Verify the cookbook is found on the Chef server

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)
```

```
-rw-r--r-- 1 technotrainer staff 47 Oct 25 22:03 Berksfile  
-rw-r--r-- 1 technotrainer staff 53 Oct 25 22:03 README.md  
-rw-r--r-- 1 technotrainer staff 1133 Oct 25 22:03 chefignore  
-rw-r--r-- 1 technotrainer staff 740 Oct 25 22:03 metadata.rb  
drwxr-xr-x 4 technotrainer staff 136 Oct 25 22:03 recipes  
drwxr-xr-x 4 technotrainer staff 136 Oct 25 22:03 spec  
drwxr-xr-x 3 technotrainer staff 102 Oct 25 22:03 test
```



Here you should find that the myiis 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 Server.

knife is a command-line tool that allows us to request and send information to the Chef Server.

knife helps users manage nodes, cookbooks, roles, environments, and more. knife does this through a series of sub-commands.

GL: knife --help



```
$ knife --help
```

```
Available subcommands: (for details, knife SUB-COMMAND --help)
```

```
** BOOTSTRAP COMMANDS **
```

```
knife bootstrap FQDN (options)
knife bootstrap windows ssh FQDN (options)
knife bootstrap windows winrm FQDN (options)
```

```
** CLIENT COMMANDS **
```

```
knife client bulk delete REGEX (options)
knife client create CLIENT (options)
knife client delete CLIENT (options)
knife client edit CLIENT (options)
```



You can look at all the commands with 'knife --help'.

This will display all the sub-commands available. In your case you want to verify that the client list contains a single entry so you need to look for help for the specific command 'knife client --help'.

GL: knife cookbook --help



```
$ knife cookbook --help
```

```
** COOKBOOK COMMANDS **  
knife cookbook bulk delete REGEX (options)  
knife cookbook create COOKBOOK (options)  
knife cookbook delete COOKBOOK VERSION (options)  
knife cookbook download COOKBOOK [VERSION] (options)  
knife cookbook list (options)  
knife cookbook metadata COOKBOOK (options)  
knife cookbook metadata from FILE (options)  
knife cookbook show COOKBOOK [VERSION] [PART] [FILENAME] (options)  
knife cookbook test [COOKBOOKS...] (options)  
knife cookbook upload [COOKBOOKS...] (options)
```



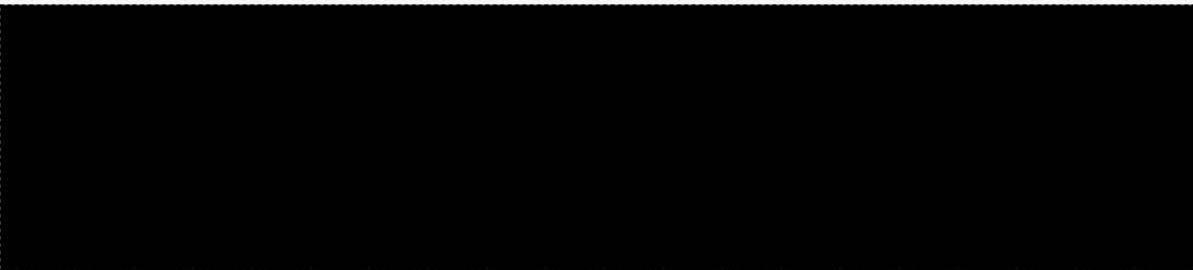
Similar to asking the Chef Server about the list of available clients, you can also ask for information about cookbooks. You can find all the commands related to the cookbooks subcommand by running `knife cookbook --help`.

Similar to the list of clients, you can examine a list of cookbooks.

GL: knife cookbook list



```
$ knife cookbook list
```



Running this command will return the cookbooks currently uploaded to the Chef Server. The empty response should come as no surprise.

You want to change that. So you are going to upload each of your cookbooks to the Chef Server.

GL: Change to the cookbooks/myiis Directory

```
└─$ cd cookbooks/myiis
```



To upload a cookbook to the Chef Server you need to be within the directory of the cookbook. Let us start with the myiis cookbook. Change directory into the myiis cookbook directory which is within the cookbooks directory.



Berkshelf

Berkshelf is a cookbook management tool that allows us to upload your cookbooks and all of its dependencies to the Chef Server.

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

To upload the cookbook you will need to use another tool called Berkshelf.

Berkshelf is a cookbook management tool that allows us to upload your cookbooks and all of its dependencies to the Chef Server. In this instance, your current cookbooks have no dependencies, but in the future when they do, Berkshelf will assist you in ensuring those are all uploaded.

<http://berkshelf.com>

GL: Run berks --help



```
$ berks --help
```

```
Commands:
berks apply ENVIRONMENT      # Apply version locks from Berksfile.lock to a ...
berks contingent COOKBOOK    # List all cookbooks that depend on the given c...
berks cookbook NAME [PATH]   # Create a skeleton for a new cookbook
berks help [COMMAND]         # Describe available commands or one specific c...
berks info [COOKBOOK]        # Display name, author, copyright, and dependen...
berks init [PATH]            # Initialize Berkshelf in the given directory
berks install                # Install the cookbooks specified in the Berksfile
berks list                   # List cookbooks and their dependencies specifi...
berks outdated [COOKBOOKS]   # List dependencies that have new versions avai...
berks package [PATH]          # Vendor and archive the dependencies of a Berk...
berks search NAME            # Search the remote source for cookbooks matchi...
berks shelf SUBCOMMAND       # Interact with the cookbook store
berks show [COOKBOOK]         # Display the path to a cookbook on disk
```



Berkshelf is a command-line tool that you can ask to see available the commands.

To see full command descriptions you can include the berks sub commands. For example: `berks apply ENVIRONMENT --help`

GL: Run berks install



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using myiis (0.2.1) from source at .
```



Berkshelf is used on a per-cookbook basis. As dependencies are often per cookbook you'll need to change into the directory of the cookbook.

You should install any dependencies that your cookbook might have. Again, in this instance there are no dependencies external to this cookbook but Berkshelf ensures that this is the case when it runs the 'berks install' command.

You'll see that it finds the current cookbook within your current directory, it contacts the Supermarket for any external dependencies, and then ...

GL: See the Berksfile.lock



```
$ ls -al (or dir -Force if using Powershell)
```

```
drwxr-xr-x 7 chef chef 4096 Aug 27 18:44 .
drwxr-xr-x 4 chef chef 4096 Aug 27 16:17 ..
drwxr-xr-x 8 chef chef 4096 Aug 27 16:07 .git
-rw-r--r-- 1 chef chef 126 Aug 27 15:46 .gitignore
drwxr-xr-x 3 chef chef 4096 Aug 27 18:45 .kitchen
-rw-r--r-- 1 chef chef 183 Aug 27 18:44 .kitchen.yml
-rw-r--r-- 1 chef chef 47 Aug 27 15:46 Berksfile
-rw----- 1 chef chef 77 Aug 27 18:45 Berksfile.lock
-rw-r--r-- 1 chef chef 54 Aug 27 15:46 README.md
-rw-r--r-- 1 chef chef 974 Aug 27 15:46 cheftignore
-rw-r--r-- 1 chef chef 198 Aug 27 15:46 metadata.rb
drwxr-xr-x 2 chef chef 4096 Aug 27 16:34 recipes
```



...it completes by writing a Berksfile.lock to the file system.

The Berksfile.lock is a receipt of all the cookbooks and dependencies found at the exact moment that you ran 'berks install'.

GL: See the Contents of the Berksfile.lock



```
$ cat Berksfile.lock
```

```
DEPENDENCIES
myiis
  path: .
  metadata: true

GRAPH
myiis (0.2.1)
```



This lock file is useful to ensure that in the future you use the same dependencies when working with the cookbook.

GL: Upload the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded myiis (0.2.1) to: 'https://api.opscode.com:443/organizations/ORG'
```



With the dependencies accounted for, it is time to upload the to the Chef Server. This is another sub-command that Berkshelf provides called 'upload'. Run the command to upload the myiis cookbook to the Chef Server.



GL: Managing Nodes with Hosted Chef

It will be easier to distribute the cookbooks we write to multiple nodes if we store them in a central repository.

Objective:

- Download and copy the required cookbooks to your local machine
- Upload the 'myiis' cookbook to the Chef server
- Verify the cookbook is found on the Chef server

GL: Display Cookbooks within Your Org



```
$ knife cookbook list
```

```
myiis      0.2.1
```



When that is complete you can return to the cookbook command that allows you to display the cookbooks within your organization by running this command. This will show you that the Chef Server has the apache cookbook that you have uploaded.

Note: You can also use the -a flag to see all versions of a cookbook. For example, if you had multiple versions of a cookbook:

```
$ knife cookbook list -a
```



GL: Managing Nodes with Hosted Chef

It will be easier to distribute the cookbooks we write to multiple nodes if we store them in a central repository.

Objective:

- ✓ Download and copy the required cookbooks to your local machine
- ✓ Upload the 'myiis' cookbook to the Chef server
- ✓ Verify the cookbook is found on the Chef server



Lab: Upload the Apache Cookbook

Let's get the cookbook on our local machine to the Chef Server

Objective:

- Navigate to the 'apache' directory
- Upload the 'apache' cookbook
- Verify the cookbook is uploaded

As a lab, upload the remaining cookbooks within the cookbooks directory. After you have done that verify that the cookbooks have been uploaded.

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: Navigate to the apache Directory



```
$ cd ~/chef-repo/cookbooks/apache  
$ ls -l
```

```
-rw-r--r-- 1 technotrainer staff 47 Oct 25 22:03 Berksfile  
-rw-r--r-- 1 technotrainer staff 53 Oct 25 22:03 README.md  
-rw-r--r-- 1 technotrainer staff 1133 Oct 25 22:03 chefignore  
-rw-r--r-- 1 technotrainer staff 740 Oct 25 22:03 metadata.rb  
drwxr-xr-x 4 technotrainer staff 136 Oct 25 22:03 recipes  
drwxr-xr-x 4 technotrainer staff 136 Oct 25 22:03 spec  
drwxr-xr-x 3 technotrainer staff 102 Oct 25 22:03 test
```



This again will be a copy of the original apache cookbook and should contain all the work we saw earlier.



Lab: Upload the Apache Cookbook

Let's get the cookbook on our local machine to the Chef Server

Objective:

- Navigate to the 'apache' directory
- Upload the 'apache' cookbook
- Verify the cookbook is uploaded

As a lab, upload the remaining cookbooks within the cookbooks directory. After you have done that verify that the cookbooks have been uploaded.

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: Install the Cookbook Dependencies



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using apache(0.1.0) from source at .
```



Run "berks install" to install all the cookbook dependencies.

Lab: Upload the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded apache (0.1.0) to:  
'https://api.opscode.com:443/organizations/ORG'
```



Run "berks upload" to upload the cookbook and all its dependencies to the Chef Server.



Lab: Upload the Apache Cookbook

Let's get the cookbook on our local machine to the Chef Server

Objective:

- Navigate to the 'apache' directory
- Upload the 'apache' cookbook
- Verify the cookbook is uploaded

As a lab, upload the remaining cookbooks within the cookbooks directory. After you have done that verify that the cookbooks have been uploaded.

Instructor Note: Allow 5 minutes to complete this exercise.

Lab: Is the apache Cookbook Uploaded?



```
$ knife cookbook list
```

```
apache      0.1.0
myiis       0.2.1
```



Lastly, run "knife cookbook list" to validate that the apache cookbook is now uploaded to the Chef Server.



Lab: Upload the Apache Cookbook

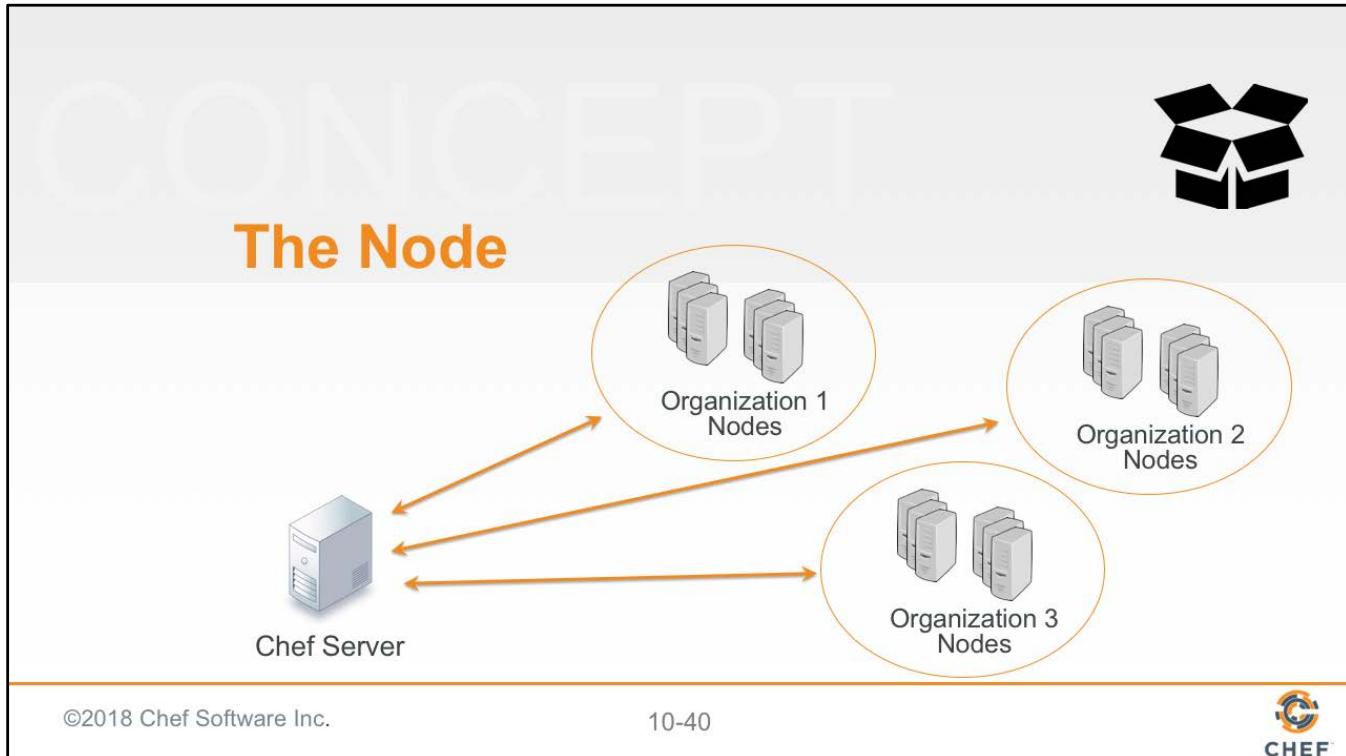
Let's get the cookbook on our local machine to the Chef Server

Objective:

- ✓ Navigate to the 'apache' directory
- ✓ Upload the 'apache' cookbook
- ✓ Verify the cookbook is uploaded

As a lab, upload the remaining cookbooks within the cookbooks directory. After you have done that verify that the cookbooks have been uploaded.

Instructor Note: Allow 5 minutes to complete this exercise.



As you know by now, a node is a server that Chef is managing. A node could be a web server, an application server, a database server, a load balancer, and so on.

A node can only join one organization. To be a node means that it has Chef installed, has configuration files in place, and when you run the chef-client application with no parameters it will successfully contact the Chef Server and ask it for the run list that it should apply and the cookbooks required to execute that run list.

When a node is part of the organization you manage that information on the Chef Server as well. A Chef Server can manage multiple organizations. Managing that information in a Chef Server allows us to use for inventory, querying and searching.



GL: Bootstrap Your Node

In this lab you will use a new instance and bootstrap it as a managed node.

You'll need the FQDN or Public IP of that instance 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 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 Server is or have the credentials to even talk to it securely. We could manually configure a node but there is an easier way of doing that through a process called 'bootstrapping'.

Bootstrapping will install Chef if necessary and then configure the node to talk securely to a specified Chef Server.

<https://learn.chef.io/modules/beyond-the-basics#/>

GL: Change to the chef-repo



```
$ cd ~/chef-repo
```



Return to the root of the chef repository.

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)
```



Verify that you have no existing nodes within your organization. You can use the 'knife node –help' command to see that you can ask for the list of all nodes within your organization with the list command.

GL: Run 'knife node list'



```
$ knife node list
```



Run "knife node list" to see that you have no nodes currently registered with your Chef Server. At this point the results should be blank.

GL: Run 'knife bootstrap –help'



```
$ knife bootstrap --help
```

```
knife bootstrap FQDN (options)
  --bootstrap-curl-options OPTIONS
    Add options to curl when install chef-client
  --bootstrap-install-command COMMANDS
    Custom command to install chef-client
  --bootstrap-no-proxy [NO_PROXY_URL|NO_PROXY_IP]
    Do not proxy locations for the node being
bootstrapped; this option is used internally by Opscode
  --bootstrap-proxy PROXY_URL  The proxy server for the node being
bootstrapped
  -t TEMPLATE,
    Bootstrap Chef using a built-in or custom
template. Set to the full path of an erb
template or use one of the built-in templates.
```



Knife provides a bootstrap subcommand that takes a number of options.

When you bootstrap an instance it is performing the following: Installing chef tools if they are not already installed; Configuring Chef to communicate with the Chef Server; Running chef-client to apply a default run list.

GL: Bootstrap Your Node

```
> knife bootstrap windows winrm IP -x USER -P PWD -N iis_web
Creating new client for node1
Creating new node for node1
Connecting to ec2-54-175-46-24.compute-1.amazonaws.com
ec2-54-175-46-24.compute-1.amazonaws.com resolving cookbooks for run list: []
ec2-54-175-46-24.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-175-46-24.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-175-46-24.compute-1.amazonaws.com [2016-09-16T16:51:21+00:00] WARN: Node
node1 has an empty run list.
ec2-54-175-46-24.compute-1.amazonaws.com Converging 0 resources
ec2-54-175-46-24.compute-1.amazonaws.com
ec2-54-175-46-24.compute-1.amazonaws.com Running handlers:

```

©2018 Chef Software Inc. 10-47

CHEF

To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-x' flag and the password '-P' flag. Include the '--sudo' flag because you are installing software and writing configuration to directories traditionally owned by the root user. Name the node with the '-N' flag. This is optional but makes it easier for us to communicate. When we ask you to look at the details of 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' Again



```
$ knife node list
```

```
iis_web
```



When bootstrapping is done, you can see that your organization knows about the new node by again running the command "knife node list". You now see that you have a new node, iis_web, uploaded to the Chef Server.

GL: View More Information About Your Node



```
$ knife node show iis_web
```

```
Node Name: iis_web
Environment: _default
FQDN: ip-172-31-8-68.ec2.internal
IP: 54.175.46.24
Run List:
Roles:
Recipes:
Platform: windows 6.3.9200
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 Server stores.

GL: Add a Recipe to a Run List



```
$ knife node run_list add iis_web "recipe[myiis]"
```

```
iis_web:  
  run_list: recipe[myiis]
```



iis_web does not have a list of recipes that it applies to the system by default. You can make Chef Server tell iis_web to apply a specific run-list the next time iis_web runs 'chef-client'.

You can do that through the 'knife node run_list add' command. In this example, you are adding to iis_web's run-list the myiis cookbook's default recipe.



WinRM Woes

Logging into systems is a pain. We can use another knife tool to allow us to send commands to all of our nodes.

We asked you to login to that remote node and run 'chef-client' to apply the run list defined for that node. This does in fact work but considering that we may need to execute this command for this node and many future nodes, it seems like a lot of windows and commands that we would need to execute.

GL: Running a Command with Knife



```
$ knife winrm IP -m -x USERNAME -P PASSWORD "chef-client"
```

```
54.159.197.193 Starting Chef Client, version 13.6.0
54.159.197.193
54.159.197.193 [2017-10-30T23:33:23+00:00] INFO: *** Chef 13.6.0 ***
54.159.197.193 [2017-10-30T23:33:23+00:00] INFO: Platform: x64-mingw32
54.159.197.193 [2017-10-30T23:33:23+00:00] INFO: Chef-client pid: 3016
54.159.197.193 [2017-10-30T23:33:23+00:00] INFO: The plugin path
C:\chef\ohai\plugins does not exist. Skipping...
54.159.197.193 [2017-10-30T23:33:27+00:00] INFO: Run List is [recipe[myiis]]
54.159.197.193 [2017-10-30T23:33:27+00:00] INFO: Run List expands to [myiis]
54.159.197.193 [2017-10-30T23:33:27+00:00] INFO: Starting Chef Run for iis_web
54.159.197.193 [2017-10-30T23:33:27+00:00] INFO: Running start handlers
54.159.197.193 [2017-10-30T23:33:27+00:00] INFO: Start handlers complete.
```



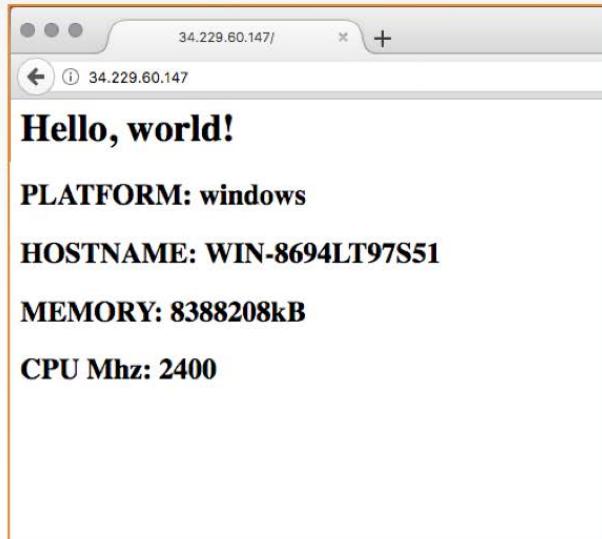
©2018 Chef Software Inc.

10-52



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

GL: Verify that the New Node Serves the Page



Verify that the node serves up the default html page that contains the node's internal IP address and hostname.



Discussion

What is the benefit of storing cookbooks in a central repository?

What is the primary tool for communicating with the Chef Server?

How did you add a node to your organization?

Answer these questions.

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



Q&A

What questions can you help you answer?

- Chef Server
- Managed Chef
- Berkshelf
- Bootstrapping Nodes

With all of the objectives complete you are finished with this section. What questions can I answer for you?



CHEFTM

©2018 Chef Software Inc.

Cookbook Attributes, Attribute Files and Dependencies

Setting Attributes within a Cookbook

©2018 Chef Software Inc.

11-1



Objectives

After completing this module, you should be able to

- Explain where cookbook attributes reside
- Create a wrapper cookbook
- Configure dependencies between cookbooks

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.

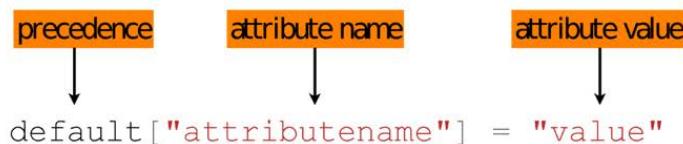
Ideally you will never have to manually edit the contents of a cookbook to be able to use it in a specific use case. The way that we can insure this is by creating cookbooks that are data driven and change their behavior based on attributes that can be overridden through the use of roles and environments. 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
 - Upload cookbook to the Chef server
 - Update the run list of the iis_web node to use the default recipe of the company_web cookbook and converge the node

We might want to create a wrapper cookbook that has access to everything found in the original cookbooks.

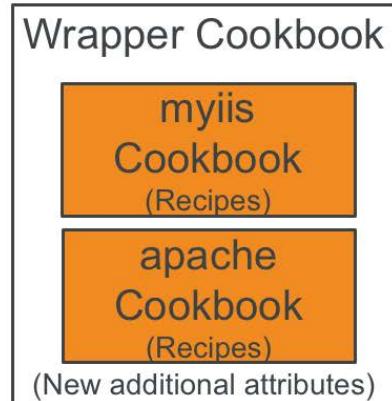
Instructor Note:

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

Wrapper Cookbooks

A wrapper cookbook is a new cookbook that encapsulates the functionality of the original cookbook(s).

It can access all of the recipes, cookbook components, and attributes found in the original cookbook(s) and implement them in new ways.



<https://docs.chef.io/supermarket.html#wrapper-cookbooks>

<https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/>

©2018 Chef Software Inc.

11-10



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

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



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 '>= 12.1' if respond_to?(:chef_version)
depends 'myiis'
depends 'apache'
```



We set our dependencies within the metadata.rb file of the wrapper cookbook. By doing this we know have access to all of the code base found in both 'myiis' as well as 'apache'.



Configuring Berks

`berks install` retrieves dependencies from the Chef Supermarket by default.

We want to override this to use the local 'myiis' and 'apache' cookbooks instead of the community versions.

Running 'berks install' resolves any dependencies that a cookbook may have. The default behavior for berkshelf is to reach out to the Chef Supermarket which is a public repository for community cookbooks. We don't want that to happen in this case and instead we will have berkshelf look locally for the dependent cookbooks.

GL: Edit Berksfile

```
~/chef-repo/cookbooks/company_web/Berksfile
```

```
source 'https://supermarket.chef.io'

metadata

cookbook 'myiis', path: '../myiis'
cookbook 'apache', path: '../apache'
```



By adding this code to the Berksfile configuration file we now will resolve these dependencies by looking locally for the cookbooks.

GL: Include Recipe Based on Platform

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

```
# Cookbook:: company_web
# Recipe:: default
#
# Copyright:: 2017, 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 cookbook to the Chef server
 - 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'
```

It's good practice to include the name of the cookbook in the attribute name – this helps trace where the value is set, although it is not enforced.



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
 - ❑ Upload cookbook to the Chef server
 - ❑ Update the run list of the iis_web node to use the default recipe of the company_web cookbook and converge the node

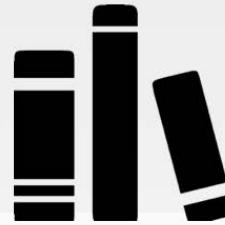


Using the company_name Attribute

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

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.

GL: Generate the Template file



```
$ chef generate template cookbooks/company_web homepage
```

```
Recipe: code_generator::template
  * directory[/Users/technotrainer/chef-repo/cookbooks/company_web/templates]
action create
  - create new directory /Users/technotrainer/chef-
repo/cookbooks/company_web/templates
  * template[/Users/technotrainer/chef-
repo/cookbooks/company_web/templates/homepage.erb] action create
    - create new file /Users/technotrainer/chef-
repo/cookbooks/company_web/templates/homepage.erb
      - update content in file /Users/technotrainer/chef-
repo/cookbooks/company_web/templates/homepage.erb from none to e3b0c4
```



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

```
/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>
```



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.



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 cookbook to the Chef server
 - Update the run list of the iis_web node to use the default recipe of the company_web cookbook and converge the node

GL: Navigate to company_web Cookbook



```
$ cd chef-repo/cookbooks/company_web
```



We must be within the 'company_web' directory for us to upload the cookbook.

GL: Upload company_web to Chef Server



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'apache' from source at ../apache
Fetching 'company_web' from source at .
Fetching 'myiis' from source at ../myiis
Fetching cookbook index from https://supermarket.chef.io...
Using apache (0.1.0) from source at ../apache
Using company_web (0.1.0) from source at .
Using myiis (0.2.1) from source at ../myiis
```



Running berks install will resolve our dependencies, which will be found locally.

GL: Upload company_web to Chef Server



```
$ berks upload
```

```
Skipping apache (0.1.0) (frozen)
/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-
5.1.1/lib/ridley/client.rb:271:in `server_url': Ridley::Client#url_prefix at
/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-
5.1.1/lib/ridley/client.rb:79 forwarding to private method
Celluloid::PoolManager#url_prefix
Uploaded company_web (0.1.0) to:
'https://api.chef.io:443/organizations/2017_oct_26'
Skipping myiis (0.2.1) (frozen)
```



berks upload will upload our new cookbook to the Chef Server.

GL: List cookbooks found on Chef Server



```
$ knife cookbook list
```

| | |
|-------------|-------|
| apache | 0.1.0 |
| company_web | 0.1.0 |
| myiis | 0.2.1 |



knife cookbook list will verify that our cookbook is now found on the Chef Server.



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 cookbook to the Chef server
 - Update the run list of the iis_web node to use the default recipe of the company_web cookbook and converge the node

GL: Set Run List for iis_web Node



```
$ knife node run_list set iis_web 'recipe[company_web]'
```

```
iis_web:  
  run_list: recipe[company_web]
```



Because we want iis_web to run the new 'company_web' cookbook, we must update its run list to reflect this. A 'knife node run_list set' will overwrite the previous run list with what we provide here.

GL: Verify Run List Update



```
$ knife node show iis_web
```

```
Node Name: iis_web
Environment: _default
FQDN: WIN-8694LT97S51.ec2.internal
IP: 34.229.225.40
Run List: recipe[company_web]
Roles:
Recipes: myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```



We verify that the run list has been updated with this command. `iis_web` no longer has the run list of '`myiis`' but rather that of '`company_web`'.

GL: Converge the iis_web Node



```
$ cd ~/chef-repo  
$ knife winrm IP -m -x USERNAME -P PASSWORD "chef-client"
```

```
34.229.225.40 [2017-10-27T02:29:09+00:00] INFO: *** Chef 12.18.31 ***  
34.229.225.40 [2017-10-27T02:29:09+00:00] INFO: Platform: i386-mingw32  
34.229.225.40 [2017-10-27T02:29:09+00:00] INFO: Chef-client pid: 908  
34.229.225.40 [2017-10-27T02:29:44+00:00] INFO: Run List is  
[recipe[company_web]]  
34.229.225.40 [2017-10-27T02:29:44+00:00] INFO: Run List expands to  
[company_web]  
34.229.225.40 [2017-10-27T02:29:44+00:00] INFO: Starting Chef Run for iis_web  
34.229.225.40 [2017-10-27T02:29:44+00:00] INFO: Running start handlers  
34.229.225.40 [2017-10-27T02:29:44+00:00] INFO: Start handlers complete.
```



At this point we want to apply this new cookbook on the iis_web node.

GL: Verify that the Node Serves the New Page



Verify that the node serves up the new Default page that now displays your company's name.



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 cookbook to the Chef server
 - ✓ Update the run list of the iis_web node to use the default recipe of the company_web cookbook and converge the node

Instructor Note:

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



Discussion

Attributes are like parameters to your cookbook, no 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?



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.



CHEFTM

©2018 Chef Software Inc.

Community Cookbooks

Find, Explore and View Chef Cookbooks

©2018 Chef Software Inc.

12-1



Objectives

After completing this module, you should be able to

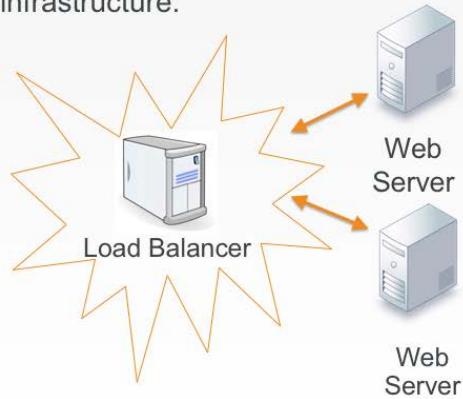
- Find cookbooks on the Chef Supermarket
- Create a wrapper cookbook for a community cookbook
- Replace the existing default values
- Upload a cookbook to the Chef Server
- Bootstrap a new node that runs the cookbook

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 a single web server running with our organization, it's now time to talk about the next goal to tackle. We need to setup a load balancer.

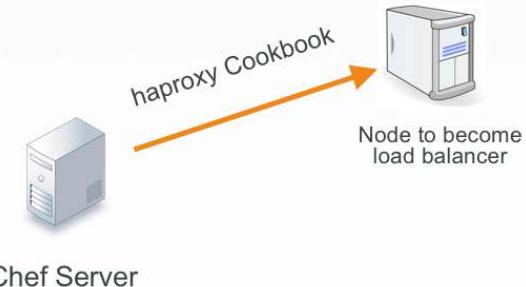
A load balancer is able to receive requests and relay them to other systems. In our case, we specifically want to use the load balancer to balance the entire traffic load between one or more systems.

This means we will need to establish a new node within our organization, install the necessary software to make the node a load balancer, and configure it so that it will relay requests to our existing node running apache 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.



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` node
- Upload cookbook to Chef Server
- Bootstrap a new node that runs the haproxy (load balancer) cookbook

Let's find the haproxy (load balancer) cookbook within the community site to learn more about it.

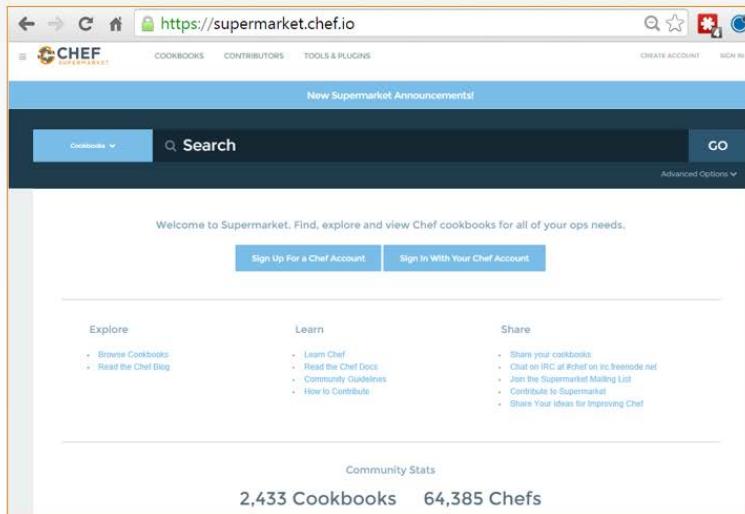
Instructor Note:

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: 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.



©2018 Chef Software Inc.

12-7



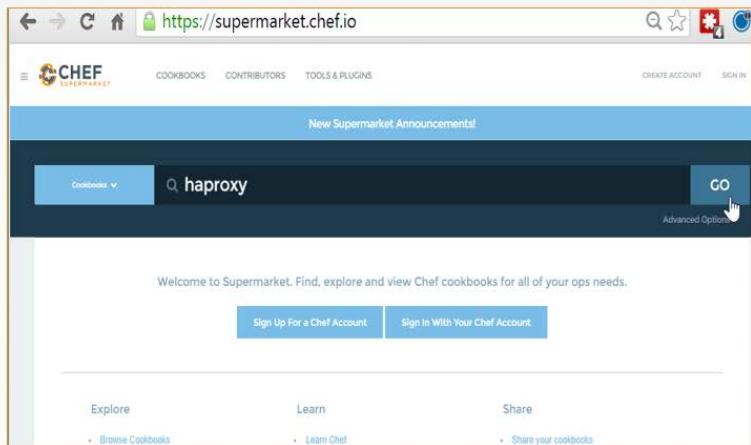
An important thing to remember is that on the community site are cookbooks managed by individuals. Chef does not verify or approve the cookbooks found in the Supermarket. These cookbooks solved problems for the original authors and then they decided to share them. This means that the cookbooks you find 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
2. Select the search field and type in [haproxy](#) in the search field. Then click the **GO** button.
3. Click the resulting [haproxy](#) link.



©2018 Chef Software Inc.

12-8



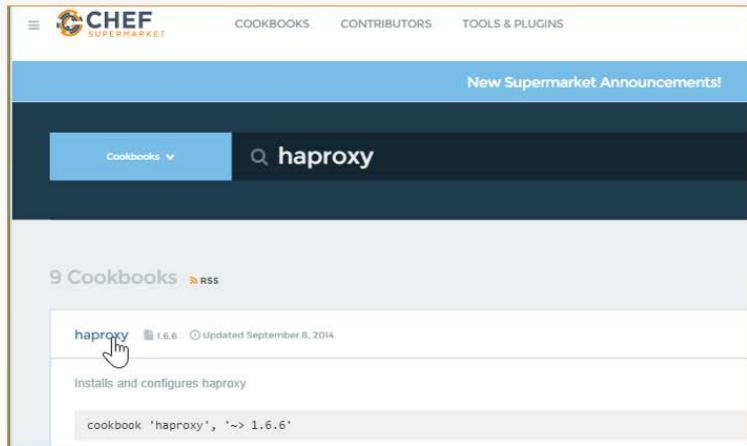
From the Supermarket main page type the search term "haproxy" and the click the GO button.

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
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 page for the 'haproxy' cookbook. At the top, there's an RSS feed icon and a 'Follow' button. Below that, the title 'haproxy' is displayed with '3.0.0' versions available. A brief description follows: 'Installs and configures haproxy'. Underneath, tabs for 'Berkshelf/Librarian', 'Policyfile', and 'Knife' are shown, along with a search bar containing 'cookbook \'haproxy\', \' = 3.0.0\''. Below these are tabs for 'README', 'Dependencies', 'Changelog', and 'Quality'. The main content area is titled 'haproxy Cookbook' and contains a note: 'Installs haproxy and prepares the configuration location.' At the bottom, there are status badges for 'build' (passing) and 'cookbook' (v5.0.0), followed by a link to 'View Source' and 'View Issues'. On the right side, a sidebar titled 'sous-chefs' shows profile icons for several maintainers. Below that, sections for 'UPDATED JANUARY 24, 2017', 'Created on October 25, 2009', 'PLATFORMS' (with icons for various operating systems), 'LICENSE' (Apache 2.0), and a 'View Details' button.

©2018 Chef Software Inc.

12-10



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 a webpage for the 'haproxy Cookbook'. At the top, there's a navigation bar with tabs: 'README' (which is selected), 'Dependencies', 'Changelog', and 'Quality'. Below the tabs, the title 'haproxy Cookbook' is displayed in a large, bold font. Underneath the title, there are two small colored boxes: a green one labeled 'build passing' and a blue one labeled 'cookbook v5.0.0'. A brief description follows: 'Installs haproxy and prepares the configuration location.' Below this, the 'Requirements' section lists 'Chef 12.1+'. The 'Platforms' section lists supported operating systems: 'Ubuntu 12.04+', 'RHEL 6+', 'CentOS6+', 'RHEL 7+', 'CentOS7+', and 'Debian 8+'.

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, there is an defined attribute that establishes the members that receive the proxy requests from the load balancer. This is available in a node attribute available through `node['haproxy']['members']` .

GL: Supermarket Cookbooks

These node attributes are different than the automatic ones defined by Ohai.

Attributes defined in a cookbook are not considered automatic.

Attributes

- `node['haproxy']['incoming_address']` - sets the address to bind the haproxy process on, 0.0.0.0 (all addresses) by default
- `node['haproxy']['incoming_port']` - sets the port on which haproxy listens
- `node['haproxy']['members']` - used by the default recipe to specify the member systems to add. Default

```
[{
  "hostname" => "localhost",
  "ipaddress" => "127.0.0.1",
  "port" => 4000,
  "ssl_port" => 4000
}, {
  "hostname" => "localhost",
  "ipaddress" => "127.0.0.1",
  "port" => 4001,
  "ssl_port" => 4001
}]
```

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

Prior to this point we have seen how node attributes are defined by Ohai but cookbooks also have this ability to define node attributes. These node attributes are different than the ones defined by Ohai as well. Ohai attributes are considered automatic attributes and generally inalienable characteristics about the node.

Attributes defined in a cookbook are not considered automatic. They are simply default values that we may change. There are many ways that we provide new default values for these. One way that we will learn is defining a wrapper cookbook.



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

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'.



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
- ❑ Upload cookbook to Chef Server
- ❑ Bootstrap a new node that runs the haproxy (load balancer) cookbook

GL: Returning 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 file content
- Committing cookbook files to git
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content
- Adding delivery configuration to feature branch
- Adding build cookbook to feature branch
- Merging delivery content feature branch to master
```

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

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



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 '>= 12.1' if respond_to?(:chef_version)

depends 'haproxy', '~> 3.0.0'
```



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.

GL: Supermarket Cookbooks

Currently, the haproxy cookbook assumes that there are two different services running on the localhost at port 4000 and port 4001.

In a moment, you'll need to change that.

Attributes

- `node['haproxy']['incoming_address']` - sets the address to bind the haproxy process on, 0.0.0.0 (all addresses) by default
- `node['haproxy']['incoming_port']` - sets the port on which haproxy listens
- `node['haproxy']['members']` - used by the default recipe to specify the member systems to add. Default

```
[  
  {"  
    "hostname" => "localhost",  
    "ipaddress" => "127.0.0.1",  
    "port" => 4000,  
    "ssl_port" => 4000  
  },  
  {"  
    "hostname" => "localhost",  
    "ipaddress" => "127.0.0.1",  
    "port" => 4001,  
    "ssl_port" => 4001  
  }]
```

<https://docs.chef.io/supermarket.html#wrapper-cookbooks>

Currently the haproxy cookbook assumes that there are two different services running on the localhost at port 4000 and port 4001. The haproxy process will relay messages to itself to those two ports.

That is not our configuration. First, we currently only have one system that we want to route traffic. Second, we want to have the traffic routed not to localhost but instead to our webserver, node1, which will have a completely different hostname and IP address.



include_recipe

A recipe can include one (or more) recipes located in cookbooks by using the `include_recipe` method.

When a recipe is included, the resources found in that recipe will be inserted (in the same exact order) at the point where the `include_recipe` keyword is located.

So we want to apply the default recipe that sets up the proxy server but we want to make some adjustments. We cannot change the original cookbook itself so we are instead going to load the contents of the original recipe in a recipe that we do control in our new cookbook. This is possible through the helper method `include_recipe`.

GL: Include the haproxy's manual recipe in default recipe

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

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

```
include_recipe 'haproxy::manual'
```



First, within the myhaproxy cookbook you will use the include_recipe method to specify the fully-qualified name of the cookbook and recipe that you want to execute. In this case, when you run your wrapped cookbooks recipe, you'll want it to run the original cookbook's manual recipe.

This is often called wrapping a recipe because our recipe calls the original recipe. The benefit is that we can define content before this recipe gets applied, to override functionality, or after this recipe gets applied, to replace functionality.

GL: Beginning to Replace Load Balancer Members

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node['haproxy']['members'] = [
  {
    'hostname' => 'localhost',
    'ipaddress' => '127.0.0.1',
    'port' => 4000,
    'ssl_port' => 4000
  },
  {
    'hostname' => 'localhost',
    'ipaddress' => '127.0.0.1',
    'port' => 4001,
    'ssl_port' => 4001
  }
]
include_recipe 'haproxy::manual'
```



Without changing anything any further, using this cookbook will simply execute the original cookbooks' recipe with all the same default values. Before you execute that recipe, you'll need to override the default values with your own.

Copy and paste the original default values into your recipe, as shown here. This is not the real information of our servers. We now want to find that information and replace this content with the true value of our node.

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.

GL: Viewing the Node's IP Address



```
$ knife node show iis_web -a ipaddress
```

```
iis_web:  
  ipaddress: 172.31.8.68
```



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
```



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: Inserting Real Node Data into the Attributes

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node['haproxy']['members'] = [
  {
    'hostname' => 'localhost',
    'ipaddress' => '127.0.0.1',
    'port' => 4000,
    'ssl_port' => 4000
  },
  {
    'hostname' => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',
    'ipaddress' => '52.8.71.11',
    'port' => 80,
    'ssl_port' => 80
  }
]
include_recipe 'haproxy::manual'
```



Remove one of the entries within the members array (shown in red).

Then update the information for the remaining member to include the public ipaddress and hostname for iis_web (shown in green).

GL: Setting the Default Attributes Precedence Level

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node.default['haproxy']['members'] = [{  
  'hostname' => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',  
  'ipaddress' => '52.8.71.11',  
  'port' => 80,  
  'ssl_port' => 80  
}]  
  
include_recipe 'haproxy::manual'
```



To replace a default attribute in a recipe you have to use:
'node.default['haproxy']['members']...'

So you need to change: 'node['haproxy']['members']' to
'node.default['haproxy']['members']'

GL: Viewing the Complete Recipe

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node.default['haproxy']['members'] = [{  
  'hostname' => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',  
  'ipaddress' => '52.8.71.11',  
  'port' => 80,  
  'ssl_port' => 80  
}]  
  
include_recipe 'haproxy::manual'
```

The final default recipe for the wrapper cookbook 'myhaproxy' looks like the above.

Save your recipe file.



Node Attribute Precedence

| | Attribute Files | Node / Recipe | Environment | Role |
|----------------|-----------------|---------------|-------------|------|
| default | 1 | 2 | 3 | 4 |
| force_default | 5 | 6 | | |
| normal | 7 | 8 | | |
| override | 9 | 10 | 12 | 11 |
| force_override | 13 | 14 | | |
| automatic | | 15 | | |

We must set the precedence of node attributes which will allow them to be overridden when chef-client is executed. There are four locations where we can set a node attribute, attribute files, recipes, environments, and 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: 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
- ❑ Upload cookbook to Chef Server
- ❑ Bootstrap a new node that runs the haproxy (load balancer) cookbook

GL: Moving to the Cookbook's Directory



```
$ cd ~/chef-repo/cookbooks/myhaproxy
```



You change into the directory for the 'myhaproxy' cookbook.

GL: Installing the Cookbook's Dependencies



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'myhaproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using cpu (2.0.0)
Using build-essential (8.0.3)
Using haproxy (3.0.4)
Using windows (3.2.0)
Using ohai (5.2.0)
Using poise (2.8.1)
Using mingw (2.0.1)
Using poise-service (1.5.2)
Using seven_zip (2.0.2)
Using myhaproxy (0.1.0) from source at .

```



We use the Berkshelf to upload our cookbooks. This is where Berkshelf really shines as a tool.

Run the command "berks install". When you run this command for a cookbook that has a dependency, you'll see that Berkshelf will download the haproxy cookbook and its dependencies as well. The haproxy cookbook is dependent on the build-essential cookbook and the cpu cookbook. If any of those cookbooks had dependencies, berkshelf would find those and download them as well.

Instructor Note: Berkshelf downloads these cookbooks into a common directory within your home path. They are not added alongside your other cookbooks.

GL: Uploading the Cookbook and Dependencies



```
$ berks upload
```

```
Uploaded build-essential (8.0.3) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded cpu (2.0.0) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded haproxy (3.0.4) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded mingw (2.0.1) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded myhaproxy (0.2.0) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded ohai (5.2.0) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded poise (2.8.1) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded poise-service (1.5.2) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded seven_zip (2.0.2) to: 'https://api.chef.io:443/organizations/devops'  
Uploaded windows (3.2.0) to: 'https://api.chef.io:443/organizations/devops'
```



After installing all the necessary dependent cookbooks, we used 'berks upload' to send the cookbook and all its dependencies to the Chef Server. This is again an easier method to manage dependencies instead of manually identifying the dependencies and then uploading each single cookbook at a time.

GL: Verifying the Cookbook has Been Uploaded



```
$ knife cookbook list
```

| | |
|-----------------|-------|
| apache | 0.1.0 |
| build-essential | 8.0.3 |
| company_web | 0.1.0 |
| cpu | 2.0.0 |
| haproxy | 3.0.4 |
| mingw | 2.0.1 |
| myhaproxy | 0.1.0 |
| myiis | 0.2.1 |
| ohai | 5.2.0 |
| poise | 2.8.1 |
| poise-service | 1.5.2 |
| seven_zip | 2.0.2 |



When that is complete you can verify that you've uploaded your cookbook and all of its dependencies.



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
- ✓ Upload cookbook to Chef Server
- ❑ Bootstrap a new node that runs the haproxy (load balancer) cookbook

GL: Navigate the chef-repo Directory



```
$ cd ~/chef-repo
```



Navigate to the 'chef-repo' directory before running the bootstrap command.

GL: Bootstrapping a New Linux Node

```
$ knife bootstrap IP -x USER -P PWD --sudo -N lb

Creating new client for lb
Creating new node for lb
Connecting to 54.242.217.190
54.242.217.190 Starting Chef Client, version 13.2.20
54.242.217.190 resolving cookbooks for run list: []
54.242.217.190 Synchronizing Cookbooks:
54.242.217.190 Installing Cookbook Gems:
54.242.217.190 Compiling Cookbooks...
54.242.217.190 [2017-10-31T01:48:06+00:00] WARN: Node lb has an empty run list.
54.242.217.190 Converging 0 resources
```

First you bootstrap a new node named lb. To communicate with the remote instance you need to provide it the credentials to connect to the system. Use the user name with the '-x' flag and the password '-P' flag. Include the '--sudo' flag because you are installing software and writing configuration to directories traditionally owned by the root user. Name the node with the '-N' flag.

GL: Viewing the New Node's Data



```
$ knife node show lb
```

```
Node Name: lb
Environment: _default
FQDN: ip-172-31-0-128.ec2.internal
IP: 54.210.192.12
Run List:
Roles:
Recipes:
Platform: centos 6.7
Tags:
```



After the node is bootstrapped, validate that it was added correctly to the organization.

GL: Defining a Run List for the Node



```
$ knife node run_list add lb "recipe[myhaproxy]"
```

```
lb:  
  run_list: recipe[myhaproxy]
```



Define an initial run list for that node to converge the default recipe of the myhaproxy cookbook.

GL: Validating the Run List has been Set



```
$ knife node show lb
```

```
Node Name: lb
Environment: _default
FQDN: ip-172-31-0-128.ec2.internal
IP: 54.210.192.12
Run List: recipe[myhaproxy]
Roles:
Recipes:
Platform: centos 6.7
Tags:
```



Ensure the run list has been set correctly for lb.



SSH Woes

Logging into systems is a pain. We can use another knife tool to allow us to send commands to all of our nodes.

We asked you to login to that remote node and run 'sudo chef-client' to apply the new run list defined for that node. This does in fact work but considering that we may need to execute this command for this node and many future nodes, it seems like a lot of windows and commands that we would need to execute.

GL: Viewing the Help for the ssh Subcommand



```
$ knife ssh --help
```

```
knife ssh QUERY COMMAND (options)
  -a, --attribute ATTR          The attribute to use for opening the connection
  - default depends on the context
  -s, --server-url URL         Chef Server URL
  --chef-zero-host HOST         Host to start chef-zero on
  --chef-zero-port PORT         Port (or port range) to start chef-zero on.
Port ranges like 1000,1010 or 8889-9999 will try all given ports until one works.
  -k, --key KEY                API Client Key
  --[no-]color                  Use colored output, defaults to false on
Windows, true otherwise
  -C, --concurrency NUM         The number of concurrent connections
  -c, --config CONFIG           The configuration file to use
  --defaults                     Accept default values for all questions
```



To make our lives easier, the 'knife' command provides a subcommand named 'ssh' that allows us to execute a command across multiple nodes that match a specified search query.

GL: Running a Command with Knife



```
$ knife ssh IP -m -x USERNAME -P PASSWORD "sudo chef-client"
```

```
ec2-34-230-9-106.compute-1.amazonaws.com Starting Chef Client, version 13.2.20
ec2-34-230-9-106.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-34-230-9-106.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-34-230-9-106.compute-1.amazonaws.com   - myhaproxy (0.2.0)
ec2-34-230-9-106.compute-1.amazonaws.com   - haproxy (3.0.4)
ec2-34-230-9-106.compute-1.amazonaws.com   - cpu (2.0.0)
ec2-34-230-9-106.compute-1.amazonaws.com   - build-essential (8.0.3)
ec2-34-230-9-106.compute-1.amazonaws.com   - seven_zip (2.0.2)
ec2-34-230-9-106.compute-1.amazonaws.com   - windows (3.2.0)
ec2-34-230-9-106.compute-1.amazonaws.com   - chai (5.2.0)
```



Run 'sudo chef-client' on your new lb node to apply the 'myhaproxy' cookbook.

GL: Viewing the Website is Being Proxied

URL of load balancer.

Output from the
iis_web server.



Point a web browser to the URL or public IP address of your load balancer. It should display the web page of the web server node that the load balancer is configured to serve.



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
- ✓ Upload cookbook to Chef Server
- ✓ Bootstrap a new node that runs the haproxy (load balancer) cookbook

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.

Discussion



What are the benefits of the Chef Super Market? And what are the drawbacks?

Is your team able to leverage community cookbooks? Is the team able to contribute to community cookbooks?

Why do you use a wrapper cookbook? When might you decide to not wrap the cookbook?

Answer these questions.

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



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.



CHEFTM

©2018 Chef Software Inc.

Managing Multiple Nodes

Create Another Apache Web Server and Add it as a Proxy Member

This section's goal is to have you bootstrap another node, this time a web server, and add it to the proxy members.

Objectives

After completing this module, you should be able to

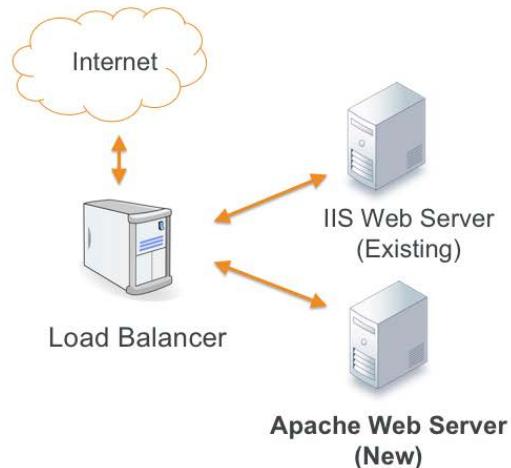
- Bootstrap, update the run_list, and run chef-client on a Linux node
- Append values to an attribute within a recipe
- Version a cookbook and upload it to the Chef Server

In this module you will learn how to bootstrap, update the run list, and run chef-client on a node. You will also learn how to update a default attribute within a recipe, version and upload a cookbook.

Managing User Traffic

You already configured the load balancer and one IIS web server node.

In this module you'll add another Apache web server node to the load balancer's pool of web servers it is directing traffic to.



After completing this module, you will have configured three nodes:

iis_web: An IIS web server

lb: The load balancer

apache_web: A Apache web server



Lab: Add a Linux Web Node

- Bootstrap a new Linux node giving it the name 'apache_web'
- Update the run list of the new node to include the company_web cookbook
- Run chef-client on that system
- Verify that the node's web server is functional

Now it's time to create a third node. The third node will be the apache web server node.

We will provide you with a new node for the following exercise.

Instructor Note:

Allow 10 minutes to complete this exercise

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

Because the student must use the IP and hostname of their web server, this must be manually added if they download the cookbook from GitHub.

Lab: Bootstrapping a New Linux Node



```
$ knife bootstrap IP -x USER -P PWD --sudo -N apache_web
```

```
Connecting to 52.90.49.196
52.90.49.196 -----> Existing Chef installation detected
52.90.49.196 Starting the first Chef Client run...
52.90.49.196 Starting Chef Client, version 13.2.20
52.90.49.196 resolving cookbooks for run list: []
52.90.49.196 Synchronizing Cookbooks:
52.90.49.196 Installing Cookbook Gems:
52.90.49.196 Compiling Cookbooks...
52.90.49.196 [2017-10-20T18:25:32+00:00] WARN: Node apache_web has an empty run list.
52.90.49.196 Converging 0 resources
52.90.49.196
52.90.49.196 Running handlers:
52.90.49.196 Running handlers complete
52.90.49.196 Chef Client finished, 0/0 resources updated in 07 seconds
```



Bootstrap the new node and name it apache_web.

Lab: Viewing the Details of the New Node



```
$ knife node show apache_web
```

```
Node Name: apache_web
Environment: _default
FQDN: ip-172-31-18-193.ec2.internal
IP: 52.90.49.196
Run List:
Roles:
Recipes:
Platform: centos 6.9
Tags:
```



Verify that you bootstrapped the node.



Lab: Add a Linux Web Node

- ✓ Bootstrap a new Linux node giving it the name 'apache_web'
- Update the run list of the new node to include the company_web cookbook
- Run chef-client on that system
- Verify that the node's web server is functional

Lab: Setting the Run List of the New Node



```
$ knife node run_list add apache_web "recipe[company_web]"
```

```
apache_web:  
  run_list: recipe[company_web]
```



Set the run list for this node by running the company_web cookbook's default recipe.



Lab: Add a Linux Web Node

- ✓ Bootstrap a new Linux node giving it the name 'apache_web'
- ✓ Update the run list of the new node to include the company_web cookbook
- ❑ Run chef-client on that system
- ❑ Verify that the node's web server is functional

Lab: Converging the Run List



```
$ knife ssh "name:apache_web" -x USERNAME -P PWD "sudo chef-client"
```

```
34.207.100.168 ----> Existing Chef installation detected
34.207.100.168 Starting the first Chef Client run...
34.207.100.168 Starting Chef Client, version 13.2.20
34.207.100.168 resolving cookbooks for run list: ["company_web"]
34.207.100.168 Synchronizing Cookbooks:
34.207.100.168   - apache (0.1.0)
34.207.100.168   - myiis (0.2.1)
34.207.100.168   - company_web (0.1.1)
34.207.100.168 Installing Cookbook Gems:
34.207.100.168 Compiling Cookbooks...
34.207.100.168 Converging 3 resources
34.207.100.168 Recipe: apache::server
34.207.100.168 * yum_package[httpd] action install (up to date)
```



Apply that run list by logging into that node and running sudo chef-client or remotely administer the node with the 'knife ssh' command as shown here. We can use search criteria like 'name:apache_web' rather than manually passing in the IP address.



Lab: Add a Linux Web Node

- ✓ Bootstrap a new Linux node giving it the name 'apache_web'
- ✓ Update the run list of the new node to include the company_web cookbook
- ✓ Run chef-client on that system
- ❑ Verify that the node's web server is functional

Lab: Verifying that the New Node Serves the Page



Verify that the node serves up the default html page that contains the node's platform, hostname, memory, and CPU speed.



Lab: Add a Linux Web Node

- ✓ Bootstrap a new Linux node giving it the name 'apache_web'
- ✓ Update the run list of the new node to include the company_web cookbook
- ✓ Run chef-client on that system
- ✓ Verify that the node's web server is functional



Lab: Update the Load Balancer

- Update the default recipe of the myhaproxy cookbook to include the new apache_web node as a member of the load balancing pool.
- Update the metadata for a minor change and upload the cookbook to the Chef Server
- Converge the load balancer
- Verify that the load balancer delivers traffic to both web server nodes.

Now that you have the third node, it is time to add that node to the member's list for the load balancer.

Instructor Note: Allow 15 minutes to complete this exercise

Lab: Displaying the New Node's IP and Hostname



```
$ knife node show apache_web -a cloud
```

```
apache_web:  
cloud:  
  local_hostname: ip-172-31-18-193.ec2.internal  
  local_ipv4: 172.31.18.193  
  local_ipv4_addrs: 172.31.18.193  
  provider: ec2  
  public_hostname: ec2-52-90-49-196.compute-1.amazonaws.com  
  public_ipv4: 52.90.49.196  
  public_ipv4_addrs: 52.90.49.196
```



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.

Lab: Adding the New Node to the Load Balancer

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

```
node.default['haproxy']['members'] = [{  
    'hostname' => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',  
    'ipaddress' => '52.8.71.11',  
    'port' => 80,  
    'ssl_port' => 80  
}, {  
    'hostname' => 'ec2-52-90-49-196.compute-1.amazonaws.com',  
    'ipaddress' => '52.90.49.196',  
    'port' => 80,  
    'ssl_port' => 80  
}  
]  
include_recipe 'haproxy::manual'
```



Add the second web server (apache_web) to the load balancer's (lb) members list. You may need to run 'knife node show apache_web -a cloud' to get the hostname and ipaddress values.



Lab: Update the Load Balancer

- ✓ Update the default recipe of the myhaproxy cookbook to include the new apache_web node as a member of the load balancing pool.
- ❑ Update the metadata for a minor change and upload the cookbook to the Chef Server
- ❑ Converge the load balancer
- ❑ Verify that the load balancer delivers traffic to both web server nodes.

Lab: Updating the Cookbook's Version

```
~/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 '0.2.0'
chef_version '>= 12.1' if respond_to?(:chef_version)

depends 'haproxy', '~> 3.0.0'
```

Update the minor version number in myhaproxy cookbook's metadata.

Lab: Uploading the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded myhaproxy (0.2.0) to:  
'https://api.chef.io:443/organizations/devops'  
Skipping ohai (5.2.0) (frozen)  
Skipping poise (2.8.1) (frozen)  
Skipping poise-service (1.5.2) (frozen)  
Skipping seven_zip (2.0.2) (frozen)  
Skipping windows (3.2.0) (frozen)
```



Run 'berks upload' to upload the myhaproxy cookbook to Chef Server.



Lab: Update the Load Balancer

- ✓ Update the default recipe of the myhaproxy cookbook to include the new apache_web node as a member of the load balancing pool.
- ✓ Update the metadata for a minor change and upload the cookbook to the Chef Server
- Converge the load balancer
- Verify that the load balancer delivers traffic to both web server nodes.

Lab: Converging the Load Balancer node



```
$ knife ssh 'name:lb' -x chef -P Cod3Can! 'sudo chef-client'
```

```
ec2-52-90-49-196.compute-1.amazonaws.com Starting Chef Client, version 13.2.20
ec2-34-230-9-106.compute-1.amazonaws.com Starting Chef Client, version 13.2.20
ec2-52-90-49-196.compute-1.amazonaws.com resolving cookbooks for run list: ["apache"]
ec2-52-90-49-196.compute-1.amazonaws.com Synchronizing Cookbooks:
  ec2-52-90-49-196.compute-1.amazonaws.com - apache (0.2.1)
  ec2-52-90-49-196.compute-1.amazonaws.com Installing Cookbook Gems:
  ec2-52-90-49-196.compute-1.amazonaws.com Compiling Cookbooks...
  ec2-52-90-49-196.compute-1.amazonaws.com Converging 3 resources
  ec2-52-90-49-196.compute-1.amazonaws.com Recipe: apache::server
  ec2-34-230-9-106.compute-1.amazonaws.com resolving cookbooks for run list: ["myhaproxy"]
  ec2-34-230-9-106.compute-1.amazonaws.com Synchronizing Cookbooks:
    ec2-54-210-192-12.compute-1.amazonaws.com - build-essential
```



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.

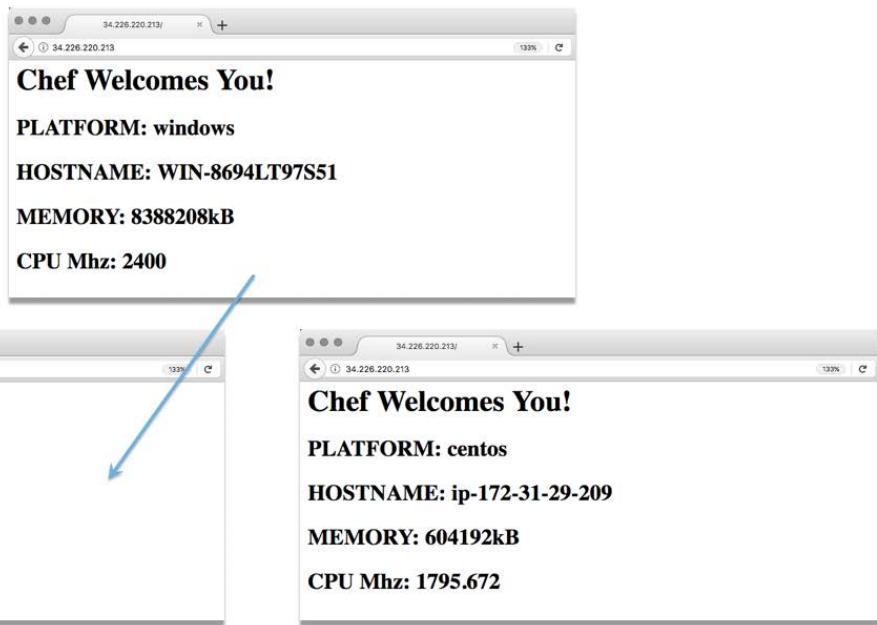
Within the output you should see the haproxy configuration file will update with a new entry that contains the information of the second member (apache_web).



Lab: Update the Load Balancer

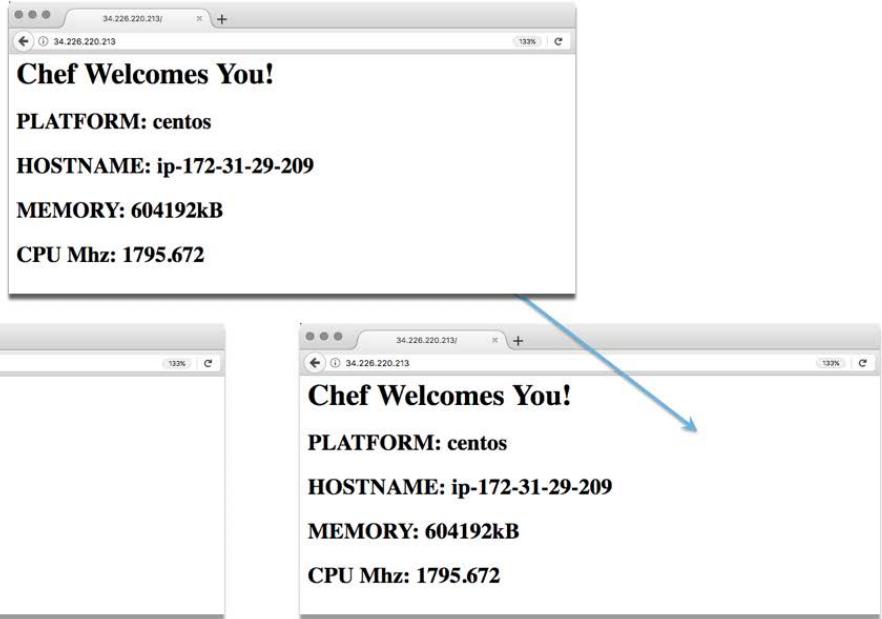
- ✓ Update the default recipe of the myhaproxy cookbook to include the new apache_web node as a member of the load balancing pool.
- ✓ Update the metadata for a minor change and upload the cookbook to the Chef Server
- ✓ Converge the load balancer
- ❑ Verify that the load balancer delivers traffic to both web server nodes.

Lab: Test the Load Balancer



Point a web browser to the URL of your 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.

Lab: Test the Load Balancer



Point a web browser to the URL of your 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.



Lab: Update the Load Balancer

- ✓ Update the default recipe of the myhaproxy cookbook to include the new apache_web node as a member of the load balancing pool.
- ✓ Update the metadata for a minor change and upload the cookbook to the Chef Server
- ✓ Converge the load balancer
- ✓ Verify that the load balancer delivers traffic to both web server nodes.

Instructor Note:

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

Because the student must use the IP and hostname of their web server, this must be manually added if they download the cookbook from GitHub.



Discussion

What is the process to set up a third web node?

What would the process be for removing a web node?

What is the most manual part of the process?

Answer these questions.

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

Q&A

What questions can we help you answer?





CHEFTM

©2018 Chef Software Inc.

Roles

Giving Your Nodes a Role

Objectives

After completing this module, you should be able to

- Create roles that will contain a particular run list of recipes
- Assign roles to nodes so you can better describe them and configure them in a similar manner.
- Set new node attributes values with a role

In this module you will give your nodes a role to better describe them so you can configure them in a similar manner.



Roles

A role describes a run list of recipes that are executed on the node.

A role may also define new defaults or overrides for existing cookbook attribute values.

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

Up until this point it has been a mouthful to describe the nodes within our organization.

The Chef Server allows us to create and manage roles. A role describes a run list of recipes that are executed on the node. A role may also define new defaults or overrides for existing cookbook attribute values. Similar to what we accomplished with the wrapper cookbook.

A node can have zero or more roles assigned to it.



Roles

When you assign a role to a node you do so in its run list.

This allows you to configure many nodes in a similar fashion.

When you assign a role to a node you do so in its run list. This allows us to configure many nodes in a similar fashion because we no longer need to re-create a long run list for each node--we simply give it a role or all the roles it needs to accomplish its desired function.



Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- Give our load balancer node a "load_balancer" Role
- Give our web nodes a "web_server" Role

In this section you will create a `load_balancer` role and assign it to the run list of `lb`. You will also will create a `web_server` role and assign it to the run list of `iis_web` and `apache_web`.

This is particularly powerful because we will no longer have to manage each of these identical nodes individually, instead we can make changes to the role that they share and all of the nodes that have this role will update accordingly.

Viewing the Help for Knife Role Subcommand



```
$ cd ~/chef-repo
$ knife role --help

** ROLE COMMANDS **

knife role bulk delete REGEX (options)
knife role create ROLE (options)
knife role delete ROLE (options)
knife role edit ROLE (options)
knife role env_run_list add [ROLE] [ENVIRONMENT] [ENTRY[,ENTRY]] (options)
knife role env_run_list clear [ROLE] [ENVIRONMENT]
knife role env_run_list remove [ROLE] [ENVIRONMENT] [ENTRIES]
knife role env_run_list replace [ROLE] [ENVIRONMENT] [OLD_ENTRY]
```



Return to the base of your Chef repository and then run 'knife role --help' to see the available commands. Similar to other commands, you can see that 'knife role' supports the ability to list currently-defined roles.

Viewing the Roles on the Chef Server



```
$ knife role list
```



When you run 'knife role list' you can see from its lack of response that you have no roles defined.

Locating the roles Directory



```
$ ls -l
```

```
-rw-r--r--@ 1 technotrainer  staff  2341 Oct 18 21:02 README.md
drwxr-xr-x@ 7 technotrainer  staff   238 Oct 18 16:09 cookbooks
drwxr-xr-x@ 5 technotrainer  staff   170 Oct 18 16:50 roles
```



Create a roles directory if necessary. If you are using the Chef Starter Kit this directory may already exist.

Defining the Load Balancer Role

```
~/chef-repo/roles/load_balancer.rb
```

```
name 'load_balancer'  
description 'Load Balancer'  
run_list 'recipe[myhaproxy]'
```



Create a file named `load_balancer.rb`. This is a ruby file that contains specific methods that allow you to express details about the role. You'll see that the role has a name, a description, and run list.

The name of the role as a practice will share the name of the ruby file unless it cannot for some reason. The name of the role should clearly describe what it attempts accomplish. The description of the role helps reinforce or clarify the intended purpose of the role. When selecting a role name that is not clear it is important that a helpful description is provided to help ensure everyone on the team understands its purpose. The run list defines the list of recipes that give the role its purpose. Currently the `load_balancer` role defines a single recipe - the `myhaproxy` cookbook's default recipe.

Uploading the Role to the Chef Server



```
$ knife role from file load_balancer.rb
```

```
Updated Role load_balancer!
```



Now you need to upload it to the Chef Server. This is done through the command 'knife role from file load_balancer.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file load_balancer.rb.

Viewing the Roles on the Chef Server



```
$ knife role list
```

```
load_balancer
```



With the role uploaded, it is time to validate that the Chef Server received it correctly. We can do that by again asking the Chef Server for a list of all the roles on the system.

Viewing the Details about the Role



```
$ knife role show load_balancer
```

```
chef_type:          role
default_attributes:
description:        Load Balancer
env_run_lists:
json_class:         Chef::Role
name:               load_balancer
override_attributes:
run_list:           recipe[myhaproxy]
```



You can ask for more details about a specific role using the above command. In this example we are requesting specific details about the role named `load_balancer`.

Viewing the Help for the knife node subcommand



```
$ 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)
```



Run 'knife node --help' to see its options.

Setting the Run List for the Node



```
$ knife node run_list set lb "role[load_balancer]"
```

```
lb:
```

```
  run_list: role[load_balancer]
```



The last step is to redefine the run list for lb. We want the run list to contain only the load_balancer role.

Previously, we used the command 'knife node run_list add' to append a new item to the existing run list. There is also a command that allows us to remove an item from the run list. There is a command that allows us to set the run list to a value provided. This will replace the existing run list with a new one that we provide.

Viewing the Details about the Node



```
$ knife node show lb
```

```
Node Name: lb
Environment: _default
FQDN: ip-172-31-25-169.ec2.internal
IP: 34.230.9.106
Run List: role[load_balancer]
Roles:
Recipes: myhaproxy, myhaproxy::default, haproxy::manual, haproxy::install_package
Platform: centos 6.9
Tags:
```



After you update the run list, you can verify that the node has the correctly-defined run list by running 'knife node show lb'.

Converging all Load Balancers



```
$ knife ssh "role:load_balancer" -x USER -P PWD "sudo chef-client"
```

```
ec2-34-230-9-106.compute-1.amazonaws.com Starting Chef Client, version 13.2.20
ec2-34-230-9-106.compute-1.amazonaws.com resolving cookbooks for run list:
[ "myhaproxy" ]
ec2-34-230-9-106.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-34-230-9-106.compute-1.amazonaws.com   - myhaproxy (0.2.1)
ec2-34-230-9-106.compute-1.amazonaws.com   - haproxy (3.0.4)
ec2-34-230-9-106.compute-1.amazonaws.com   - cpu (2.0.0)
ec2-34-230-9-106.compute-1.amazonaws.com   - build-essential (8.0.3)
ec2-34-230-9-106.compute-1.amazonaws.com   - seven_zip (2.0.2)
ec2-34-230-9-106.compute-1.amazonaws.com   - windows (3.2.0)
ec2-34-230-9-106.compute-1.amazonaws.com   - chai (5.2.0)
```



You can use 'knife ssh' to run 'sudo chef-client' on all the nodes again to ensure that nothing has changed.

In this instance we only interested in having lb run the command so we can get a little more creative with the search criteria and find nodes with the role load_balancer. In this case there is only one result.

Within the results, nothing should change. Switching over to the role did not change the fundamental recipes that were applied to the node.



Lab: Define the `web_server` Role

- Create a `web_server` role using the `company_web` cookbook as it's run list
- Upload the role to the Chef Server
- Assign the `iis_web` node and `apache_web` node the `web_server` role
- Converge both web servers

As a lab, create a `web_server` role that uses the `company_web`'s default recipe as its run list and assign this to both of your web servers.

Defining the Web Server Role

```
~/chef-repo/roles/web_server.rb
```

```
name 'web_server'  
description 'Apache and IIS Web Servers'  
run_list 'recipe[company_web]'
```



First we create a file named `web_server.rb` in the `roles` directory.

The name of the role is `web`. The description should be `Web Server`. The run list you define should contain the `company_web` cookbook's default recipe.



Lab: Define the `web_server` Role

- ✓ Create a `web_server` role using the `company_web` cookbook as it's run list
- ❑ Upload the role to the Chef Server
- ❑ Assign the `iis_web` node and `apache_web` node the `web_server` role
- ❑ Converge both web servers

Upload the Web Server Role



```
$ knife role from file web_server.rb
```

```
Updated Role web_server
```



You need to share the role with the Chef Server so upload that file.

Use the command 'knife role from file web_server.rb'. 'knife' knows where to look for that role to upload it.

Verifying the Roles on the Chef Server



```
$ knife role list
```

```
load_balancer  
web_server
```



Verify that the role can be found on the Chef Server.

Viewing the Details about the Role



```
$ knife role show web_server
```

```
chef_type:          role
default_attributes:
description:        Apache and IIS Web Servers
env_run_lists:
json_class:         Chef::Role
name:               web_server
override_attributes:
run_list:           recipe[company_web]
```



Verify specific information about the role. Specifically, does it have the run list that we defined?



Lab: Define the `web_server` Role

- ✓ Create a `web_server` role using the `company_web` cookbook as it's run list
- ✓ Upload the role to the Chef Server
- ❑ Assign the `iis_web` node and `apache_web` node the `web_server` role
- ❑ Converge both web servers

Setting the IIS Web Node's Run List



```
$ knife node run_list set iis_web "role[web_server]"
```

```
iis_web:  
  run_list: role[web_server]
```



Set `iis_web`'s run list to be the `web_server` role.

Setting the Apache Web Node's Run List



```
$ knife node run_list set apache_web "role[web_server]"
```

```
apache_web:  
  run_list: role[web_server]
```



And we then set apache_web's run list to be the web_server role.



Lab: Define the `web_server` Role

- ✓ Create a `web_server` role using the `company_web` cookbook as it's run list
- ✓ Upload the role to the Chef Server
- ✓ Assign the `iis_web` node and `apache_web` node the `web_server` role
- ❑ Converge both web servers

Converging all Web Nodes (Linux)



```
$ knife ssh "role:web_server AND os:linux" -x USER -P PWD "sudo chef-client"
```

```
ec2-184-73-96-131.compute-1.amazonaws.com Starting Chef Client, version
13.2.20
ec2-184-73-96-131.compute-1.amazonaws.com resolving cookbooks for run list:
["company_web"]
ec2-184-73-96-131.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-184-73-96-131.compute-1.amazonaws.com   - company_web (0.1.0)
ec2-184-73-96-131.compute-1.amazonaws.com   - myiis (0.2.1)
ec2-184-73-96-131.compute-1.amazonaws.com   - apache (0.1.0)
ec2-184-73-96-131.compute-1.amazonaws.com Installing Cookbook Gems:
ec2-184-73-96-131.compute-1.amazonaws.com Compiling Cookbooks...
ec2-184-73-96-131.compute-1.amazonaws.com Converging 1 resources
ec2-184-73-96-131.compute-1.amazonaws.com Recipe: company_web::default
```



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 Linux nodes with the role set to web_server.

Converge All Web Nodes (Windows)



```
> knife winrm "role:web_server AND os:windows" -x USER -P PWD "chef-client"
```

```
ERROR: Network Error. A connection attempt failed because the connected party did  
not properly respond after a period of time, or established connection failed  
because connected host has failed to respond. - connect(2) for "172.31.10.176" port  
5985 (172.31.10.176:5985)  
Check your knife configuration and network settings
```

All nodes that have the web role

We can use knife winrm to log into multiple remote hosts and run command but it cannot seem to connect. Why?

To verify that everything is working the same as before, run 'knife winrm'. In this instance the query syntax is going to find all Windows nodes with the role set to `web_server` using the specified attribute on the node to match against. But there seems to be an issue...

Capture Nodes' IP

```
💻 > knife node show iis_web -a cloud.public_ipv4  
  
iis_web:  
  cloud.public_ipv4: 34.196.63.231
```

Issue: We will use the `node['cloud']['public_ipv4']` attribute value

For us to use 'knife winrm' and resolve the public IP address, we will need to add the '`-a cloud.public_ipv4`' flag.

Converge Web Node (Windows)



```
> knife winrm "role:web_server AND os:windows" -a cloud.public_ipv4 -x USER  
-P PWD "chef-client"
```

```
54.159.197.193 Starting Chef Client, version 13.6.0  
54.159.197.193  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: *** Chef 13.6.0 ***  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: Platform: x64-mingw32  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: Chef-client pid: 496  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: The plugin path  
C:\chef\ohai\plugins does not exist. Skipping...  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Run List is [role[web_server]]  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Run List expands to [company_web]  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Starting Chef Run for iis_web  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Running start handlers  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Start handlers complete.
```



To verify that everything is working the same as before, run 'knife winrm'. In this instance the query syntax is going to find all Windows nodes with the role set to web_server. We include the '-a cloud.public_ipv4' flag and the command now works correctly.



Lab: Define the `web_server` Role

- ✓ Create a `web_server` role using the `company_web` cookbook as it's run list
- ✓ Upload the role to the Chef Server
- ✓ Assign the `iis_web` node and `apache_web` node the `web_server` role
- ✓ Converge both web servers



Roles for Everyone

We will give our nodes a role to better describe them and so we can configure them in a similar manner.

Objective:

- ✓ Give our load balancer node a "load_balancer" Role
- ✓ Give our web nodes a "web_server" Role

In this section you will create a `load_balancer` role and assign it to the run list of `lb`. You will also will create a `web_server` role and assigned it to the run list of `iis_web` and `apache_web`.

This is particularly powerful because we will no longer have to manage each of these identical nodes individually, instead we can make changes to the role that they share and all of the nodes that have this role will update accordingly.

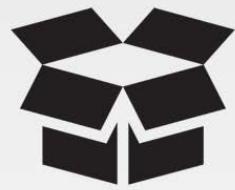


Update the Company Name

Wait!

Your company was just acquired by **E Corp** and now your website must be updated to display the correct company name.

Oh no! Our company was just bought out by E Corp and our website is now displaying incorrect information. Let's override this with our new role.



Node Attribute Precedence

| | Attribute Files | Node / Recipe | Environment | Role |
|----------------|-----------------|---------------|-------------|------|
| default | 1 | 2 | 3 | 4 |
| force_default | 5 | 6 | | |
| normal | 7 | 8 | | |
| override | 9 | 10 | 12 | 11 |
| force_override | 13 | 14 | | |
| automatic | | 15 | | |

©2018 Chef Software Inc.

14-34



We set the 'company_name' node attribute within an attribute file with a default precedence. If we want to change the value associated with this node attribute to reflect our new reality, we have the option to do so from our newly created web_server role.

GL: Updating the Web Server Role

```
~/chef-repo/roles/web_server.rb

name 'web_server'
description 'Apache and IIS Web Servers'
run_list 'recipe[company_web]'
default_attributes 'company_web' => {'company_name' => 'E Corp'}
```



By adding this line to our web_server.rb file, when we upload this to the Chef Server we should see this information update our welcome page after we run chef-client on our web server nodes.

Upload the Web Server Role



```
$ knife role from file web_server.rb
```

```
Updated Role web_server
```



You need to share the role with the Chef Server so upload that file.

Use the command 'knife role from file web_server.rb'. 'knife' knows where to look for that role to upload it.

Converge Web Nodes



```
$ knife winrm "name:iis_web" -a cloud.public_ipv4 -x USER -P PWD "chef-client"  
$ knife ssh "name:apache_web" -x USER -P PWD "sudo chef-client"  
  
54.159.197.193 Starting Chef Client, version 13.6.0  
54.159.197.193  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: *** Chef 13.6.0 ***  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: Platform: x64-mingw32  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: Chef-client pid: 496  
54.159.197.193 [2017-10-31T02:49:21+00:00] INFO: The plugin path  
C:\chef\ohai\plugins does not exist. Skipping...  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Run List is [role[web_server]]  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Run List expands to [company_web]  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Starting Chef Run for iis_web  
54.159.197.193 [2017-10-31T02:49:24+00:00] INFO: Running start handlers
```



Converging both our `iis_web` and `apache_web` nodes will allow us to verify our change.

Verify the Changes to Web Page



And we indeed we see that 'E Corp Welcomes You!'



Discussion

What are the benefits of using roles? What are the drawbacks?

Roles can contain roles. How many of these nested roles would make sense?

Answer these questions.

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

Q&A

What questions can we help you answer?





CHEFTM

©2018 Chef Software Inc.

Search

Update a Cookbook to Dynamically Use Nodes with the Server Role

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 Array and Ruby Hash dynamically

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 web role.



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.

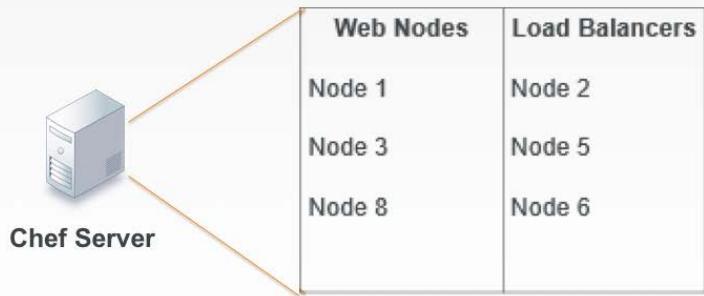
To add new servers as load balancer members, we would need to bootstrap a new web server and then update our myhaproxy cookbook to include that new web server. But that seems dramatically inefficient to have to update a cookbook recipe.

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 Server and Search

Chef 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 Server.



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

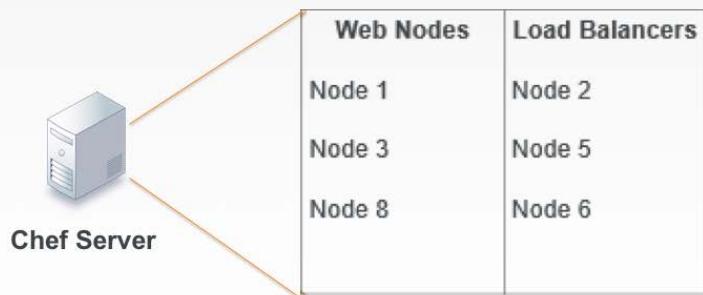
https://docs.chef.io/chef_search.html#search-indexes

The Chef Server maintains a representation of all the nodes within an infrastructure and provides a way for us to discover these systems through Search.

Search is a service discovery tool that allows us to query the Chef Server across a few indexes. One such index is on our nodes.

The Chef Server and Search

We can ask the Chef 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 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 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 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: lb
Environment: _default
FQDN: ip-172-31-23-107.ec2.internal
IP: 34.226.220.213
Run List: recipe[myhaproxy]
Roles:
Recipes: myhaproxy, myhaproxy::default, haproxy::manual, haproxy::install_package
Platform: centos 6.9
Tags:

Node Name: iis_web
Environment: _default
FQDN: WIN-8694LT97S51.ec2.internal
IP: 34.229.225.40
Run List: role[web_server]
Roles:
Recipes: company_web, company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:
```



So run the command `knife search node "*:*`" – you should see all nodes returned.

However, querying and returning every node is not exactly what we need to solve our current problem. In our scenario we want only to return a subset of our nodes (only the nodes that are web servers) and also we only want the IP address of those nodes, not any the other attributes returned.

Let's examine the search criteria more so we can understand how it works and how we can use it to find a subset of the nodes, and only the info we want for those nodes.

Demo: View Information for All Server Nodes



```
> knife search node "role:web_server"
```

```
Node Name: iis_web
Environment: _default
FQDN: WIN-8694LT97S51.ec2.internal
IP: 34.229.225.40
Run List: role[web_server]
Roles:
Recipes: company_web, company_web::default, myiis::default, myiis::server
Platform: windows 6.3.9600
Tags:

Node Name: apache_web
Environment: _default
FQDN: ip-172-31-29-209.ec2.internal
IP: 34.207.100.168
Run List: role[web_server]
Roles:
Recipes: company_web, company_web::default, apache::default, apache::server
Platform: centos 6.9
Tags:
```



We can query the Chef Server for specific nodes such as those with the assigned role of 'web_server'

Demo: Return Public Hostname for Servers



```
> knife search node "role:web_server" -a cloud.public_hostname
```

```
iis_web:  
  cloud.public_hostname: ec2-34-229-225-40.compute-1.amazonaws.com  
  
apache_web:  
  cloud.public_hostname: ec2-34-207-100-168.compute-1.amazonaws.com
```



The '-a' flag allows you to specify a particular attribute from those nodes.

Search Syntax within a Recipe

```
all_web_nodes = search('node', 'role:web_server')
```

creates and names a variable

assigns the value of the operation on the right into the variable on the left

invokes the search method

the index or items to search

the search criteria - key:value

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 Server. These are: nodes; roles; and 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 'all_web_nodes'. Variables within Ruby are created immediately when you assign them.

Search Syntax within a Recipe

```
all_web_nodes = search('node', 'role:web_server')
```

Search the Chef Server for all node objects that have the role equal to 'web_server' and store the results into a local variable named "all_web_nodes".

This example syntax could be translated to mean: Search the Chef Server for all node objects that have the role equal to 'web_server' and store the results into a local variable named 'all_web_nodes'.

Hard Coding Example

```
node.default['haproxy']['members'] = [{  
    'hostname' => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',  
    'ipaddress' => '52.8.71.11',  
    'port' => 80,  
    'ssl_port' => 80  
},  
{  
    'hostname' => 'ec2-54-176-64-173.us-west-1.compute.amazonaws.com',  
    'ipaddress' => '54.175.46.48',  
    'port' => 80,  
    'ssl_port' => 80  
}  
]  
include_recipe 'haproxy::manual'
```

Previously, we had been hard coding the hostname and ipaddress values in our wrapped myhaproxy recipe. We can request these values from the Chef Server through the `knife node show` command. The hostname and ipaddress values are captured by Ohai and sent to the Chef Server. On the Chef 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'.



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 web_server role
- 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 web_server role.

Instructor Note:

The GitHub repo for the 1.0.0 version of the 'myhaproxy' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-myhaproxy.git>

Removing the Hard-Coded Members

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
node.default['haproxy']['members'] = [{  
    'hostname' => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',  
    'ipaddress' => '52.8.71.11',  
    'port' => 80,  
    'ssl_port' => 80  
},  
{  
    'hostname' => 'ec2-54-176-64-173.us-west-1.compute.amazonaws.com',  
    'ipaddress' => '54.175.46.48',  
    'port' => 80,  
    'ssl_port' => 80  
}  
]  
include_recipe 'haproxy::manual'
```



Edit the 'myhaproxy' cookbook's default recipe and remove the current default recipe where you hard-coded the members.

Using Search to Find the Web Servers

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

```
all_web_nodes = search('node', 'role:web_server')

#TODO: Convert all found nodes into hashes with ipaddress,
#      hostname, port, ssl_port
#TODO: Assign all the hashes to the node's haproxy members
#      attribute.

include_recipe 'haproxy::manual'
```



Replace it with an updated recipe that searches for all nodes that have the 'web_server' role defined.

The search method's first parameter is asking the Chef Server to look at all the nodes within our organization.

The search method's second parameter is asking the Chef 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.

Creating an Array to Store the Converted Members

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

all_web_nodes = search('node', 'role:web_server')

members = []

#TODO: Convert all found nodes into hashes with ipaddress\
#      hostname, port, ssl_port

node.default['haproxy']['members'] = members

include_recipe 'haproxy::manual'
```



Unfortunately we cannot simply assign our array of web nodes into the haproxy's `members` attribute because it needs a hash that contains the keys 'hostname', 'ipaddress', 'port', and 'ssl_port'. We will need to convert each of the web node objects into a structure that the haproxy member's attribute expects.

First we create an empty array and assign that empty array into a local variable named `members`. `members` is an array that we will populate with the hashes we will create later; until then we will write a TODO for us. Then we will assign that array into the `node.default['haproxy']['members']`.

Populating the Members with Each New Member

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
all_web_nodes = search('node', 'role:web_server')

members = []

all_web_nodes.each do |web_node|
  member = {}

  # TODO: Populate the hash with hostname, ipaddress, port, and
  #        ssl_port

  members.push(member)
end

node.default['haproxy']['members'] = members
```

include_recipe 'haproxy::manual'



So we need to loop through the array of all the web nodes stored in `all_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 convert them into a hash object.

So every member of the array is visited and every member of the array runs through the block of code.

Populating the Hash with Node Details

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
# ... BEFORE THE LOOP IN THE RECIPE ...

all_web_nodes.each do |web_node|
  member = {
    'hostname' => web_node['cloud']['public_hostname'],
    'ipaddress' => web_node['cloud']['public_ipv4'],
    'port' => 80,
    'ssl_port' => 80
  }
  members.push(member)
end
```

... AFTER THE LOOP IN THE RECIPE ...



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 `all_web_nodes`. Each web node in the array is sent through the block. When inside the block of code it is referred to as `web_node`. Inside the block the first thing that is created is another local variable named `member` which is assigned a hash that contains the web_node's hostname and the web_node's ipaddress. Then the local variable `member`, which contains that hash is pushed into the array of members. This adds the member to the end of the array. When we are done looping through every web node the `members` array contains a list of all these hash objects.

Viewing the Final Recipe

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```
all_web_nodes = search('node','role:web_server')

members = []

all_web_nodes.each do |web_node|
  member = {
    'hostname' => web_node['cloud']['public_hostname'],
    'ipaddress' => web_node['cloud']['public_ipv4'],
    'port' => 80,
    'ssl_port' => 80
  }
  members.push(member)
end

node.default['haproxy']['members'] = members

include_recipe 'haproxy::manual'
```



Instructor Note:

The GitHub repo for the 1.0.0 version of the 'myhaproxy' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-myhaproxy.git>

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 web_server role
- ❑ 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

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 '>= 12.1' if respond_to?(:chef_version)

depends 'haproxy', '~> 3.0.0'
```



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 web_server role
- ✓ 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

Uploading the Cookbook



```
$ berks upload
```

```
Uploaded build-essential (2.2.3) to:  
'https://api.opscode.com:443/organizations/steveessentials2'  
Uploaded cpu (0.2.0) to:  
'https://api.opscode.com:443/organizations/steveessentials2'  
Uploaded haproxy (1.6.6) to:  
'https://api.opscode.com:443/organizations/steveessentials2'  
Uploaded myhaproxy (1.0.0) to:  
'https://api.opscode.com:443/organizations/steveessentials2'  
PS C:\Users\sdelfante\chef-repo\cookbooks\myhaproxy>
```



Upload the cookbook using the `berks upload` command.

Instructor Note: During the course the learner may find they have a mistake with the cookbook and need to re-upload the cookbook. Berkshelf will 'freeze' the versions of the cookbooks that you upload. This is to prevent you from accidentally overriding cookbooks that you may have already created. It is a best practice to not re-upload a cookbook again with new changes if they share the same version. During this course, however, it is important that the version numbers be aligned to make future sections work correctly so it OK to do in the training environment. To re-upload a cookbook with Berkshelf replacing the existing cookbook can be done with `berks upload --force`



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 web_server role
- ✓ 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 web role.

Converging the Load Balancer Node



```
$ knife ssh "role:load_balancer" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list: ["myhaproxy"]
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com   - build-essential
ec2-54-210-192-12.compute-1.amazonaws.com   - cpu
ec2-54-210-192-12.compute-1.amazonaws.com   - haproxy
ec2-54-210-192-12.compute-1.amazonaws.com   - myhaproxy
ec2-54-210-192-12.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-192-12.compute-1.amazonaws.com Converging 9 resources
```



Use `knife ssh` and ask only the nodes with the role `load_balancer` to run `sudo chef-client`. This is more efficient than targeting all of the nodes as we did before and more accurate than targeting the lb "role:lb".

This ensures that all nodes that are also load balancers check in with the Chef Server--similar to how we are targeting only the web server nodes in the recipe.

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 web_server role
- ✓ 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

E Corp Welcomes You!

PLATFORM: windows

HOSTNAME: WIN-DQFQCUFHDCP

MEMORY: 1048176kB

CPU Mhz: 2400

E Corp Welcomes You!

PLATFORM: windows

HOSTNAME: WIN-DQFQCUFHDCP

MEMORY: 1048176kB

CPU Mhz: 2400

E Corp Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-26-186

MEMORY: 604192kB

CPU Mhz: 1799.999

©2018 Chef Software Inc. 15-28



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.

E Corp Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-26-186

MEMORY: 604192kB

CPU Mhz: 1799.999

E Corp Welcomes You!

PLATFORM: windows

HOSTNAME: WIN-DQFQCUFHDCP

MEMORY: 1048176kB

CPU Mhz: 2400

E Corp Welcomes You!

PLATFORM: centos

HOSTNAME: ip-172-31-26-186

MEMORY: 604192kB

CPU Mhz: 1799.999

©2018 Chef Software Inc. 15-29 

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 web_server role
- ✓ 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

Instructor Note:

The GitHub repo for the 1.0.0 version of the 'myhaproxy' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-myhaproxy.git>

Discussion



What happens when new web nodes are added to the organization?
Removed?

What happens if you were to terminate a web node instance without
removing it from the Chef Server?

Answer these questions.

"Terminate" here means to turn off the machine or have the cloud provider disable the machine so that it is no longer online and network addressable.

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

Q&A

What questions can we help you answer?





©2018 Chef Software Inc.

Running chef-client as a Service

Introducing the chef-client cookbook

©2018 Chef Software Inc.

16-1



Lesson Objectives

After completing this module, you should be able to:

- Use knife to work with the Chef Supermarket site API
- Override community cookbook defaults using wrapper cookbooks
- Run chef-client as a service/task

In this module we are setting up chef-client to run as a service on our Linux nodes and a task on our windows machine.



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"
```

```
ec2-54-242-217-190.compute-1.amazonaws.com client.pem  client.rb  first-boot.json  ohai/
ec2-184-73-96-131.compute-1.amazonaws.com  client.pem  client.rb  first-boot.json  ohai/
```



chef-client looks for its configuration information in the directory '/etc/chef' by default – although this is configurable itself!

This directory contains, not only its own configuration file (which we'll look at shortly), but also its key to authenticate with the Chef Server, an Ohai plugins directory and a first-boot.json file. The first-boot.json file is generated from the workstation as part of the initial knife bootstrap subcommand, and contains the initial runlist that chef-client should run after Chef has been installed, and before it first registers with the Chef Server.

Demo: View chef-client config File (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD \
"cat /etc/chef/client.rb"
```

```
ec2-184-73-96-131.compute-1.amazonaws.com chef_server_url
"https://api.chef.io/organizations/2017_october_30"
ec2-184-73-96-131.compute-1.amazonaws.com validation_client_name "chef-validator"
ec2-184-73-96-131.compute-1.amazonaws.com log_location STDOUT
ec2-184-73-96-131.compute-1.amazonaws.com node_name "apache_web"
ec2-184-73-96-131.compute-1.amazonaws.com
ec2-54-242-217-190.compute-1.amazonaws.com chef_server_url
"https://api.chef.io/organizations/2017_october_30"
ec2-54-242-217-190.compute-1.amazonaws.com validation_client_name "chef-validator"
ec2-54-242-217-190.compute-1.amazonaws.com log_location STDOUT
ec2-54-242-217-190.compute-1.amazonaws.com node_name "lb"
ec2-54-242-217-190.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 client, in that they both communicate with the Chef Server over the API - the file '/etc/chef/client.rb' is the equivalent to the 'knife.rb' file on the workstation.

Demo: View chef-client config Directory (Windows)



```
$ knife winrm "os:windows" -x USER -P PWD  
-a cloud.public_ipv4 "dir C:\chef"
```

```
54.159.197.193  Directory of C:\chef  
54.159.197.193  
54.159.197.193 10/31/2017  12:59 AM    <DIR>          .  
54.159.197.193 10/31/2017  12:59 AM    <DIR>          ..  
54.159.197.193 10/31/2017  12:59 AM    <DIR>          backup  
54.159.197.193 10/30/2017  11:33 PM    <DIR>          cache  
54.159.197.193 10/30/2017  11:30 PM      1,706 client.pem  
54.159.197.193 10/30/2017  11:30 PM      335 client.rb  
54.159.197.193 10/30/2017  11:30 PM      17 first-boot.json  
54.159.197.193 02/01/2016  06:41 AM    <DIR>          ohai  
54.159.197.193 10/30/2017  11:28 PM      316 wget.ps1  
54.159.197.193 10/30/2017  11:28 PM      1,923 wget.vbs
```



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"
```

```
54.159.197.193 chef_server_url "https://api.chef.io/organizations/2017_october_30"  
54.159.197.193  
54.159.197.193 validation_client_name "chef-validator"  
54.159.197.193 file_cache_path "c:/chef/cache"  
54.159.197.193 file_backup_path "c:/chef/backup"  
54.159.197.193 cache_options ({:path => "c:/chef/cache/checksums", :skip_expires => true})  
54.159.197.193 node_name "iis_web"  
54.159.197.193 log_level :info  
54.159.197.193 log_location STDOUT
```





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.



Lets Examine the `chef-client` Cookbook

We're going to use one recipe on our node from the `chef-client` cookbook.

`chef-client::service`
(via `chef-client::default`)

Let's take a closer look at the service recipe of the `chef-client` cookbook.

GL: View the chef-client::default Recipe

```
~/berkshelf/cookbooks/chef-client-<version>/recipes/default.rb

#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
if platform?('windows')
  include_recipe 'chef-client::task'
else
  include_recipe 'chef-client::service'
end
```

The default recipe just makes a call to the recipe 'chef-client::service'--this is the recipe that will set chef-client to run as a service.

GL: View the chef-client::service Recipe

```
~/berkshelf/cookbooks/chef-client-<version>/recipes/service.rb
```

- The recipe supports a number of **service** providers and styles.
- It works on a lot of **platforms**.
- Everything is controllable through **attributes**.

```
supported_init_styles = %w(
  bsd
  init
  launchd
  smf
  src
  systemd
  upstart
  windows
)

init_style = node['chef_client']['init_style']

# Services moved to recipes
if supported_init_styles.include? init_style
  include_recipe "chef-client::#{init_style}_service"
else
  log 'Could not determine service init style, manual
  intervention required to start up the chef-client
  service.'
end
```

There's a lot to this recipe so we won't cover it all in detail.

There are several init styles that can be selected by setting the node attribute in the recipe. "init" is the default, and what we're going to use. Also available: smf, upstart, arch, runit, bluepill, daemontools, winsw. "cron" is a separate recipe since it is not technically running as a "service".



Introducing chef-client Cookbook

We will use the chef-client community cookbook to configure chef-client on each of our nodes to run periodically

Objective:

- Upload chef-client wrapper cookbook to Chef Server
- Configure each of our nodes to run chef-client as a service

Instructor Note:

The GitHub repo for the 'mychef_client' cookbook can be found at:
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/>

Create mychef_client Wrapper Cookbook



```
$ cd chef-repo  
$ chef generate cookbook cookbooks/mychef_client
```

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

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



Change to your chef-repo directory and then generate your new cookbook.

Instructor Note: The generate cookbook command needs the path in the name of the cookbook here to ensure that it generates the cookbook in the cookbooks directory. Without it the cookbook will be generated within the root of the chef repository. If that happens simply have the learner move the newly generated cookbook into the cookbooks directory.

GL: Update mychef_client metadata.rb

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'

Edit mychef_client Default Recipe

cookbooks/mychef_client/recipes/default.rb

```
#  
# Cookbook:: mychef_client  
# Recipe:: default  
#  
# Copyright:: 2017, The Authors, All Rights Reserved.  
  
include_recipe 'chef-client::default'
```

This recipe just calls the recipe `chef-client::default`



Install the Dependencies



```
$ cd cookbooks/mychef_client  
$ berks install
```

```
Resolving cookbook dependencies...  
Fetching 'mychef_client' from source at .  
Fetching cookbook index from https://supermarket.chef.io...  
Installing chef-client (9.0.0)  
Using windows (3.2.0)  
Installing logrotate (2.2.0)  
Installing compat_resource (12.19.0)  
Using mychef_client (0.1.0) from source at .  
Installing cron (4.2.0)  
Using ohai (5.2.0)
```



Demo: View Berksfile.lock



```
$ cat Berksfile.lock
```

```
DEPENDENCIES
  mychef_client
    path: .
    metadata: true

GRAPH
  chef-client (9.0.0)
  cron (>= 2.0.0)
  logrotate (>= 1.9.0)
  windows (>= 2.0.0)
  compat_resource (12.19.0)
  ...
  ...
```



Upload Cookbooks to Chef Server



```
$ berks upload
```

```
/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-
5.1.1/lib/ridley/client.rb:271:in `server_url': Ridley::Client#url_prefix at
/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-5.1.1/lib/ridley/client.rb:79
forwarding to private method Celluloid::PoolManager#url_prefix

Uploaded chef-client (9.0.0) to:
'https://api.chef.io:443/organizations/2017_october_30'

/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-
5.1.1/lib/ridley/client.rb:271:in `server_url': Ridley::Client#url_prefix at
/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-5.1.1/lib/ridley/client.rb:79
forwarding to private method Celluloid::PoolManager#url_prefix

Uploaded compat_resource (12.19.0) to:
'https://api.chef.io:443/organizations/2017_october_30'

/opt/chefdk/embedded/lib/ruby/gems/2.4.0/gems/ridley-
5.1.1/lib/ridley/client.rb:271:in `server_url': Ridley::Client#url_prefix at
```



View Cookbooks on Chef Server



```
$ knife cookbook list
```

| | |
|-----------------|---------|
| apache | 0.3.0 |
| chef-client | 7.0.0 |
| compat_resource | 12.16.2 |
| cron | 3.0.0 |
| haproxy | 0.2.1 |
| logrotate | 2.1.0 |
| mychef_client | 0.1.0 |
| ohai | 4.2.2 |
| windows | 2.1.1 |
| workstation | 0.2.1 |





chef-client as a Service

We will add the chef-client default recipe to the roles for each node.

GL: Create the new 'base' role

`chef-repo/roles/base.rb`

```
name 'base'  
description 'Base Role'  
run_list 'recipe[mychef_client]'
```



Let's create a 'base' role. This role will be nested in our other roles and contains a run list of policy that we want to have run on all of our nodes.

Upload it to the Chef Server



```
$ knife role from file base.rb
```

```
Updated Role base
```



Now you need to upload it to the Chef Server. This is done through the command 'knife role from file base.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named knife role from file base.rb.

GL: Add the base Role to load_balancer Role

```
chef-repo/roles/load_balancer.rb
name 'load_balancer'
description 'Load Balancer'
run_list 'role[base]', 'recipe[myhaproxy]'
```



By adding the 'base' role to our load_balancer role, we have now nested this role and all the policy contained in 'base' will now be included in the load_balancer role.

GL: Add the base Role to web_server Role

chef-repo/roles/web_server.rb

```
name 'web_server'  
description 'Apache and IIS Web Servers'  
run_list 'role[base]', 'recipe[company_web]'  
default_attributes 'company_web' => {'company_name' => 'E Corp'}
```



We do the same for our web_server role.

Upload the roles Files



```
$ cd ~/chef-repo  
$ knife role from file load_balancer.rb web_server.rb
```

```
Updated Role load_balancer  
Updated Role web_server
```



You need to share the role with the Chef Server so upload that file.

Use the command 'knife role from file load_balancer.r web_server.rb'. 'knife' knows where to look for that role to upload it.

Converge All Nodes (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-162-31-253.compute-1.amazonaws.com  Starting Chef Client, version 12.13.37
ec2-54-167-232-148.compute-1.amazonaws.com Starting Chef Client, version 12.13.37
ec2-54-205-59-139.compute-1.amazonaws.com  Starting Chef Client, version 12.13.37
ec2-54-167-232-148.compute-1.amazonaws.com resolving cookbooks for run list: ["chef-client", "apache"]
ec2-54-205-59-139.compute-1.amazonaws.com  resolving cookbooks for run list: ["chef-client", "apache"]
ec2-54-162-31-253.compute-1.amazonaws.com  resolving cookbooks for run list: ["chef-client", "haproxy"]
ec2-54-167-232-148.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-205-59-139.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-162-31-253.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-54-205-59-139.compute-1.amazonaws.com      - chef-client (5.0.0)
...
...
```



To verify that everything is working the same as before, run 'knife ssh' for both of these nodes. In this instance the query syntax is going to find all nodes with the OS of linux.

Verify chef-client is Running (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD  
"ps awux | grep chef-client"
```

```
ec2-514-88-185-159.compute-1.amazonaws.com root      10369  0.0 10.1 266928 61116 ?  
S1  12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c  
/etc/chef/client.rb -P /var/run/chef/client.pid -i 1800 -s 300  
...  
ec2-514-88-169-195.compute-1.amazonaws.com root      14922  0.0 10.1 267020 61092 ?  
S1  12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c  
/etc/chef/client.rb -P /var/run/chef/client.pid -i 1800 -s 300  
...
```



Its using our full-stack installation path automatically, and the -d says "runs as a daemon in the background"

We log to /var/log/chef/client.log (configurable). The -i is the interval, a number of seconds (30 minutes here) (configurable) and the last bit is splay (mitigating the thundering herd problem)

Instructor Note: embdded dir path--talk about Omnibus installer if not mentioned already.

Converge All Nodes (Windows)



```
$ knife winrm "os/windows" -x USER -P PWD  
-a cloud.public_ipv4 "chef-client"
```

```
54.159.197.193 Starting Chef Client, version 13.6.0  
54.159.197.193  
54.159.197.193 [2017-10-31T03:45:21+00:00] INFO: *** Chef 13.6.0 ***  
54.159.197.193 [2017-10-31T03:45:21+00:00] INFO: Platform: x64-mingw32  
54.159.197.193 [2017-10-31T03:45:21+00:00] INFO: Chef-client pid: 2860  
54.159.197.193 [2017-10-31T03:45:21+00:00] INFO: The plugin path  
C:\chef\ohai\plugins does not exist. Skipping...  
54.159.197.193 [2017-10-31T03:45:25+00:00] INFO: Run List is [role[web_server]]  
54.159.197.193 [2017-10-31T03:45:25+00:00] INFO: Run List expands to [mychef-client,  
company_web]  
54.159.197.193 [2017-10-31T03:45:25+00:00] INFO: Starting Chef Run for iis_web
```



We need to apply this new policy to our Windows node as well.

Verify chef-client is Running (Windows)



```
$ knife winrm 'os:windows' -x USER -P PWD -a  
cloud.public_ipv4 'schtasks /Query | findstr /i "chef-client"'
```

```
54.159.197.193 chef-client  
54.159.197.193 10/23/2017 11:39:00 PM Running
```



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:

- ✓ Upload chef-client cookbook to Chef Server
- ✓ Configure each of our nodes to run chef-client as a service

Instructor Note:

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



Change Default Settings

Wait!

There has just been a mandate that every node in the infrastructure must run chef-client every 5 minutes.

Example: Setting the chef-client run Interval

```
chef-repo/cookbooks/chef-client/attributes/default.rb

...
default['chef_client']['log_file']      = 'client.log'
default['chef_client']['interval']     = '1800'
default['chef_client']['splay']         = '300'
default['chef_client']['conf_dir']      = '/etc/chef'
default['chef_client']['bin']          = '/usr/bin/chef-client'

...
```

We need to change the value of the attribute

`default['chef_client']['interval']` from **1800** (seconds) to
300 (...but don't change it in this file.)

So we need to change the attribute `default['chef_client']['interval']` in `chef-client` cookbook.

But what if a new cookbook version is released and we want to upgrade to it?

We'd need to re-implement these changes!

Maintenance nightmare – especially if its been refactored!

Good practice would be to not edit a community cookbook that you have wrapped.
So we will set the interval attribute as shown on the next slide.

GL: Update the 'base' Role

`chef-repo/roles/base.rb`

```
name 'base'  
description 'Base Role'  
run_list 'recipe[mychef_client]'  
default_attributes 'chef_client' => {'interval' => 300}
```



Editing the base.rb file as so will set the interval node attribute to 300 seconds rather than 1800.

Upload it to the Chef Server



```
$ cd ~/chef-repo/roles  
$ knife role from file base.rb
```

```
Updated Role base!
```



Now you need to upload it to the Chef Server. This is done through the command 'knife role from file base.rb'.

The knife tool understands that you are uploading a role file and will look within the roles folder to find a file named base.rb.

Converge All Nodes (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD "sudo chef-client"
```

```
ec2-54-242-217-190.compute-1.amazonaws.com Starting Chef Client, version 13.2.20
ec2-184-73-96-131.compute-1.amazonaws.com  Starting Chef Client, version 13.2.20
ec2-54-242-217-190.compute-1.amazonaws.com resolving cookbooks for run list:
["mychef-client", "myhaproxy"]
ec2-184-73-96-131.compute-1.amazonaws.com  resolving cookbooks for run list:
["mychef-client", "company_web"]
ec2-184-73-96-131.compute-1.amazonaws.com  Synchronizing Cookbooks:
ec2-184-73-96-131.compute-1.amazonaws.com    - mychef-client (0.1.0)
ec2-184-73-96-131.compute-1.amazonaws.com    - chef-client (9.0.0)
ec2-184-73-96-131.compute-1.amazonaws.com    - cron (4.2.0)
ec2-184-73-96-131.compute-1.amazonaws.com    - compat_resource (12.19.0)
ec2-184-73-96-131.compute-1.amazonaws.com    - logrotate (2.2.0)
ec2-184-73-96-131.compute-1.amazonaws.com    - windows (3.2.0)
```



To verify that everything is working the same as before, run 'knife ssh' for both of these nodes. In this instance the query syntax is going to find all nodes with operating system of Linux.

Verify chef-client is Running (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD \
  "ps awux | grep chef-client"
```

```
ec2-514-88-185-159.compute-1.amazonaws.com root      10369  0.0 10.1 266928 61116 ?
S1  12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 300 -s 300
...
ec2-514-88-169-195.compute-1.amazonaws.com root      14922  0.0 10.1 267020 61092 ?
S1  12:54  0:00 /opt/chefdk/embedded/bin/ruby /usr/bin/chef-client -d -c
/etc/chef/client.rb -P /var/run/chef/client.pid -i 300 -s 300
...
```



Its using our full-stack installation path automatically, and the -d says "runs as a daemon in the background"

We log to /var/log/chef/client.log (configurable). The -i is the interval, a number of seconds (30 minutes here) (configurable) and the last bit is splay (mitigating the thundering herd problem)

Instructor Note: embdded dir path--talk about Omnibus installer if not mentioned already.

Converge All Nodes (Windows)



```
$ knife winrm "os/windows" -x USER -P PWD \
-a cloud.public_ipv4 "chef-client"
```

```
54.159.197.193 Starting Chef Client, version 13.6.0
54.159.197.193
54.159.197.193 [2017-10-31T03:52:58+00:00] INFO: *** Chef 13.6.0 ***
54.159.197.193 [2017-10-31T03:52:58+00:00] INFO: Platform: x64-mingw32
54.159.197.193 [2017-10-31T03:52:58+00:00] INFO: Chef-client pid: 1424
54.159.197.193 [2017-10-31T03:52:58+00:00] INFO: The plugin path
C:\chef\ohai\plugins does not exist. Skipping...
54.159.197.193 [2017-10-31T03:53:02+00:00] INFO: Run List is [role[web_server]]
54.159.197.193 [2017-10-31T03:53:02+00:00] INFO: Run List expands to [mychef-client,
company_web]
54.159.197.193 [2017-10-31T03:53:02+00:00] INFO: Starting Chef Run for iis_web
```



Let's apply this policy to our Windows nodes as well.

Verify chef-client is Running (Windows)



```
$ knife winrm 'os:windows' -x USER -P PWD -a  
cloud.public_ipv4 'schtasks /Query | findstr /i "chef-client"'
```

```
54.159.197.193 chef-client  
54.159.197.193 10/23/2017 11:39:00 PM Ready
```



And check again to make sure that the task is running.



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.



CHEFTM

©2018 Chef Software Inc.

Data Bags

Working with Custom Data Sets

Objectives



After completing this module, you should be able to

- Understand how to use and manage a Data Bag
- Create and upload a Data Bag to Chef Server
- Query Data Bag information with the CLI
- Utilize Data Bag information from within recipes

CONCEPT



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?

CONCEPT



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

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",  
  "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]
```



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
```

```
WARNING: Unencrypted data bag detected, ignoring any provided secret options.  
comment: I am a centos 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 windows user
data_bag: users
id: windows_user
platform: windows

chef_type: data_bag_item
comment: I am a centos user
data_bag: users
id: centos_user
platform: centos
```



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
- Upload 'myusers' cookbook to Chef Server
- Update web_server role
- Converge web nodes

How might we go about creating users on our nodes dynamically using information stored within our 'users' data bag?

Instructor Note:

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

CONCEPT



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
```

```
Compiling Cookbooks...  
Recipe: code_generator::cookbook  
  * directory[C:/Users/USER/chef-repo/cookbooks/myusers] action create  
    - create new directory C:/Users/USER/chef-repo/cookbooks/myusers  
  * template[C:/Users/USER/chef-repo/cookbooks/myusers/metadata.rb] action  
create_if_missing  
    - create new file C:/Users/USER/chef-repo/cookbooks/myusers/metadata.rb  
    - update content in file C:/Users/USER/chef-repo/cookbooks/myusers/metadata.rb from  
none to 899276  
      (diff output suppressed by config)  
  * template[C:/Users/USER/chef-repo/cookbooks/myusers/README.md] action  
create_if_missing
```



Go ahead and 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 default recipe
- Upload 'myusers' cookbook to Chef Server
- Update web_server role
- 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 default recipe
- Upload 'myusers' cookbook to Chef Server
- Update web_server role
- Converge web nodes

GL: Change to the cookbooks/myusers Directory



```
$ cd cookbooks/myusers
```



Navigate to the 'myusers' directory for us to upload with berkshelf.

GL: Run berks install



```
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'myusers' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using myusers (0.1.0) from source at .
```



Running the 'berks install' will resolve any dependencies of which this cookbook has none.

GL: Upload the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded myusers (0.1.0) to: 'https://api.opscode.com:443/organizations/ORG_NAME'
```



A 'berks upload' will get our new cookbook up to the Chef Server.

GL: Display Cookbooks within your Org.



```
$ knife cookbook list
```

```
apache          0.1.0
build-essential 7.0.2
compat_resource 12.16.2
cpu             1.0.0
haproxy         2.0.0
mingw           1.2.4
myhaproxy      0.1.0
myusers         0.1.0
ohai            4.2.3
seven_zip       2.0.2
windows          2.1.1
workstation     0.2.1
```



And we verify that the cookbook is now found on the Chef Server.



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
- ✓ Upload 'myusers' cookbook to Chef Server
- Update web_server role
- Converge web nodes

GL: Update the Web Server Role to include myusers

~/chef-repo/roles/web_server.rb

```
name 'web_server'  
description 'Apache and IIS Web Servers'  
run_list 'role[base]', 'recipe[myusers]', 'recipe[company_web]'  
default_attributes 'company_web' => { 'company_name' => 'E Corp' }
```



We will want to add the 'myusers' default recipe to the run list of our web servers.

GL: Upload Role to the Chef Server



```
$ knife role from file web_server.rb
```

```
Updated Role web_server
```



And we update the `web_server` role.

GL: View Details of the Role



```
$ knife role show web_server
```

```
chef_type:          role
default_attributes:
  company_web:
    company_name: E Corp
description:        Apache and IIS Web Servers
env_run_lists:
json_class:         Chef::Role
name:               web_server
override_attributes:
run_list:
  role[base]
  recipe[myusers]
  recipe[company_web]
```



Verifying that the `web_server` role now has the '`myusers`' cookbook in the run list, we are now ready to converge our web servers.



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
- ✓ Upload 'myusers' cookbook to Chef Server
- ✓ Update web_server role
- ❑ Converge web nodes

GL: Converging all Web Nodes (Linux)



```
$ knife ssh "role:web_server AND os:linux" -x USER -P PWD "sudo chef-client"
```

```
ec2-52-90-49-196.compute-1.amazonaws.com Starting Chef Client, version  
13.2.20  
ec2-52-90-49-196.compute-1.amazonaws.com resolving cookbooks for run list:  
["myusers", "apache"]  
ec2-52-90-49-196.compute-1.amazonaws.com Synchronizing Cookbooks:  
ec2-52-90-49-196.compute-1.amazonaws.com   - myusers (0.1.0)  
ec2-52-90-49-196.compute-1.amazonaws.com   - apache (0.2.1)  
ec2-52-90-49-196.compute-1.amazonaws.com Installing Cookbook Gems:  
ec2-52-90-49-196.compute-1.amazonaws.com Compiling Cookbooks...  
ec2-52-90-49-196.compute-1.amazonaws.com Converging 4 resources  
ec2-52-90-49-196.compute-1.amazonaws.com Recipe: myusers::default  
...
```



GL: Converge All Web Nodes (Windows)



```
> knife winrm "role:web_server AND os:windows" -a cloud.public_ipv4 -x USER  
-P PWD "chef-client"
```

```
34.196.63.231 [2017-01-20T14:34:57+00:00] INFO: *** Chef 12.13.37 ***  
34.196.63.231 [2017-01-20T14:34:57+00:00] INFO: Platform: i386-mingw32  
34.196.63.231 [2017-01-20T14:34:57+00:00] INFO: Chef-client pid: 2460  
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Run List is [role[web]]  
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Run List expands to [myusers,myiis]  
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Starting Chef Run for node1  
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Running start handlers  
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Start handlers complete.  
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Loading cookbooks [myusers@0.1.0,  
myiis@0.2.1]  
52.71.198.206 Synchronizing Cookbooks:  
52.71.198.206   - myusers (0.1.0)  
...
```



GL: Check local users for Apache Server



```
$ knife ssh "name:apache_web" -x USER -P PWD "cat /etc/passwd"
```

```
....  
ec2-34-226-207-222.compute-1.amazonaws.com sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin  
ec2-34-226-207-222.compute-1.amazonaws.com chef:x:500:500:ChefDK User:/home/chef:/bin/bash  
ec2-34-226-207-222.compute-1.amazonaws.com dbus:x:81:81:System message bus:/sbin/nologin  
ec2-34-226-207-222.compute-1.amazonaws.com dockerroot:x:498:498:Docker User:/var/lib/docker:/sbin/nologin  
ec2-34-226-207-222.compute-1.amazonaws.com apache:x:48:48:Apache:/var/www:/sbin/nologin  
ec2-34-226-207-222.compute-1.amazonaws.com centos_user:x:501:501:I am a centos 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"
```

| | |
|----------------------------------|----------------------|
| 52.90.49.196 User name | windows_user |
| 52.90.49.196 Full Name | I am a windows user |
| 52.90.49.196 Comment | |
| 52.90.49.196 User's comment | |
| 52.90.49.196 Country/region code | 000 (System Default) |
| 52.90.49.196 Account active | Yes |
| 52.90.49.196 Account expires | Never |
| | |



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 default recipe
- ✓ Upload 'myusers' cookbook to Chef Server
- ✓ Update web_server role
- ✓ 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>



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 and upload cookbook
- Converge your web_server 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.

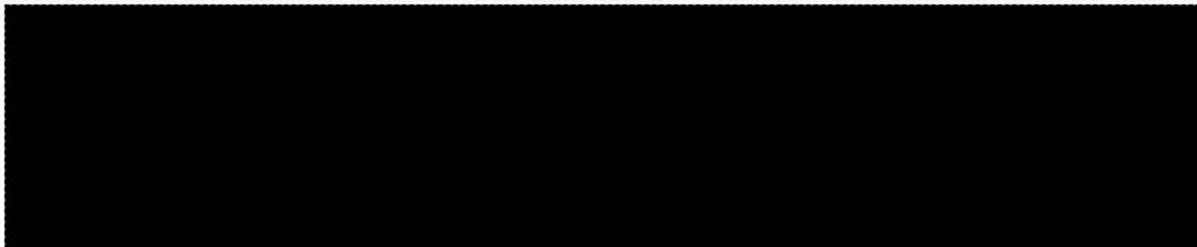
Instructor Note:

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

Lab: Create a data_bags/groups Directory



```
$ 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.



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 and upload cookbook
- ❑ Converge your web_server 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'

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.



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 and upload cookbook
- ❑ Converge your web_server 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 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: 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 and upload cookbook
- Converge your web_server 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: 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'
```

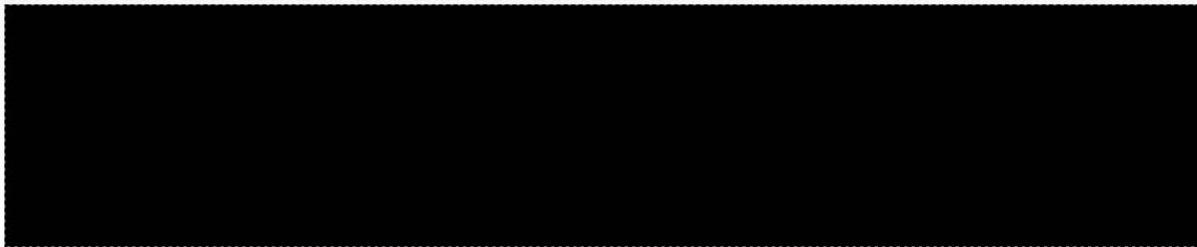


This is considered a minor change to the cookbook and is reflected in the metadata.rb file.

Lab: Change to the cookbooks/myusers Directory



```
$ cd cookbooks/myusers
```



Jump into the 'myusers' cookbook so that we are able to upload.

Lab: Upload the Cookbook to the Chef Server



```
$ berks upload
```

```
Uploaded myusers (0.2.0) to: 'https://api.opscode.com:443/organizations/ORG_NAME'
```



The new 0.2.0 version of 'myusers' is uploaded to the Chef Server.

Lab: Display Cookbooks within your Org.



```
$ knife cookbook list
```

```
apache          0.2.1
build-essential 7.0.2
compat_resource 12.16.2
cpu             1.0.0
haproxy         2.0.0
mingw           1.2.4
myhaproxy      0.1.0
myusers        0.2.0
ohai            4.2.3
seven_zip       2.0.2
windows         2.1.1
workstation     0.2.1
```



And we verify this with a 'knife cookbook list'.



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 and upload cookbook
- Converge your web_server 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: Converging all Web Nodes (Linux)



```
$ knife ssh "role:web_server AND os:linux" -x USER -P PWD "sudo chef-client"
```

```
ec2-52-90-49-196.compute-1.amazonaws.com Starting Chef Client, version  
13.2.20  
ec2-52-90-49-196.compute-1.amazonaws.com resolving cookbooks for run list:  
["myusers", "apache"]  
ec2-52-90-49-196.compute-1.amazonaws.com Synchronizing Cookbooks:  
ec2-52-90-49-196.compute-1.amazonaws.com   - myusers (0.1.0)  
ec2-52-90-49-196.compute-1.amazonaws.com   - apache (0.2.1)  
ec2-52-90-49-196.compute-1.amazonaws.com Installing Cookbook Gems:  
ec2-52-90-49-196.compute-1.amazonaws.com Compiling Cookbooks...  
ec2-52-90-49-196.compute-1.amazonaws.com Converging 4 resources  
ec2-52-90-49-196.compute-1.amazonaws.com Recipe: myusers::default  
...
```



Now that our new version of our cookbook is now uploaded, let's converge our nodes.

Lab: Converge all Web Nodes (Windows)



```
> knife winrm "role:web_server AND os:windows" -a cloud.public_ipv4 -x USER  
-P PWD "chef-client"

34.196.63.231 [2017-01-20T14:34:57+00:00] INFO: *** Chef 12.13.37 ***
34.196.63.231 [2017-01-20T14:34:57+00:00] INFO: Platform: i386-mingw32
34.196.63.231 [2017-01-20T14:34:57+00:00] INFO: Chef-client pid: 2460
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Run List is [role[web]]
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Run List expands to [myusers,myiis]
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Starting Chef Run for node1
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Running start handlers
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Start handlers complete.
34.196.63.231 [2017-01-20T14:35:34+00:00] INFO: Loading cookbooks [myusers@0.1.0,
myiis@0.2.1]
52.71.198.206 Synchronizing Cookbooks:
52.71.198.206   - myusers (0.1.0)
...
...
```





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 and upload cookbook
- ✓ Converge your web_server 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: 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 cgred: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
```



Verify that the new group was created by looking at the 'group' file.



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 and upload cookbook
- ✓ Converge your web_server 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: 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.
```



On our Windows machine we verify the new group was created with this command.



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 and upload cookbook
- ✓ Converge your web_server nodes
- ✓ Verify the new group on apache_web with: `cat /etc/group`
- ✓ Verify the new group on iis_web with: `net localgroup GROUP_NAME`

Instructor Note:

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



Discussion

When should we utilize data bags instead of node attributes?

When creating a new data bag, what index on the Chef server does the data bag get added to?



Q&A

What questions can we help you answer?



CHEFTM

©2018 Chef Software Inc.

Environments

Using Environments to Reflect Organization Patterns and Workflow

Objectives

After completing this module, you should be able to

- Create a production and acceptance environment
- Deploy a node to an environment
- Update a search query to be more exact

In this section, you will learn how to create an environment, deploy a node to an environment, and update a search query to be more exact.

Keeping Your Infrastructure Current

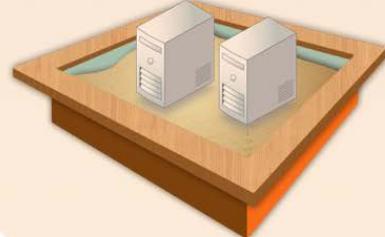
Changing Needs
Changing Software
Growing Organization
Increased Website Popularity



Production



Acceptance



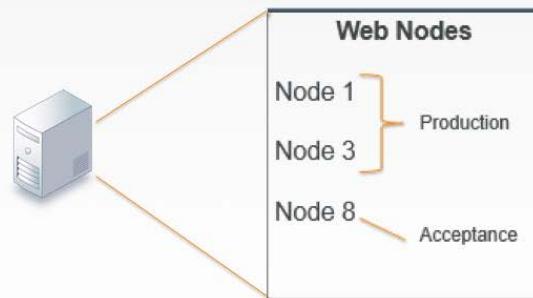
So, we have updated our load balancer's myhaproxy cookbook to dynamically search for and update nodes. Everything is as it should be. But our system is like a living, breathing thing that must grow and be updated to fit our changing needs. We need to find a way to update and test new tools, features 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 environments are useful.

Environments

Environments can define different functions of nodes that live on the same system.

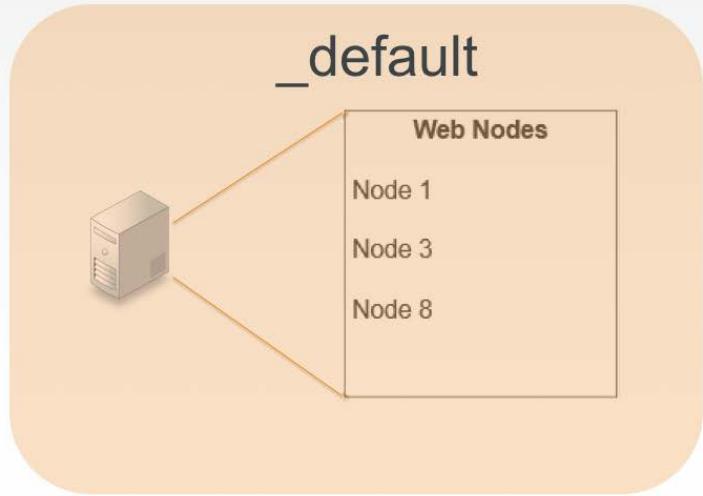


You likely are familiar with the concept of environments. An environment can best be defined as a logical separation of nodes that most often describe the life-cycle of an application. Each environment signifies different behaviors and policies to which a node adheres for a given application or platform.

For example, environments can be separated into 'acceptance' and 'production'. "Acceptance" would be where we may make allowances for constant change and updates and for applications to be deployed with each release. "Production" might be where we lock down our infrastructure and policies. Production would be what the outside world sees, and would remain unaffected by changes and upgrades until you specifically release them. Chef also has a concept of an environment. A Chef environment allows us to define a list of policies that we will allow by defining a cookbook.

Environments

Every organization or infrastructure starts with the `_default` environment.



Chef also has a concept of an environment. Chef uses environments to map an organization's real-life workflow to what can be configured and managed using the Chef server.

Every organization begins with a single environment called the `_default` (underscore default) environment, which cannot be modified or deleted.

Therefore, you must create custom environments to define your organization's workflow.



Group Lab: Production

Let's create a reliable environment for our nodes.

Objective:

- Create a Production Environment
- Add the IIS web node to Production
- Add the load balancer node to Production

GL: Viewing the Help of the environment Subcommand



```
$ cd ~/chef-repo  
$ knife environment --help  
  
** ENVIRONMENT COMMANDS **  
  
knife environment compare [ENVIRONMENT...] (options)  
knife environment create ENVIRONMENT (options)  
knife environment delete ENVIRONMENT (options)  
knife environment edit ENVIRONMENT (options)  
knife environment from file FILE [FILE...] (options)  
knife environment list (options)  
knife environment show ENVIRONMENT (options)
```



Because we still are communicating with the Chef server, let's ask Chef for help regarding available environment commands.

So change into chef-repo and then run 'knife environment --help'.

GL: Viewing the List of Environments GL:



```
$ knife environment list
```

```
_default
```



Remember, we use 'list' to view existing environments.

As previously stated, we see the _default environment has already been created.

GL: Viewing the Details of an Environment



```
$ knife environment show _default

chef_type:           environment
cookbook_versions:
default_attributes:
description:        The default Chef environment
json_class:         Chef::Environment
name:               _default
override_attributes:
```



Let's see how this environment looks.

GL: Creating an Environments Directory



```
$ mkdir environments
```



GL: Defining a Production Environment

```
~/chef-repo/environments/production.rb
```

```
name 'production'
description 'Where we run production code'

cookbook 'company_web', '= 0.1.0'
cookbook 'myhaproxy', '= 1.0.0'
```



Then we need to create a production.rb file. Like in the roles.rb files, we must provide a name and description. Additionally, we need to define cookbook restrictions to lock down specific versions of both the apache and myhaproxy cookbooks.

By adding this information to production.rb, we are telling our nodes to use these specific versions of these specific cookbooks. Obviously, what this means is that as we work on newer versions of these cookbooks, we won't break anything in the production environment. Okay, so now that we have captured our 'good' environment in this file, let's save it and upload it.

GL: Uploading the Production Environment



```
$ knife environment from file production.rb
```



Using the knife environment command, let's upload the production.rb file. This should be familiar because it is just like the command we used to upload roles.

GL: Viewing the List of Environments



```
$ knife environment list
```

```
_default
```

```
production
```



Okay, let's use our list command to make sure the file uploaded correctly.

GL: Viewing the Details of an Environment



```
$ knife environment show production

chef_type:           environment
cookbook_versions:
  company_web: = 0.1.0
  myhaproxy:   = 1.0.0
default_attributes:
description:        Where we run production code
json_class:         Chef::Environment
name:               production
override_attributes:
```



If we use the knife environment show command, we can see how the production.rb file looks.

Note the cookbook versions that we set are shown here.



Group Lab: Production

Let's create a reliable environment for our nodes.

Objective:

- Create a Production Environment
- Add the IIS web node to Production
- Add the load balancer node to Production

Now that we have the environment defined it is time to move one of our web nodes into that environment.

GL: Searching for All Nodes



```
$ knife search node "*:*"
```

```
3 items found
```

```
Node Name: lb
```

```
Environment: _default
```

```
FQDN: ip-172-31-23-107.ec2.internal
```

```
IP: 34.226.220.213
```

```
Run List: recipe[myhaproxy]
```

```
Roles:
```

```
Recipes: myhaproxy, myhaproxy::default, haproxy::manual,  
haproxy::install_package
```

```
Platform: centos 6.9
```

```
Tags:
```



If we search our nodes, we see that all three nodes have been set to the `_default` environment. How do we change this?

GL: Viewing the Help of the node Subcommand



```
$ 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)
```



Now, we need to set the environments for our nodes. Let's ask Chef for help on that as well.

Let's use the knife node environment set command.

GL: Viewing the Help of the environment Subcommand

```
 $ knife node environment set --help  
  
knife node environment set NODE ENVIRONMENT  
  -s, --server-url URL          Chef Server URL  
  --chef-zero-host HOST         Host to start chef-zero on  
  --chef-zero-port PORT        Port (or port range) to start chef-zero on.  
Port ranges like 1000,1010 or 8889-9999 will try all given ports until one works.  
  -k, --key KEY                API Client Key  
  --[no-]color                 Use colored output, defaults to false on  
Windows, true otherwise  
  -c, --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  
  -e, --editor EDITOR          Set the editor to use for interactive commands
```



But how does that command work, exactly?

It looks like we just add the environment name at the end of the command to set that environment on a node.

GL: Setting the Node's Environment to Production



```
$ knife node environment set iis_web production
```

```
iis_web:  
  chef_environment: production
```



So, let's do that for `iis_web`.

The results don't really tell us much, so let's take a look at `iis_web`.

GL: Viewing the Details about the Node



```
$ knife node show iis_web
```

```
Node Name: iis_web
Environment: production
FQDN: WIN-8694LT97S51.ec2.internal
IP: 34.229.225.40
Run List: role[web_server]
Roles:
Recipes: company_web, company_web::default, myiis::default,
myiis::server
Platform: windows 6.3.9600
Tags:
```



Using knife node show, we can see iis_web's attributes. Note that it has indeed been set to the production environment.



GL: Production

Let's create a reliable environment for our nodes.

Objective:

- Create a Production Environment
- Add the IIS web node to Production
- Add the load balancer node to Production

GL: Setting the Node's Environment to Production



```
$ knife node environment set lb production
```

```
lb:  
  chef_environment: production
```



Let's also add our load_balancer to the production environment.

GL: Viewing the Details about the Node



```
$ knife node show lb
```

```
Node Name: lb
Environment: production
FQDN: ip-172-31-23-107.ec2.internal
IP: 34.226.220.213
Run List: recipe[myhaproxy]
Roles:
Recipes: myhaproxy, myhaproxy::default, haproxy::manual,
haproxy::install_package
Platform: centos 6.9
Tags:
```



And, it looks like lb was successfully set to the production environment.



Group Lab: Production

Let's create a reliable environment for our nodes.

Objective:

- ✓ Create a Production Environment
- ✓ Add the IIS web node to Production
- ✓ Add the load balancer node to Production



Lab: The Acceptance Environment

- Create an environment named "acceptance" that has no cookbook restrictions
- Move apache_web into the acceptance environment
- Run chef-client on all the nodes

In this lab I would like for you to create an acceptance environment that has no restrictions at all for the policy applied. Place the apache_web node within this new environment and run chef-client on all your nodes.

Lab: Defining the Acceptance Environment

```
~/chef-repo/environments/acceptance.rb
```

```
name 'acceptance'  
description 'Where code and apps are tested'  
# No Cookbook Restrictions
```



First, let's create a new rb file in our chef-repo/environments directory. Let's name it acceptance.

In the Acceptance environment, we don't want to lock-down the cookbook versions, so we are not going to place restrictions on the cookbooks.

Lab: Uploading the Environment



```
$ knife environment from file acceptance.rb
```

```
Updated Environment acceptance
```



Let's upload that .rb file to the Chef server.

Lab: Viewing the List of Environments



```
$ knife environment list
```

```
_default  
production  
acceptance
```



And let's make sure that this environment file was added properly.

Lab: Viewing the Details of the Environment



```
$ knife environment show acceptance
```

```
chef_type:           environment
cookbook_versions:
default_attributes:
description:        Where code and applications are tested
json_class:         Chef::Environment
name:               acceptance
override_attributes:
```



And last, but not least, let's ask the Chef Server to show us the acceptance environment.



Lab: The Acceptance Environment

- ✓ Create an environment named "acceptance" that has no cookbook restrictions
- ❑ Move apache_web into the acceptance environment
- ❑ Run chef-client on all the nodes

Lab: Setting the Node to Acceptance Environment



```
$ knife node environment set apache_web acceptance
```

```
apache_web:  
  chef_environment: acceptance
```



Okay, let's set apache_web to the acceptance environment.

Lab: Viewing Details about the Node



```
$ knife node show apache_web
```

```
Node Name: apache_web
Environment: acceptance
FQDN: ip-172-31-29-209.ec2.internal
IP: 34.207.100.168
Run List: role[web_server]
Roles:
Recipes: company_web, company_web::default, apache::default,
apache::server
Platform: centos 6.9
Tags:
```



And confirm that it has been set properly.



Lab: The Acceptance Environment

- ✓ Create an environment named "acceptance" that has no cookbook restrictions
- ✓ Move apache_web into the acceptance environment
- ❑ Run chef-client on all the nodes

Lab: Converging All the Nodes (Linux)



```
$ knife ssh "os:linux" -x USER -P PWD "sudo chef-client"
```

```
ec2-34-226-220-213.compute-1.amazonaws.com Starting Chef Client, version  
13.2.20  
ec2-34-207-100-168.compute-1.amazonaws.com Starting Chef Client, version  
13.2.20  
ec2-34-226-220-213.compute-1.amazonaws.com resolving cookbooks for run list:  
["myhaproxy"]  
ec2-34-207-100-168.compute-1.amazonaws.com resolving cookbooks for run list:  
["company_web"]  
ec2-34-207-100-168.compute-1.amazonaws.com Synchronizing Cookbooks:  
ec2-34-207-100-168.compute-1.amazonaws.com   - myiis (0.2.1)  
ec2-34-207-100-168.compute-1.amazonaws.com   - company_web (0.1.1)  
ec2-34-207-100-168.compute-1.amazonaws.com   - apache (0.1.0)  
ec2-34-207-100-168.compute-1.amazonaws.com Installing Cookbook Gems:
```



Using the knife ssh let's run chef client on all the Linux nodes.

Lab: Converging All the Nodes (Windows)



```
$ knife winrm "os/windows" -a cloud.public_ipv4 -x  
USER -P PWD "chef-client"
```

```
ec2-54-175-46-24.compute-1.amazonaws.com Starting Chef Client, version 12.3.0  
ec2-54-210-86-164.compute-1.amazonaws.com Starting Chef Client, version 12.3.0  
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0  
ec2-54-210-86-164.compute-1.amazonaws.com resolving cookbooks for run list: ["apache"]  
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:  
["myhaproxy"]  
ec2-54-210-86-164.compute-1.amazonaws.com Synchronizing Cookbooks:  
ec2-54-210-86-164.compute-1.amazonaws.com - apache  
ec2-54-210-86-164.compute-1.amazonaws.com Compiling Cookbooks...  
ec2-54-210-86-164.compute-1.amazonaws.com Converging 3 resources  
ec2-54-210-86-164.compute-1.amazonaws.com Recipe: apache::server  
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:  
ec2-54-210-192-12.compute-1.amazonaws.com - build-essential
```



Using knife winrm let's run chef client on all the windows nodes.



Lab: The Acceptance Environment

- ✓ Create an environment named "acceptance" that has no cookbook restrictions
- ✓ Move apache_web into the acceptance environment
- ✓ Run chef-client on all the nodes



GL: Separating Search

The search criteria that we defined in the Load Balancer does not consider where the nodes are located. That needs to change.

Objective:

- Update the load balancer's search criteria to respect environments
- Update the load balancer cookbook's version number and upload it
- Update the Production environment to allow new cookbook version
- Converge the load balancer node

Now that we have created our two environments and set each node to a specific environment, we need to update the search that we use to find the web nodes to consider the environment to ensure that the load balancer only communicates with the nodes that share its environment.

Instructor Note:

The GitHub repo for the 2.0.0 version of the 'myhaproxy' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-myhaproxy.git>

GL: Viewing the Existing Search Criteria

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

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

all_web_nodes = search('node','role:web_server')

members = []

#...
```



Looking at the existing recipe in the load balancer's myhaproxy cookbook, we can review the original search syntax. If we want to search by environments, what would we need to add here?

GL: Updating the Search to Consider Environment

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

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

all_web_nodes = search('node',"role:web_server AND chef_environment:#{node.chef_environment}")

members = []

#...
```



Search the Chef Server for all node objects that have the role equal to 'web_server' and also share the same environment as the current node applying this recipe. The nodes currently applying this recipe are the nodes with the role set to load_balancer.

Now that we've made our changes, let's save this file.



GL: Separating Search

The search criteria that we defined in the Load Balancer does not consider where the nodes are located. That needs to change.

Objective:

- Update the load balancer's search criteria to respect environments
- Update the load balancer cookbook's version number and upload it
- Update the Production environment to allow new cookbook version
- Converge the load balancer node

GL: Updating the Version of the Cookbook

```
~/chef-repo/cookbooks/myhaproxy/metadata.rb

name          'myhaproxy'
maintainer    'The Authors'
maintainer_email 'you@example.com'
license        'all_rights'
description    'Installs/Configures myhaproxy'
long_description 'Installs/Configures myhaproxy'
version        '2.0.0'

depends 'haproxy', '~> 3.0.0'
```



Before we upload the new myhaproxy cookbook to the server, we probably want to update the version number. What type of change have we made here?

Answer: Major

Because we are performing a major change, let's set the version number to 2.0.0

GL: Uploading the Cookbook to Chef Server



```
$ berks upload
```

```
Uploaded myhaproxy (2.0.0) to: 'https://api.chef.io:443/organizations/2017_oct_26'  
Skipping ohai (5.2.0) (frozen)  
Skipping poise (2.8.1) (frozen)  
Skipping poise-service (1.5.2) (frozen)  
Skipping seven_zip (2.0.2) (frozen)  
Skipping windows (3.2.0) (frozen)
```



And finally berks upload.



GL: Separating Search

The search criteria that we defined in the Load Balancer does not consider where the nodes are located. That needs to change.

Objective:

- ✓ Update the load balancer's search criteria to respect environments
- ✓ Update the load balancer cookbook's version number and upload it
- ❑ Update the Production environment to allow new cookbook version
- ❑ Converge the load balancer node



A Brief Recap

We restricted the production environment to a specific cookbook versions and we then updated the myhaproxy cookbook.

What must we do next for this change to be applied to our load balancer node?

Before we run 'chef-client' to bring everything up to date, let's think about what we've done. First, in the production environment, we restricted our cookbooks to a specific version. Second, we created an acceptance environment with no cookbook restrictions. Third, we set specific nodes to each of these environments. Fourth, we updated the myhaproxy default.rb to include environment search criteria. And lastly, we changed the version number in the myhaproxy metadata.rb file.

Because we have updated the version number we need to update the cookbook version restrictions in the Production environment. Otherwise it will continue to use the previous version of the cookbook.

GL: Updating the Version Constraints for the Environment

```
~/chef-repo/environments/production.rb

name 'production'
description 'Where we run production code'

cookbook 'company_web', '= 0.1.0'
cookbook 'myhaproxy', '= 2.0.0'
```



So let's go back into our production.rb and update it to include the new version number.

GL: Uploading the Environment

```
└─$ cd ~/chef-repo  
└─$ knife environment from file production.rb
```

```
Updated Environment production
```



Change to ~/chef-repo and then run 'knife environment from file production.rb'.

GL: Verifying Version Number for production



```
$ knife environment show production
```

```
chef_type:           environment
cookbook_versions:
  company_web: = 0.1.0
  myhaproxy:   = 2.0.0
default_attributes:
description:        Where we run production code
json_class:         Chef::Environment
name:               production
override_attributes:
```



And let's make sure that the production.rb on Chef server has the correct version of myhaproxy designated.



GL: Separating Search

The search criteria that we defined in the Load Balancer does not consider where the nodes are located. That needs to change.

Objective:

- ✓ Update the load balancer's search criteria to respect environments
- ✓ Update the load balancer cookbook's version number and upload it
- ✓ Update the Production environment to allow new cookbook version
- ❑ Converge the load balancer node

GL: Converging the Load Balancer



```
$ knife ssh "role:load_balancer" -x USER -P PWD "sudo chef-client"

ec2-34-226-220-213.compute-1.amazonaws.com Starting Chef Client, version
13.2.20

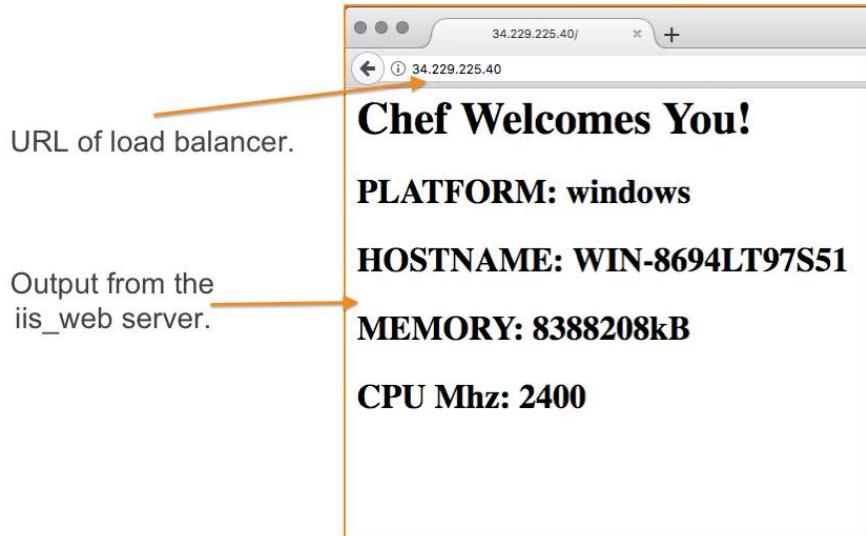
ec2-34-226-220-213.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]

ec2-34-226-220-213.compute-1.amazonaws.com Synchronizing Cookbooks:
  - haproxy (3.0.4)
  - cpu (2.0.0)
  - build-essential (8.0.3)
  - seven_zip (2.0.2)
  - windows (3.2.0)
  - ohai (5.2.0)
  - myhaproxy (2.0.0)
```



And use 'sudo chef-client' to converge the load balancer node.

GL: Only the iis_web Website is Being proxied



Point a web browser to the URL or public IP address of your load balancer. It should display the web page of the iis_web server node that the load balancer is configured to serve.



GL: Separating Search

The search criteria that we defined in the Load Balancer does not consider where the nodes are located. That needs to change.

Objective:

- ✓ Update the load balancer's search criteria to respect environments
- ✓ Update the load balancer cookbook's version number and upload it
- ✓ Update the Production environment to allow new cookbook version
- ✓ Converge the load balancer node

Instructor Note:

The GitHub repo for the 2.0.0 version of the 'myhaproxy' cookbook can be found at:
<https://github.com/chef-training/devops-cookbooks-myhaproxy.git>



Discussion

What is the benefit of constraining cookbooks to a particular environment?

What are the benefits of **not** constraining cookbooks to a particular environment?

Answer these questions.

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

Q&A

What questions can we help you answer?





CHEFTM

©2018 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



Another great place to practice Chef is our awesome Learn Chef Rally site. These interactive modules provide you with an opportunity to run through exercises similar to those we did in this class, and it is updated when new Chef features are introduced. This is one of the most robust self-guided tutorial sites out there.



Beyond Essentials

- What happens during a knife bootstrap?
- What happens during a chef-client run?
- What is the security model used by chef-client?
- 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.

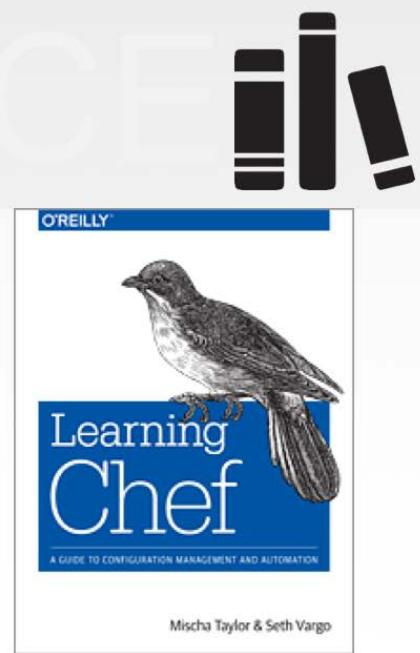
Any question you have, you probably will find the answer for on our Docs site.

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

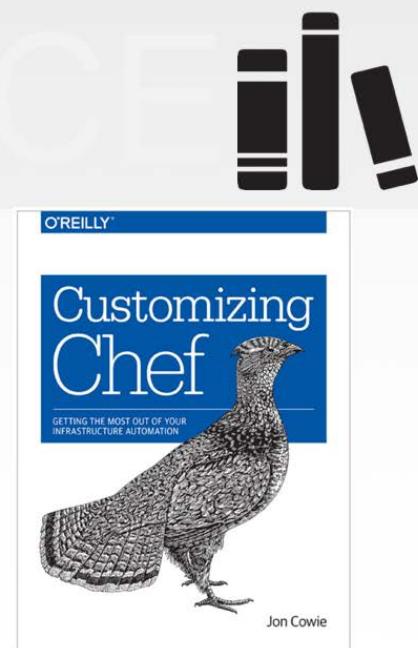
A Guide to Configuration Management and Automation



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



©2018 Chef Software Inc.

20-10

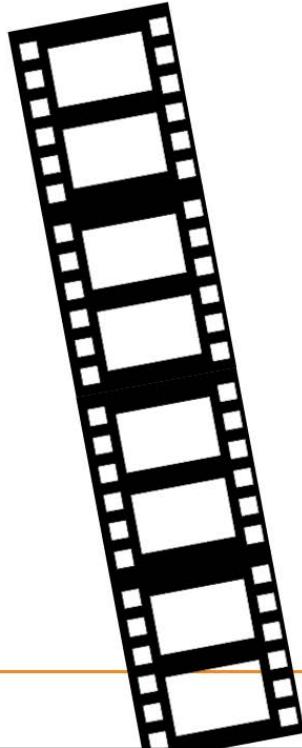


Additionally, you may want to read Jon Cowie's Getting the Most Out of Your Infrastructure Automation. It's also available on O'Reilly.

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!

foodfightshow.org



The Podcast where DevOps chefs do battle

©2018 Chef Software Inc.

19-12



Food Fight is a bi-weekly podcast for the Chef community. We bring together the smartest people in the Chef community and the broader DevOps world to discuss the thorniest issues in system administration.

Chef Developers' 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!

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.



©2018 Chef Software Inc.

<https://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 2018 will be held in Chicago during May.



©2018 Chef Software Inc.

<https://www.chef.io/summit/>



The Chef Community will gather for two days of open space sessions and brainstorm on Chef best practices.

The Chef Community Summit is a facilitated Open Space event. The participants of the summit propose topics, organize an agenda, and discuss and work on the ideas that are most important to the community.



CHEFTM

Thank You!

©2018 Chef Software Inc.

And finally, thank you so much! Hope you enjoyed class the last few days!