Progress®
Chef®

Chef Intermediate

Introduction

Expectations

You will leave this class with the ability to extend the components of Cookbooks.

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

Expectations

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

Break It: If everything works the first time go back and make some changes. Explore! Discovering the boundaries will help you when you continue on your journey.

Discussion and Labs

This course is designed to be hands on. You will run lots of commands, write lots of code, and express your understanding.

- **Discussion:** We will talk about the concepts introduced and the work that we have completed

- **Lab:** You will be asked to perform the task on your own

Day 1

Setting up Test Kitchen

TDD: Writing Tests First

Build Reliable Cookbooks

Agility with Unit Testing & ChefSpec

Refactoring Using Attributes

Refactor Testing to Multiple Platforms

Intro to Ruby

Day 2

Adding Ruby Gems

Creating Custom Resources

Extending Custom Resources

Refining Custom Resources

Testing Custom Resources

Custom Ohai Plugins

Intro to CI/CD Pipeline

Introduce Yourselves

Name

Current job role

Previous job roles / Background

Experience with Chef

Favorite Text Editor

Expectations

Questions?

Q&A

What questions can we answer for you?

Progress®
Chef®

Setting Up Test Kitchen

Objectives

After completing this module, you should be able to:

- Create a Chef Repo
- Configure Test Kitchen
- Use Test Kitchen to create, converge, and verify a recipe

Progress Chef

Concept

Test Driven Development

1. Define a test set for the unit first
2. Then implement the unit
3. Finally verify that the implementation of the unit makes the tests succeed.
4. Refactor

Concept

Building a Web Server

1. Install the httpd/apache2 package
2. Write out a test page
3. Start and enable the httpd/apache2 service

Concept

Defining Scenarios

Given SOME CONDITIONS

When an EVENT OCCURS

Then I should EXPECT THIS RESULT

Concept

Scenario: Potential User Visits Website

Given that **I am a potential user**

When **I visit the company website in my browser**

Then I should **see a welcome message**

Concept

Instructor Demo: Inspec Documentation

Test if a file exists	**Test file does not exist**
describe **file**('/tmp') do it { should exist } end	describe **file**('/tmpest') do it { should_not exist } end

https://docs.chef.io/inspec/resources/

Progress Chef

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 02: Setting Up Test Kitchen

- *Generate a Chef repo*
- *Create a web server cookbook*
- *Configure Test Kitchen*
- *Create a virtual testing environment*

Note: There is no longer a Lab 1 in this course.

Progress®
Chef®

Writing a Test First

Objectives

After completing this module, you should be able to:

- Write an integration test
- Use Test Kitchen to create, converge, and verify a recipe
- Develop a cookbook with a test-driven approach

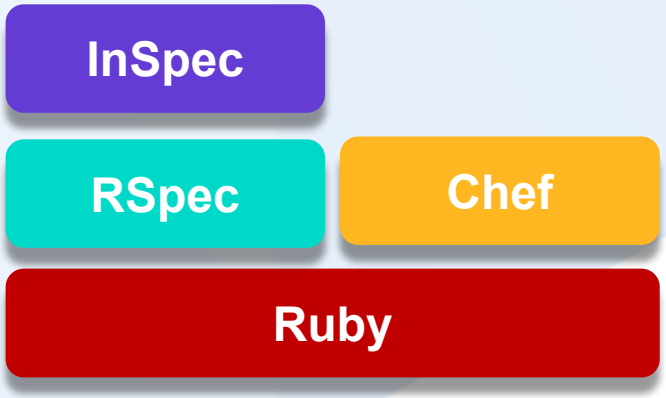
Concept

RSpec and InSpec

RSpec is a Domain Specific Language (DSL) that allows you to express and execute expectations. These expectations are expressed in examples that are asserted in different example groups.

InSpec builds upon RSpec to give you tools that allow you to express expectations about the state of infrastructure.

https://github.com/rspec/rspec



Where do Tests Live?

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

`.../test/integration/default/` is in every cookbook by default and is a pointer (folder) that tests are placed into to be run.

Test Kitchen will look for tests to run under this directory. This corresponds to the value specified in the Test Kitchen configuration file (kitchen.yml) in the suites section.

Where do Tests Live?

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

`default_test.rb` Is the auto-generated file that provides an example. More examples are on Inspec.io.

The **`default_test.rb`** file is a Ruby file that contains the tests that we want to run when we spin up a test instance.

The "default" in **`default_test.rb`** comes from the recipe name with which the test file is associated

Kitchen List

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

PLATFORMS x SUITES

Running `kitchen list` will show that matrix.

Concept

Kitchen Create

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

Create one or more instances.

Create CentOS Instance

Kitchen Converge

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

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

Concept

Kitchen Verify

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

Create, converge, and verify one or more instances.

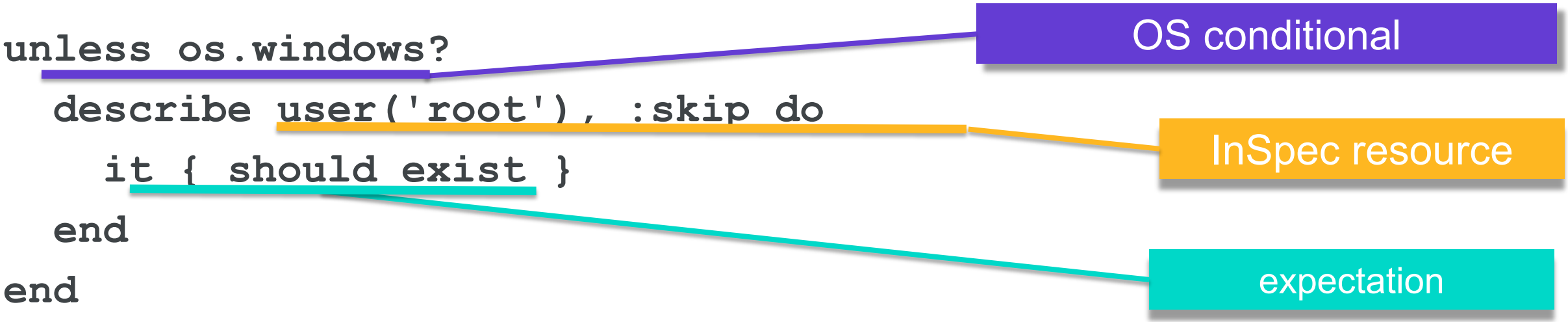
Auto-generated Spec File in Cookbook

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

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

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

Components of an InSpec Example

```
unless os.windows?
  describe user('root'), :skip do
    it { should exist }
  end
end
```



When not on Windows, I expect the user named 'root', to exist.

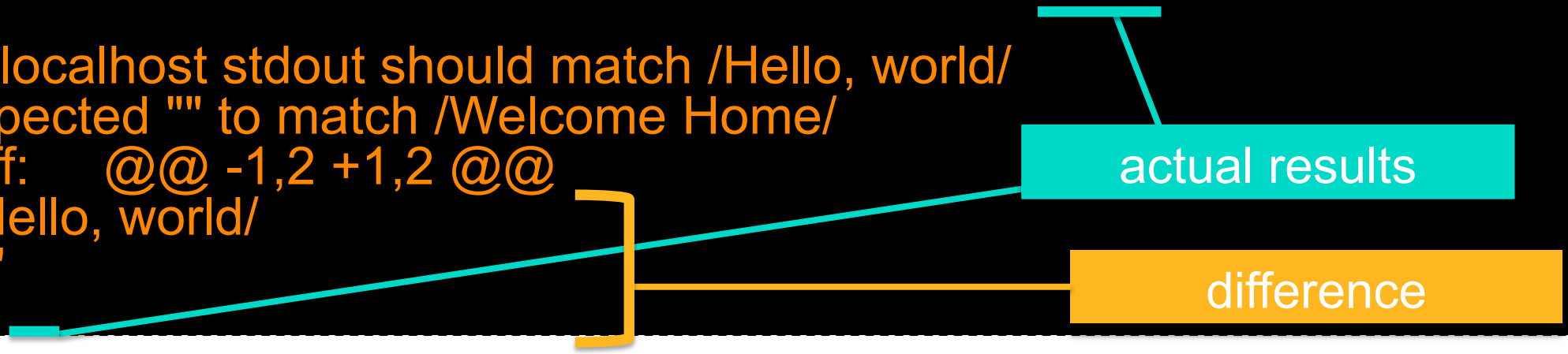
Components of an InSpec Example

```
describe port(80), :skip do
  it { should not be listening }
end
```

When on any platform, I expect the port 80 **not** to be listening for incoming connections.

Examine Failure #1

```
Ø  should be listening
expected `Port 80.listening?` to return true, got false

Ø  localhost stdout should match /Hello, world/
expected "" to match /Welcome Home/
Diff:     @@ -1,2 +1,2 @@
-/Hello, world/
+""
```



Examine the Test Summary

```
Port 80
  Ø should be listening
  expected `Port 80.listening?` to return true, got false
Command curl
  Ø localhost stdout should match /Welcome Home/
  expected "" to match /Welcome Home/
  Diff:     @@ -1,2 +1,2 @@
  -/Hello, world/
  +""
Test Summary 0 successful, 2 failures, 0 skipped
```

A final summary contains the length of execution time with the results shows that RSpec verified 2 examples and found 2 failures.

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 03: Write a test first (TDD)

- *Write two test cases*
- *Test the test cases*

Progress®
Chef®

Building a Reliable Cookbook

Objectives

After completing this module, you should be able to:

- Update the default recipe to include chef resources
- Test the resources' ability to update to the desired state

Progress Chef

Write the Default Recipe for the Cookbook

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

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2018 The Authors, All Rights Reserved.
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome Home!</h1>'
end

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

Questions?

Q&A

What questions can we answer for you?

Progress Chef

Exercise

Lab 04: Build a Reliable Cookbook

- Add resources to the default recipe
- Test the newly added resources

This time it will be different.

Progress Chef

Progress®
Chef®

Faster Feedback with Unit Testing

Objectives

After completing this module, you should be able to:

- Write unit tests with a speedier response time
- Explain the importance and limitations of unit testing
- Verify tests using ChefSpec

Concept

Integration tests utilize Test Kitchen

Integration testing operates at the cookbook or recipe level. This provides more complete and accurate results.

Unit tests utilize ChefSpec

Unit testing operates at the resource level. This provides faster results, but can give an error when the cookbook works correctly.

Concept

Slower Feedback Cycle

The slower the feedback loop the less value it provides to you while developing your cookbooks. You are less inclined to run the test suite. Which means you will likely miss issues as they happen.

Concept

External Dependencies

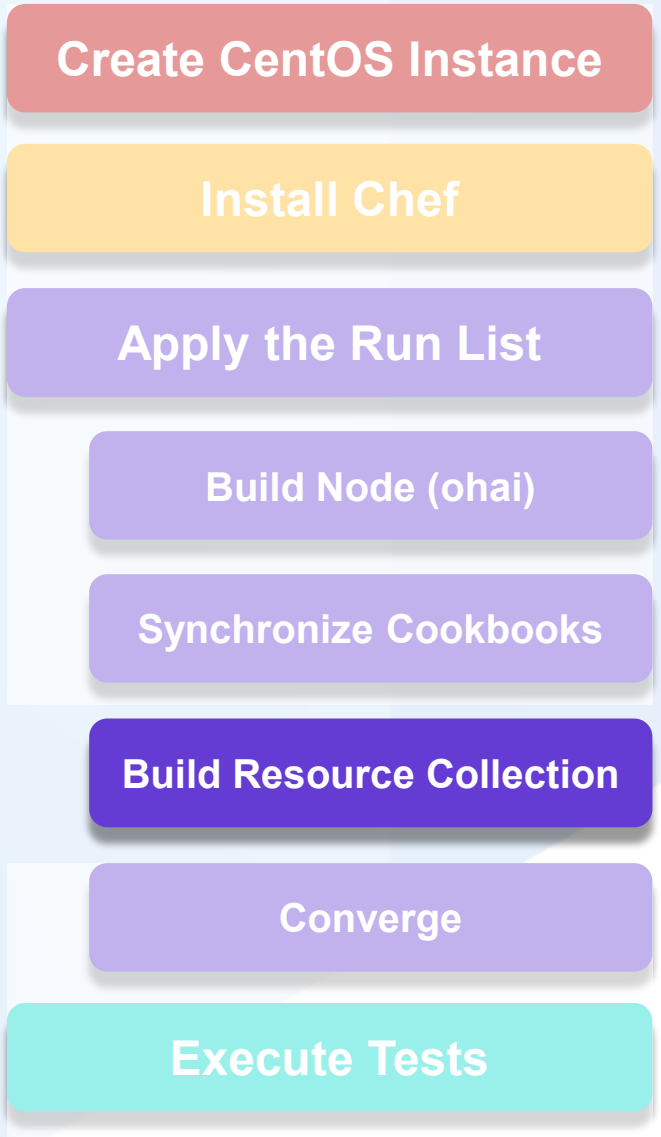
The speed of integration testing is affected by creating the test instance, installing chef, and applying the run list.

- Create CentOS Instance
- Install Chef
- Apply the Run List
- Build Node (ohai)
- Synchronize Cookbooks
- Build Resource Collection
- Converge
- Execute Tests

Concept

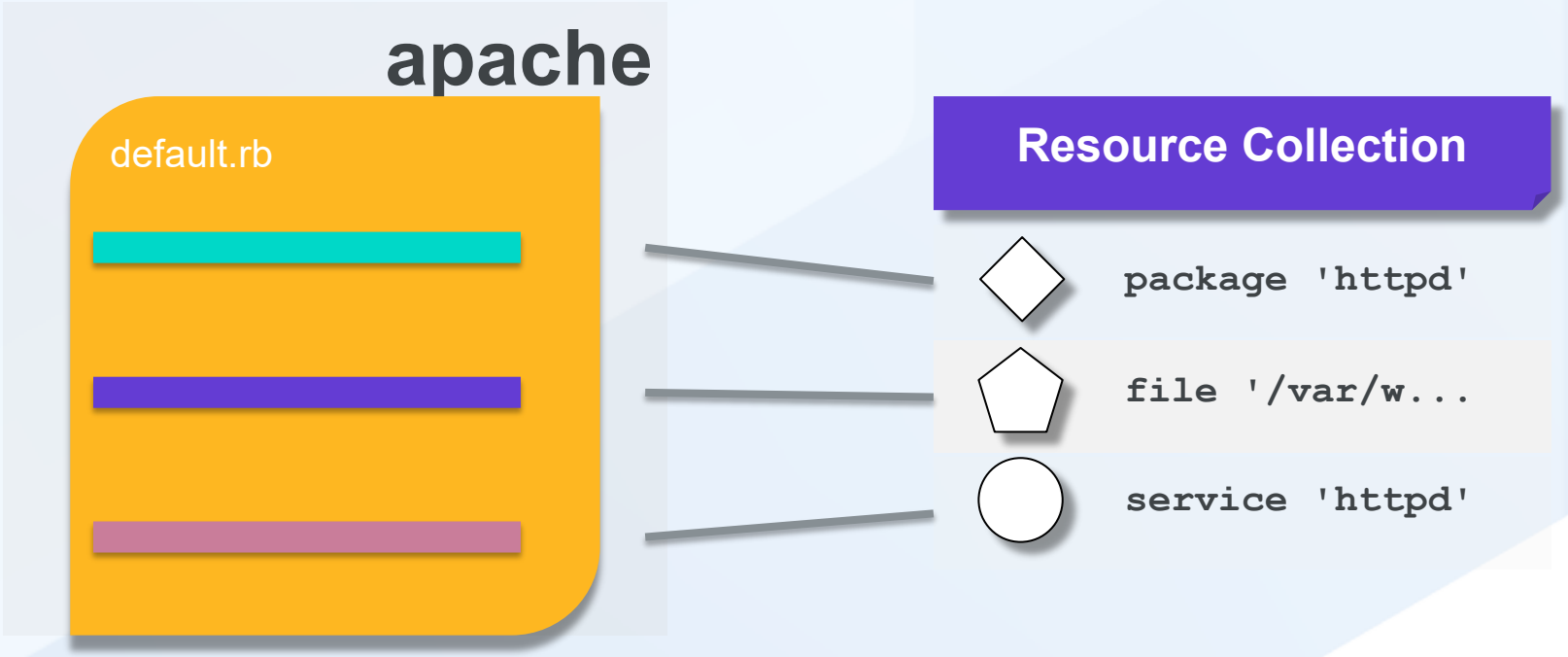
Build Resource Collection

The resource collection is a list of all the resources and recipes loaded across all the recipes within the run list.



Concept

Resource Collection

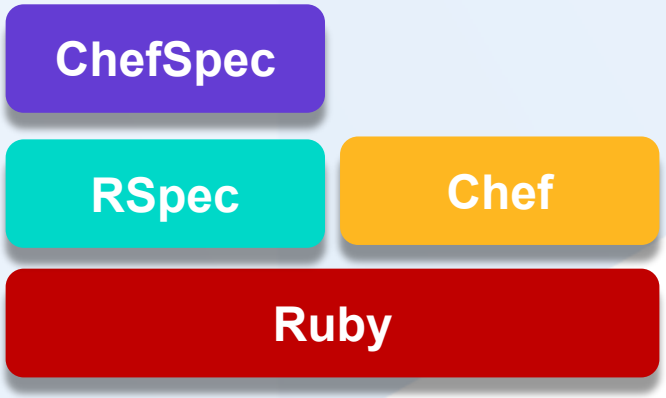


Concept

RSpec and ChefSpec

RSpec is a Domain Specific Language (DSL) that allows you to express and execute expectations. These expectations are expressed in examples that are asserted in different example groups.

ChefSpec provides helpers and tools that allow you to express expectations about the state of **resource collection**.



Test Kitchen versus ChefSpec

ChefSpec

Test Kitchen using InSpec

ChefSpec Documentation

Chef Spec is documented at https://docs.chef.io/workstation/chefspec/

Find within the documentation examples of testing for the `package` resource

There are also examples in the Chef Spec Git repo:

https://github.com/chefspec/chefspec

View the Unit Test for the Default Recipe

~/chef-repo/cookbooks/apache/spec/unit/recipes/default_spec.rb

```
require 'spec_helper'

describe 'apache::default' do
  context 'When all attributes are default, on Ubuntu 20.04' do
    Platform 'ubuntu', '20.04'

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end
  end

  context 'When all attributes are default, on CentOS 8' do
    Platform 'centos', '8'

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end
  end

end
```

These Are the Three Things to Test

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

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright:: 2018, The Authors, All Rights Reserved.
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome Home!</h1>'
end

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

Create a Pending Test

~/chef-repo/cookbooks/apache/spec/unit/recipes/default_spec.rb

```
    # ... START OF THE SPEC FILE ...
    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs the httpd package'

  end
end
```

Write Test Verifying the Package Install

~/chef-repo/cookbooks/apache/spec/unit/recipes/default_spec.rb

```
 # ... START OF THE SPEC FILE ...
    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs the httpd package ' do
      expect(chef_run).to install_package('httpd')
    end
  end
end
```

Test File Resource

~/chef-repo/cookbooks/apache/spec/unit/recipes/default_spec.rb

```
 # ... START OF THE SPEC FILE ...
  it 'creates the index file' do
      expect(chef_run).to
        render_file('/var/www/html/index.html')
      .with_content('<h1>Welcome Home!</h1>')
  end

  it 'starts the httpd service' do
    expect(chef_run).to start_service('httpd')
    expect(chef_run).to enable_service('httpd')
  end
end
```

This is all on one line in your file

Heckling Your Code

Mutation testing is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in small ways.

Comment out Resource

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

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2018 The Authors, All Rights Reserved.

# package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome Home!</h1>'
end

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

Concept

Discussion

What functionality did you test in the integration tests?

What functionality did you test in these unit tests?

What do you see as the scope of unit testing versus integration testing?

What are the differences between a ChefSpec test and a InSpec test?

Questions?

Q&A

What questions can we answer for you?

Progress Chef

Exercise

Lab 5: Agility with Unit Testing & ChefSpec

- Review and run the existing tests
- Identify the tests that we need to write
- Write and execute the tests to identify the failure
- Fix the code and execute the tests to see success

The faster the feedback from our tests, the more likely we are to run them.
The more likely we are to run them means they will catch more issues.

Progress®
Chef®

Refactoring Using Attributes

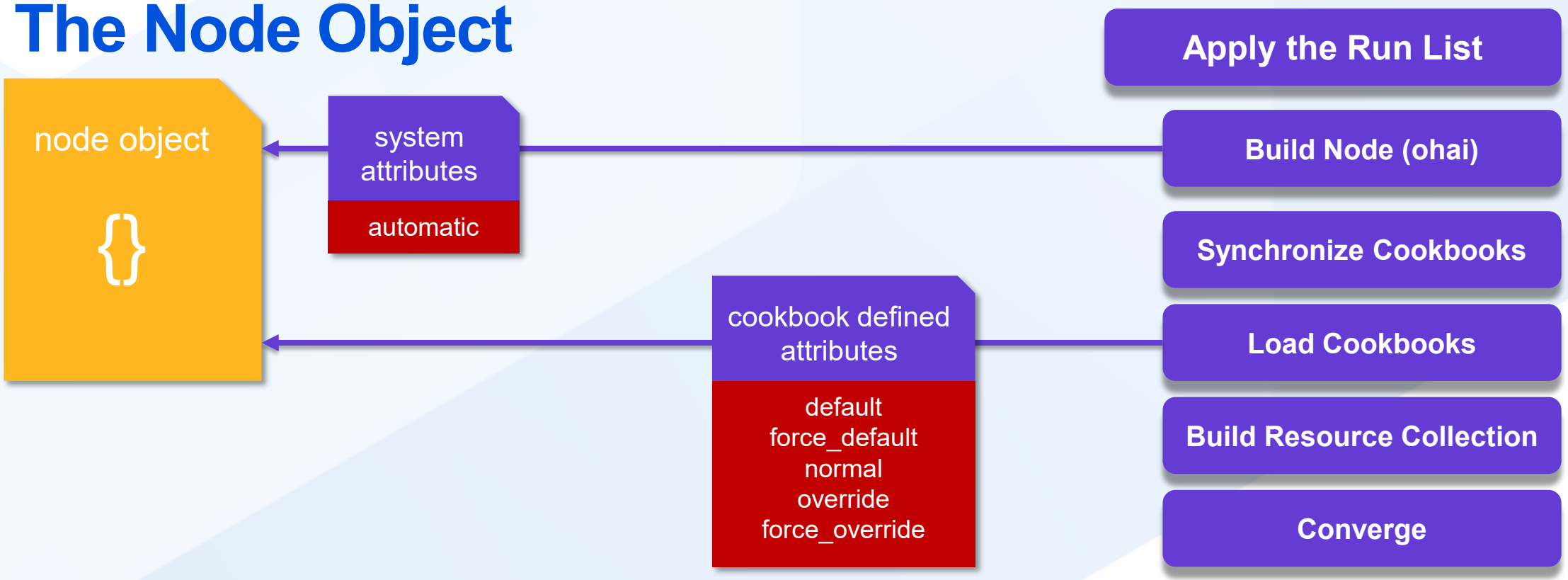
Objectives

After completing this module, you should be able to:

- Refactor resources to use attributes

Concept

The Node Object



https://docs.chef.io/attribute_precedence/

Current Recipe with Hard-Coded Values

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

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome Home!</h1>'
end

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

Create the Node Attributes

~/chef-repo/cookbooks/apache/attributes/default.rb

```
default['apache']['package_name'] = 'httpd'
default['apache']['service_name'] = 'httpd'
default['apache']['default_index_html'] = '/var/www/html/index.html'
```

Update the Recipe to Use the Node Attributes

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

```
package node['apache']['package_name']

file node['apache']['default_index_html'] do
  content '<h1>Welcome Home!</h1>'
end

service node['apache']['service_name'] do
  action [:enable, :start]
end
```

Concept

Discussion

What are the benefits of providing the package name and service name as node attributes?

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 06: Refactoring Using Attributes

- Refactor the Install recipe to use a Node attribute
- Execute the tests and verify that the tests fail
- Create the attributes file and add the Node attribute
- Execute the tests and verify that the tests pass

Time to remove all the hard-coded values and make them attributes.

Progress®
Chef®

Refactor Testing for Multiple Platforms

Objectives

After completing this module, you should be able to:

- Define expectations for multiple platforms
- Implement a cookbook that supports multiple platforms

Keep in Mind the TDD Approach

In order to refactor our cookbook for another platform we should follow these steps:

- Create new unit tests
- Create new integration tests
- Write code that causes the tests to pass

Update the Unit Test

~/chef-repo/cookbooks/apache/spec/unit/recipes/default_spec.rb

```
  # ... REST OF SPEC FILE ...
  context 'When all attributes are default, on Ubuntu 20.04' do
    Platform 'ubuntu', '20.04'

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs the apache2 package' do
      expect(chef_run).to install_package('apache2')
    end
  # ... MORE TESTS ADDED IN LAB ...
  end
end
```

Add the Ubuntu platform to the Kitchen Configuration

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

```
---
platforms:
  - name: centos-7
    driver:
...
  - name: ubuntu-20.04
    driver:
      image: dokken/ubuntu-20.04
      privileged: true
      pid_one_command: /bin/systemd/systemd
      volumes:
        - /sys/fs/cgroup:/sys/fs/cgroup:ro # required by systemd
```

Switching on Node Platform

To control the flow of execution we need to employ some Ruby conditional statements. Conditional statements allow us to alter this control flow. Because we have access to the power of Ruby we have many choices.

https://docs.chef.io/infra_language/checking_platforms/

Update the Attribute to Support Platforms

~/chef-repo/cookbooks/apache/attributes/default.rb

```
case node['platform']
when 'ubuntu'
  default['apache']['package_name'] = 'apache2'
  default['apache']['service_name'] = 'apache2'
  default['apache']['default_index_html'] = '/var/www/html/index.html'
else
  default['apache']['package_name'] = 'httpd'
  default['apache']['service_name'] = 'httpd'
  default['apache']['default_index_html'] = '/var/www/html/index.html'
end
```

Heckle the code

~/chef-repo/cookbooks/apache/attributes/default.rb

```
case node['platform']
when 'ubuntu'
  default['apache']['package_name'] = 'apache2'
  default['apache']['service_name'] = 'apache2'
  default['apache']['default_index_html'] = '/var/www/html/index.html2'
else
  default['apache']['package_name'] = 'httpd'
  default['apache']['service_name'] = 'httpd'
  default['apache']['default_index_html'] = '/var/www/html/index.html'
end
```

Concept

Discussion

What are the benefits and drawbacks of defining unit tests for multiple platforms?

What are the benefits and drawbacks of defining integration tests for multiple platforms?

When testing multiple platforms would you start with integration tests or unit tests?

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 07: Refactor Testing for Multiple Platforms

- Write a test that verifies the Install recipe chooses the correct package on CentOS & Ubuntu
- Execute the tests and verify that the tests fail
- Update the attribute to provide support for CentOS & Ubuntu
- Execute the tests and verify that the tests pass

The best of both worlds!

Progress®
Chef®

Introduction to Ruby, IRB

Objectives

After completing the lesson, you should be able to:

- Explain what a REPL is, and how to use it
- Use the basics of the Ruby language

Continued on the next slide...

Objectives Continued

After completing the lesson, you should be able to describe the function of the following in Ruby:

- Variable Assignment
- Basic Arithmetic
- Strings
- Truthiness
- Operators
- Hashes
- Conditionals
- Method Declaration
- Classes
- Objects

irb - the interactive ruby shell

- irb is the interactive ruby shell
- It is a REPL for Ruby
  * Read-Eval-Print-Loop
- An interactive programming environment that is super-handy for trying things out
- A basic understanding of Ruby is needed for more advanced topics that we will cover tomorrow

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 08: Intro to Ruby

- Use the Interactive Ruby Shell

Progress®
Chef®

Adding Ruby Gems

Objectives

After completing this module, you should be able to:

- Define a Ruby gem
- Know when to look for a Ruby gem
- Load a Ruby gem

Concept

Intro to Ruby Gems

- Are simply open source libraries and classes that contain Ruby code and provide extra functionality
- Can be installed as needed similar to RPM packages in RedHat
- Rspec is a Ruby gem

Concept

When to Use Ruby Gems

- Existing Chef resources are not able to accomplish a task
- Tasks can be made simpler by having additional functions
- This lab will add a gem that enables access to Hashicorp Vault, but many of the popular secret management platforms also have a gem available.

How to Load Ruby Gems

- All machines with Chef Infra Client installed have two instances of Ruby. One is the standard, system-wide instance of Ruby, the other is a dedicated instance that is available only to Chef Infra Client
- Use the **chef_gem** resource to install gems into the instance of Ruby that is dedicated to Chef Infra Client
- Use the **gem_package** resource to install gems system-wide

https://docs.chef.io/resources/chef_gem/

https://docs.chef.io/resources/gem_package/

Where to Find Ruby Gems

- https://rubygems.org/
- Gems can be stored on an internal server when security is a concern
- Both of the Chef resources for adding gems have a source property that can be used to specify where to pull Ruby gems from

Adding a Ruby Gem

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

```
chef_gem 'vault' do
  compile_time true
end

require 'vault'
```

Using new functionality

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

```
Vault.configure do |config|
  # The address of the Vault server
  config.address = 'http://IP_ADDRESS:8200'

  # The token to authenticate with Vault
  config.token = 'ROOT_TOKEN'
end

creds = creds = Vault.kv('secret').read('password')
```

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 09: Adding Ruby Gems

- Setup Hashicorp Vault
- Store a password
- Add ‘vault’ Ruby Gem
- Access password stored in vault

Progress®
Chef®

Creating Custom Resources

Objectives

After completing this module, you should be able to:

- Create a custom resource

Custom Resources

- Packages of code
- Can be written using Chef Resources, Ruby code, or a combination of both
- Predecessors:
  - HWRP's - Pure Ruby
  - Definitions - Uses Chef resources, but limited in functionality
  - LWRP's - Like Custom Resources, but lacking a few features

Custom Resources

- Stored in the *cookbookname*/resources folder
- Named derived from *cookbookname_filename*
- Full path is ~/chef-repo/cookbooks/apache/resources/vhost.rb
  - Cookbook name = apache
  - file name = vhost
  - Resource name = apache_vhost

Custom Resources

Can define:

- Actions
- Default action
- Allowed inputs (properties)
  - Property types
  - Property default values
  - Name attributes (a mapping to a particular property)

Creating Apache Vhosts

- Create a custom resource that creates an apache vhost
- Allow the custom resource to have configurable properties

This will make our recipe much cleaner.

Implementing the Create Action

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
action :create do
  directory '/srv/apache/admins/html' do
    recursive true
    mode '0755'
  end

  template '/etc/httpd/conf.d/admins.conf' do
    source 'conf.erb'
    mode '0644'
    variables(document_root: '/srv/apache/admins/html',port: 8080)
    notifies :restart, 'service[httpd]'
  end

  file '/srv/apache/admins/html/index.html' do
    content '<h1>Welcome admins!</h1>'
  end
end
```

Using the New Custom Resource

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

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Welcome home!</h1>'
end

apache_vhost 'admins' do
  action :create
end

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

Questions?

Q&A

What questions can we answer for you?

Progress Chef

Exercise

Lab 10: Creating Custom Resources

Create a new custom resource for the Apache cookbook

Progress®
Chef®

Extending Custom Resources

Objectives

After completing this module, you should be able to:

- Use properties to make custom resources more flexible
- Define a new action within the custom resource

Concept

Hard coded site name!

We have a problem in that our vhost name, 'admins', is hardcoded throughout the 'resources/vhost.rb' and 'recipe/default.rb' files.

This make the custom resource inflexible and difficult to change.

We should make that configurable.

Resource Properties

A property is defined with the following syntax.

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
property :site_name, String
```

1 property method

2 name (symbol)

3 type

4 Optional Parameters

Resource Properties

Properties are defined in the resource definition and then available within definition of the resource within the recipe.

Usage in recipe

```
apache_vhost 'admins' do
  site_name 'admins'
  action :create
end
```

Creation in custom resource

```
property :site_name, String
```

Resource Properties

Using properties created in the custom resource requires the new_resource method. This is to avoid any potential conflicts with existing property names.

Usage in custom resource

```
directory "/srv/apache/#{new_resource.site_name}/html" do
  recursive true
  mode "0755"
end
```

https://docs.chef.io/custom_resource_glossary/#new_resourceproperty

Resource Actions

Custom resources can have multiple actions defined that expand its functionality.

Common examples are create and remove

Remove the Welcome Site

When apache installs itself it defines a default site on port 80. Up until this point we have relied on this site. We now want to remove that initial welcome site but we want to do that we a new action we will define on the custom resource we are defining.

Defining the Remove Action

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
…
# ... CREATE ACTION ...

action :remove do
  directory "/srv/apache/#{new_resource.site_name}" do
    action :delete
  end

  file “#{node[‘apache’][‘conf_dir’]}/#{new_resource.site_name}.conf” do
    action :delete
  end
end
```

Progress Chef

Using the Remove Action

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

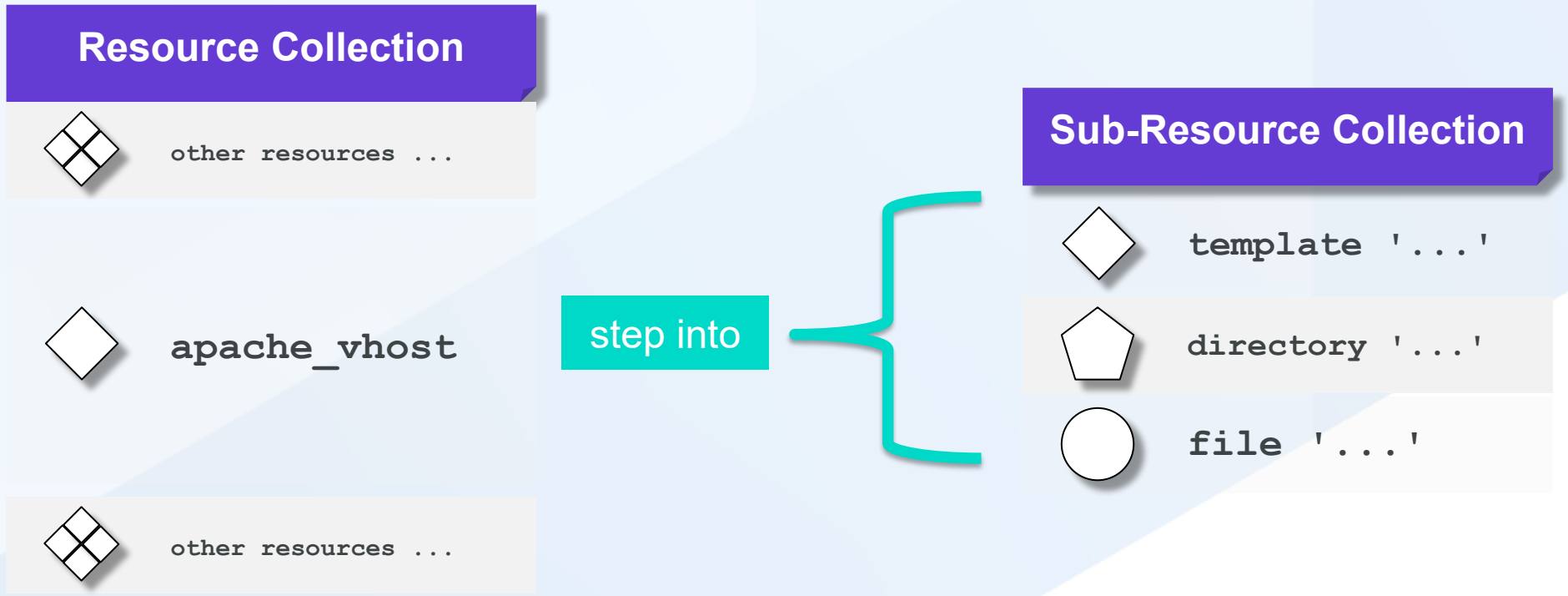
```
package 'httpd'

apache_vhost 'welcome' do
  site_name 'welcome'
  action :remove
end

apache_vhost 'admins' do
  site_name 'admins'
  site_port 8080
  action :create
end
# ... REMAINDER OF RECIPE ...
```

Testing Custom Resources

- Custom resources add a layer of difficulty to our testing
- They are in a sub-resource collection
- The step_into method is required to test custom resources

Concept

A Sub-Resource Collection



Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 11: Extending Custom Resources

- Update an action with a custom resource to use the property
- use step_into

Progress Chef

Progress®
Chef®

Refining Custom Resources

Objectives

After completing this module, you should be able to:

- Define the default action for a custom resource
- Set a custom resource's name to a property
- Set a default value for a custom resource property
- Define notifications correctly when creating custom resources

Progress Chef

Selecting a Default Action

Resources often have a default action. The default action is often the action that is the least surprising. For most resources it is an additive/restorative action that moves the system into the desired state.

The First Action is the Default

The first action within the custom resource definition is the default one. But you can explicitly define the default action within the resource definition.

Defining a Default Action for the Resource

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
property :site_name, String
property :site_port, Integer

default_action :create

action :create do
  directory "/srv/apache/#{new_resource.site_name}/html"
do
    recursive true
    mode "0755"
  end
# ... REMAINING RESOURCE DEFINITION ...
```

Removing the Action Property

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

```
# ... INITIAL RECIPE DEFINITION ...

apache_vhost 'users' do
  site_port 80
  site_name 'users'
  action :create
end

apache_vhost 'admins' do
  site_port 8080
  site_name 'admins'
  action :create
end
```

Clarity in the Custom Resource

There is some duplication in the declaration of the resource. The name of the resource and the site_name. We want to default to use the name specified in the resource as the site_name. This is similar to other resources.

Tying the Resource Name to the Property

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
property :site_name, String, name_attribute: true
property :site_port, Integer

default_action :create

action :create do
  directory "/srv/apache/#{new_resource.site_name}/html"
do
    recursive true
    mode "0755"
  end
# ... REMAINING RESOURCE DEFINITION ...
```

Removing the site_name Property

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

```
package 'apache'

apache_vhost 'users' do
  site_port 80
  site_name 'users'
end

apache_vhost 'admins' do
  site_port 8080
  site_name 'admins'
end
...
```

Setting Default Values for Properties

Properties can also have default values. When deploying a website the default port that one would expect to see that site is on port 80.

Setting a Default Value for the Property

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
property :site_name, String, name_attribute: true
property :site_port, Integer, default: 80

default_action :create

action :create do
  directory "/srv/apache/#{new_resource.site_name}/html"
do
    recursive true
    mode "0755"
  end
# ... REMAINING RESOURCE DEFINITION ...
```

Removing the site_port Property

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

```
package 'apache'

apache_vhost 'users' do
  site_port 80
end

apache_vhost 'admins' do
  site_port 8080
end

...
```

Resource Notifications

Defining a notification in a resource within a custom resource creates a fragile relationship. One that we want to address by removing any notifications to outside resources.

Removing the Notification from the Resource

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
# ... INITIAL RESOURCE DEFINITION ...

 template "/etc/httpd/conf.d/#{new_resource.site_name}.conf" do
    source "conf.erb"
    mode "0644"
    variables(
    document_root:
"/srv/apache7#{new_resource.site_name}/html",
         port: new_resource.site_port)
    notifies :restart, "service[httpd]"
  end
# ... REMAINDER OF CUSTOM RESOURCE ...
```

Adding the Notification to the Resources

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

```
apache_vhost node['apache']['site_name'] do
      site_name node['apache']['site_name']
  notifies :restart, node['apache']['service_resource']
  action :remove
end

apache_vhost 'users' do
  notifies :restart, node['apache']['service_resource']
end

apache_vhost 'admins' do
      site_port 8080
  notifies :restart, node['apache']['service_resource']
end
```

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 12: Refining Custom Resources

- Define a default action for the custom resource
- Set the resource name as the site_name property
- Set the default value of the site_port property
- Move the resource notifications to the recipe

Progress®
Chef®

Completing Unit Test suite

Objectives

After completing this module, you should be able to:

- Complete the unit testing suite

Concept

Unit testing

As you recall, unit testing is done at the resource level. Our current unit testing suite is only testing some of our resources which could allow errors to get past.

Untested Resources

~/chef-repo/cookbooks/apache/resources/vhost.rb

```
action :create do
  directory "/srv/apache/#{new_resource.site_name}/html" do
    recursive true
    mode '0755'
  end

  template "#{node['apache']['conf_dir']}/#{new_resource.site_name}.conf" do
    source 'conf.erb'
    mode '0644'
    variables(document_root: "/srv/apache/#{new_resource.site_name}/html",
port: new_resource.site_port)
  end

  file "/srv/apache/#{new_resource.site_name}/html/index.html" do
    content "<h1>Welcome #{new_resource.site_name}!</h1>"
  end
end
```

New Unit tests

~/chef-repo/cookbooks/apache/spec/unit/recipes/default_spec.rb

```
it 'creates the conf directory' do
      expect(chef_run).to create_directory('/srv/apache/users/html')  +
    end

    it 'creates the conf directory' do
      expect(chef_run).to create_directory('/srv/apache/admins/html') +
    end

    it 'creates the conf file' do
      expect(chef_run).to render_file('/etc/apache2/sites-                +
enabled/users.conf')
    end

    it 'creates the conf file' do
      expect(chef_run).to render_file('/etc/apache2/sites-                +
enabled/admins.conf')
    end
```

Questions?

Q&A

What questions can we answer for you?

Exercise

Lab 13: Completing Unit Test suite

- Complete the unit testing suite

Progress
Chef

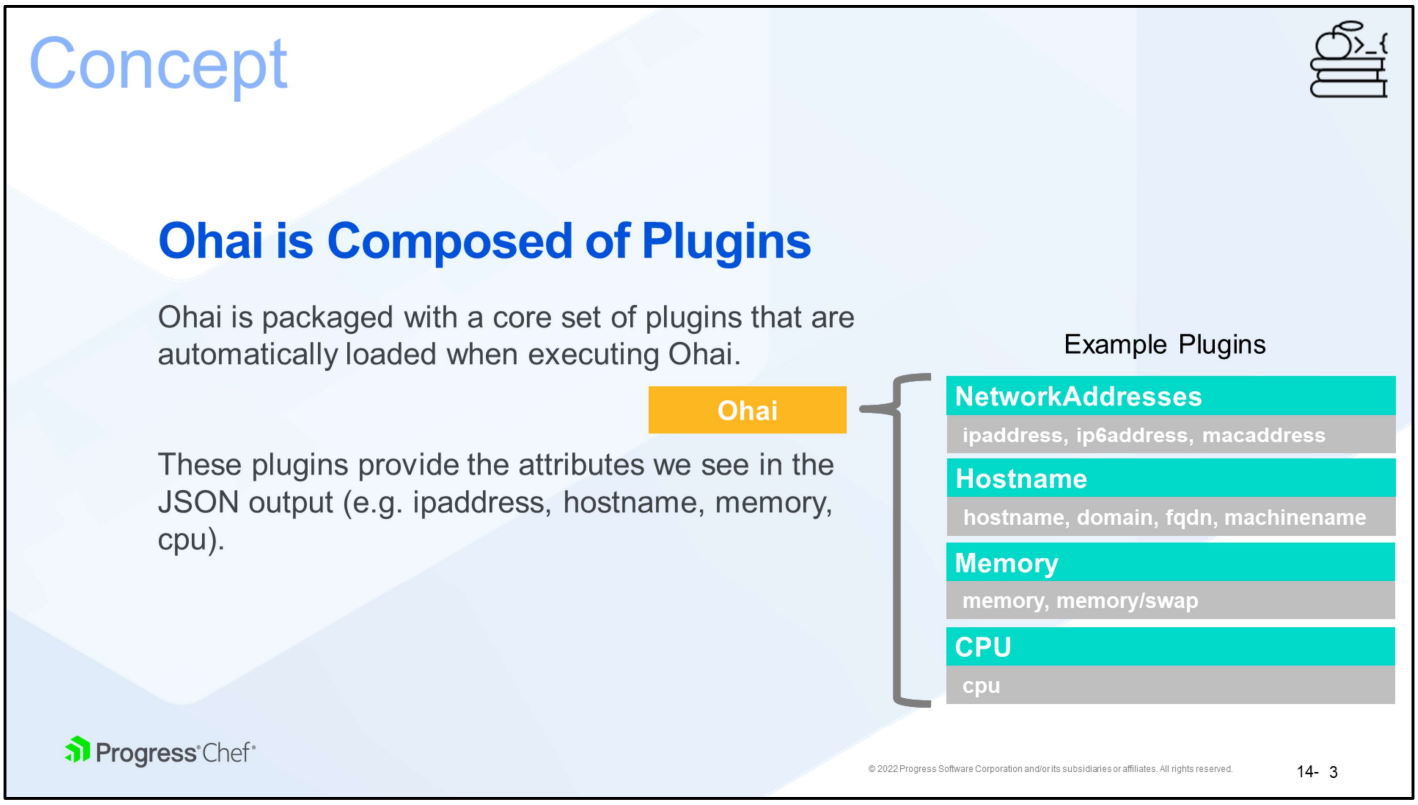
Creating Ohai Plugins

Objectives

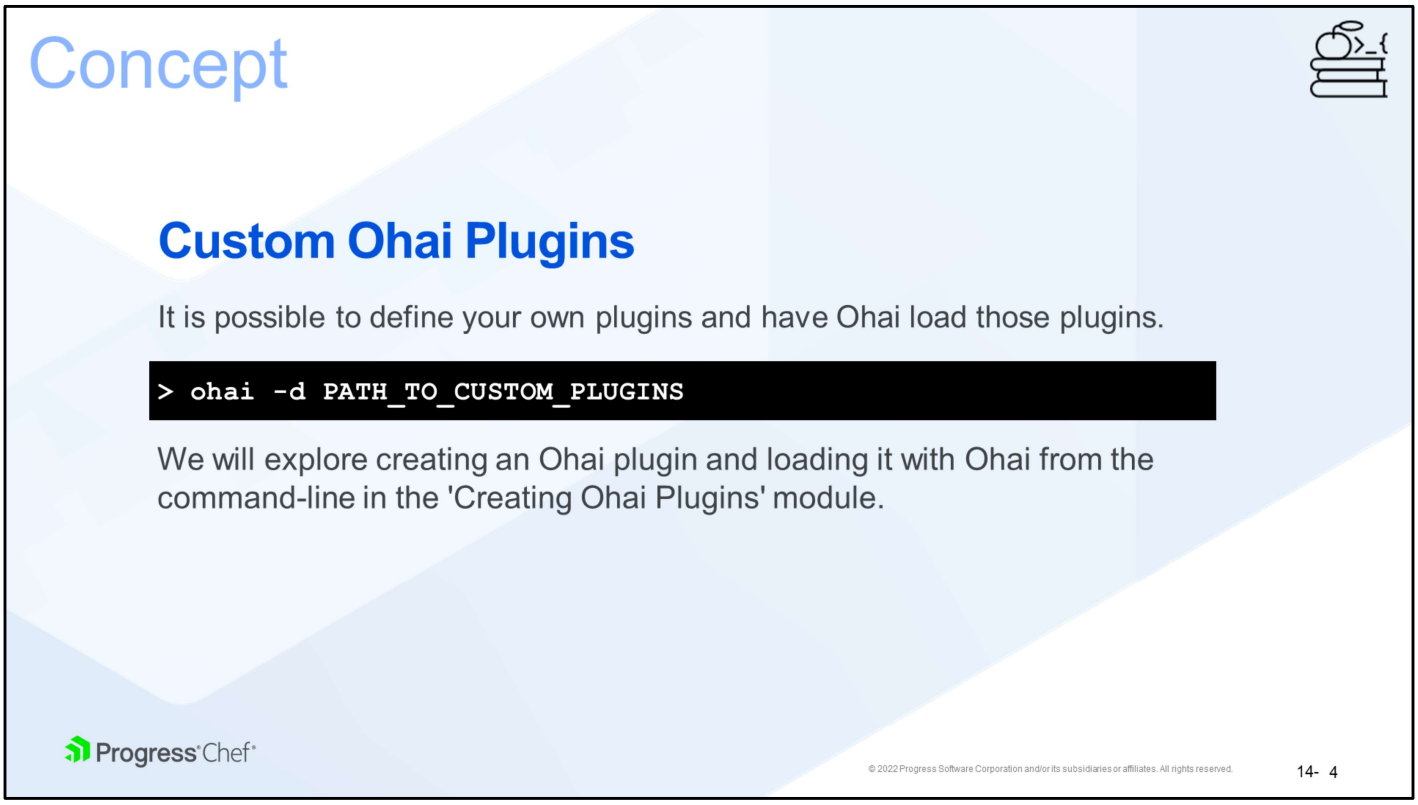
After completing this module, you should be able to:

- Create an Ohai plugin
- Test Ohai plugin using Test Kitchen

Progress Chef

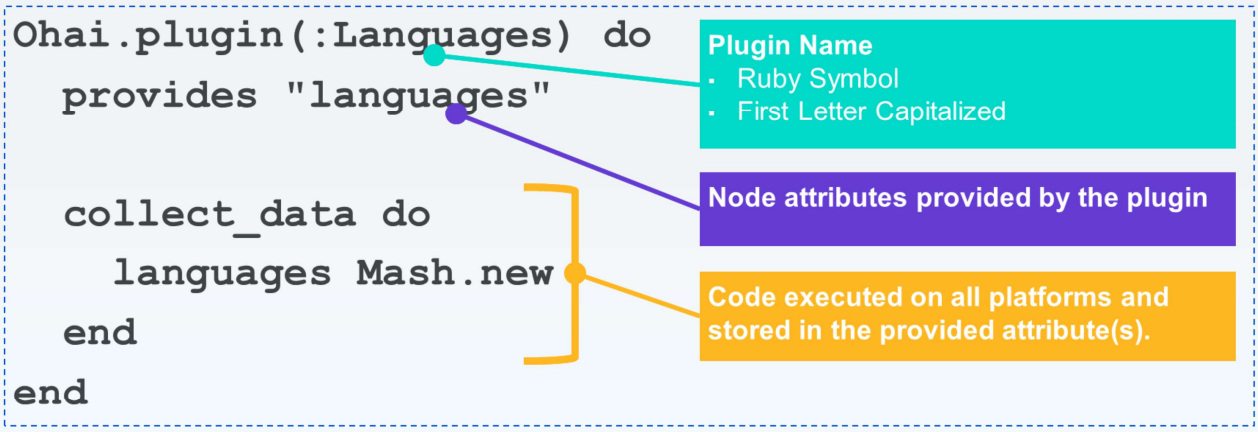


As we saw in the previous module Ohai provides a large set of attributes that it provides through plugins. All the data that Ohai collects are stored in plugins. Ohai comes packaged with a core set of plugins that capture a lot of common data across many different platforms.



Ohai is composed of plugins that collect these different attributes. When you execute Ohai it will load the core plugins that are packaged with it.

Viewing the Languages Plugin



The Languages plugin provides the node attribute 'languages' which is populated, on all platforms, with a Mash.

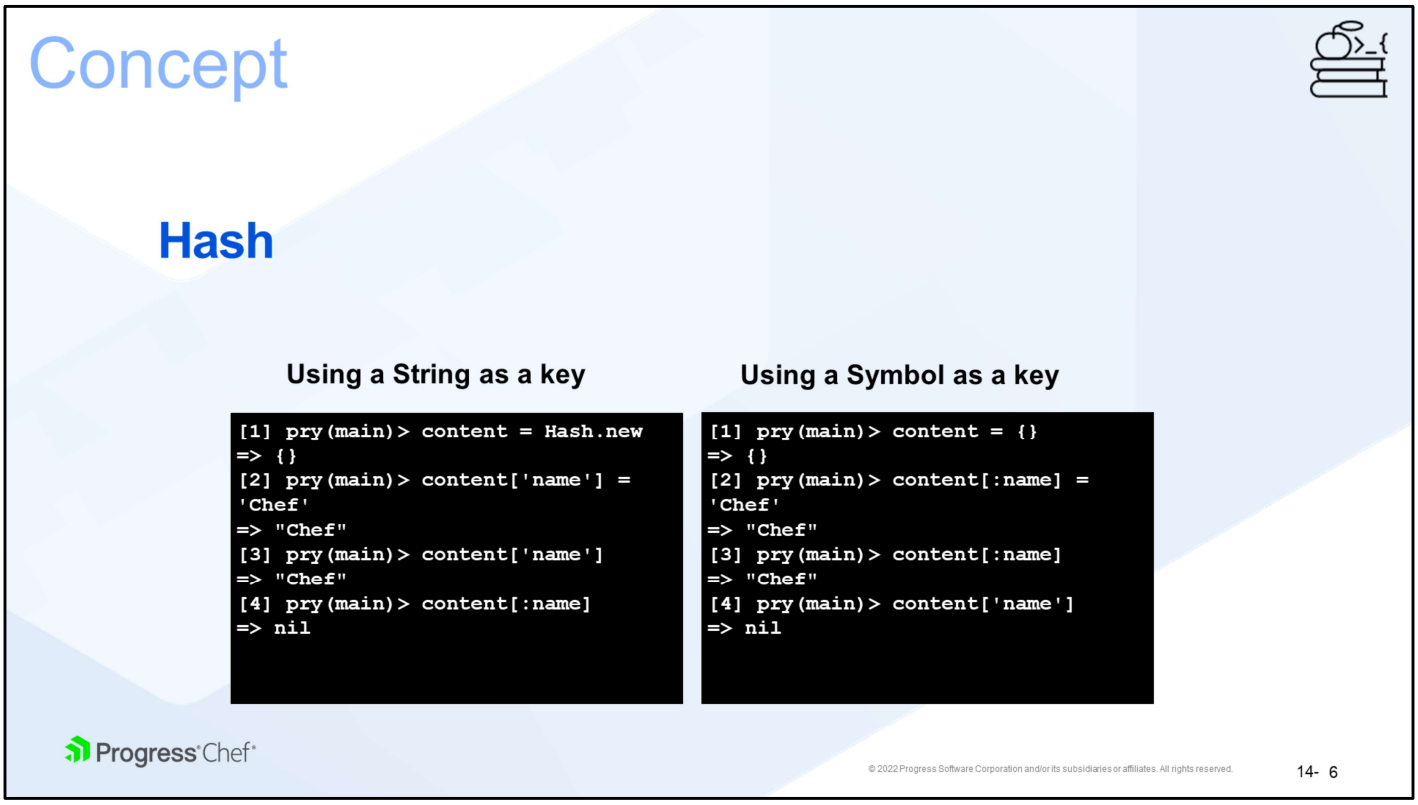
Progress Chef

A plugin starts with invoking a method on the Ohai class with a single parameter. That parameter provided is the symbol name of the plugin. All Ohai plugins must have a symbol name with the first letter capitalized.

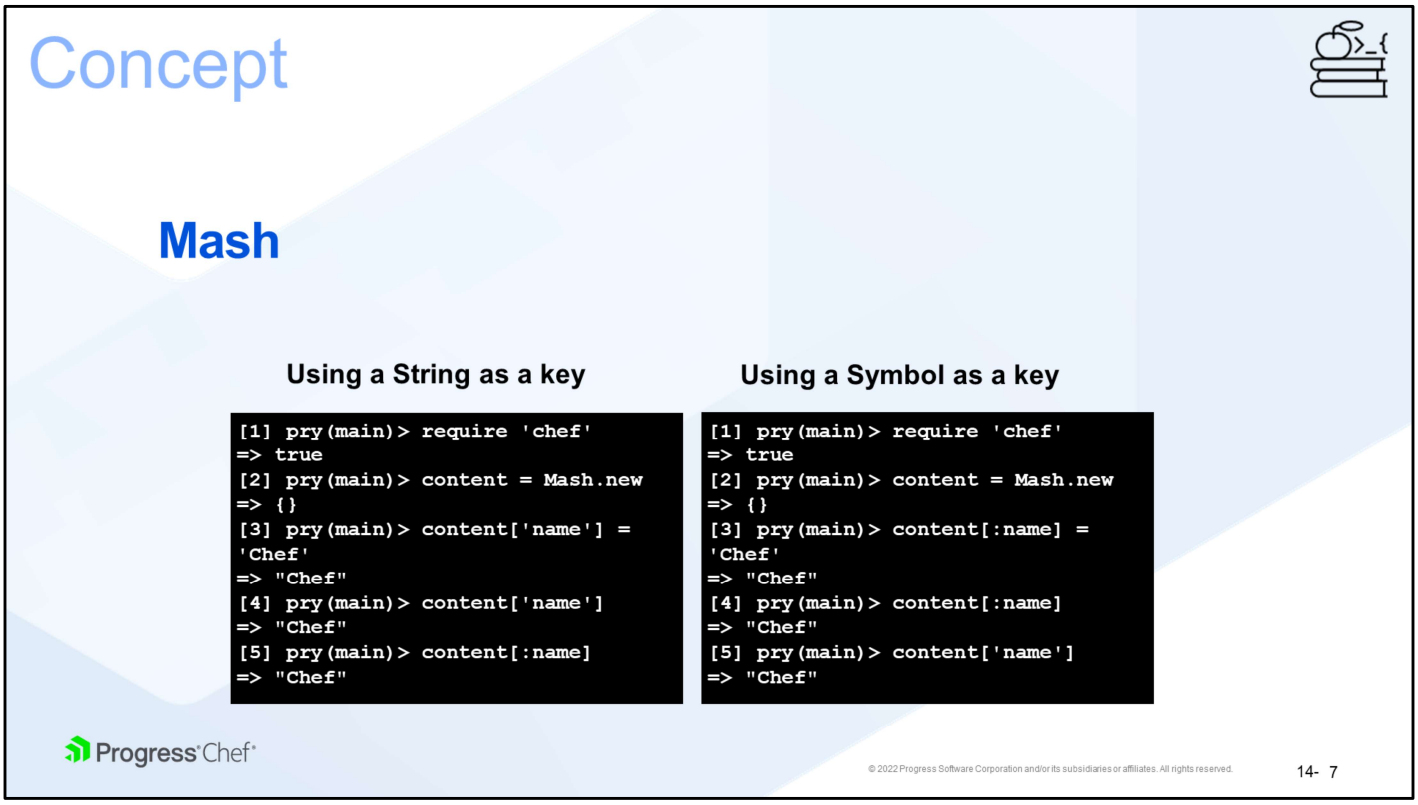
The remainder of the plugin is defined within the block of the 'plugin' method. The 'provides' method specifies what attribute or attributes the plugin will be added to the node object. The 'collect_data' method defines a block which contains the code that is executed on all platforms. This block of code will often times set the values of the attributes the plugin provides.

This plugin is named Languages. It provides the languages attribute on the node. This languages attribute is populated with the contents of a new Mash.

But what is a Mash?



To understand what a Mash is first let's talk about Ruby's Hash. Hashes allow you to store values with a key; often times these keys are Ruby Strings or Ruby Symbols. When you want to retrieve that value you need to provide the same key. So if say you stored data with a Symbol key it is only retrievable with a Symbol key. The same could be said for using a String key.



A Mash is similar to a Ruby Hash except that it is indifferent to whether you provide it a String key or Symbol key. Either of those types of keys will return value stored by the other. This more lenient data structure allows for these two keys to be used interchangeably. Allowing us to use whichever key style we prefer without being penalized if we were to guess the key style that differs from other plugins.

Viewing the Python Plugin

~/ohai/lib/ohai/plugins/python.rb

```
Ohai.plugin(:Python) do
  provides "languages/python"

  depends "languages"

  collect_data do
    begin
      so = shell_out("python -c \"import sys; print (sys.version)\"")
      # Sample output:
      # 2.7.11 (default, Dec 26 2015, 17:47:53)
      # [GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)]
      if so.exitstatus == 0
        python = Mash.new
        output = so.stdout.split
        python[:version] = output[0]
        if output.length >= 6
```

Node attributes provided by the plugin

This plugin depends on the attributes in the Languages to be defined

Progress Chef

Here within the Python plugin we see the same structure with a dependency and a significant amount of work being done in the 'collect_data' method block. The attribute provided by this plugin can be found on the node object under the specified path. Remember this is the same path structure you use on the command-line when wanting to traverse the attributes provided.

The dependency described here states that this plugin requires that the node attribute value 'languages' must be defined first before this plugin will execute. Ohai will determine how to execute the plugins based on these dependencies.

Viewing the Python Plugin

~/ohai/lib/ohai/plugins/python.rb

```
Ohai.plugin(:Python) do
  provides "languages/python"

  depends "languages"

  collect_data do
    begin
      so = shell_out("python -c \"import sys; print (sys.version)\"")
      # Sample output:
      # 2.7.11 (default, Dec 26 2015, 17:47:53)
      # [GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)]
      if so.exitstatus == 0
        python = Mash.new
        output = so.stdout.split
        python[:version] = output[0]
        if output.length >= 6
```

Progress Chef

Within the collect_data block we use a helper method named 'shell_out'. This 'shell_out' method accepts a single parameter which is the command to run. This 'shell_out' method will generate an object for which you can ask for the standard output, standard error, and the exit status.

This command is executed and if the status is successful (0 status code) then look at the standard output, split it into multiple lines, extract the version and possibly any build date information, and then store that information into the Mash that was created by the Languages plugin. If a failure occurs at any point catch that error and display a debug message.

You will find that most Ohai plugins will fit the following pattern. Perform a system related call to collect some data, use Ruby to process that data, and then store the data.

Concept

collect_data code blocks

Ohai plugins can contain many collect_data blocks. It is important to remember that only **ONE** collect_data block will be run.

```
collect_data(:default) do
  # some Ruby code
end

collect_data(:windows) do
  # some Ruby code
end

collect_data(:linux) do
  # some Ruby code
end
```

https://docs.chef.io/ohai_custom/#collect_data

Progress Chef

Concept

Community Ohai Plugins

There are a few Ohai plugins developed and maintained by the Chef community that are available. They can be used as is, modified to fit your needs, or as an example of what is possible.

https://docs.chef.io/plugin_community/

Progress Chef

Questions?

Q&A

What questions can we answer for you?

Progress Chef

Exercise

Lab 14: Creating Ohai Plugins

- Develope custom Ohai plugin
- Write test cases to test the custom ohai plugin
- Manually validate the included attributes using ohai.

Progress Chef

Progress Chef

Progress®
Chef®

Intro to CI/CD Pipeline

Objectives

After completing this module, you should be able to:

- Understand what a CI/CD pipeline is used for
- Identify common CI/CD pipeline operations

What is a CI/CD Pipeline

- CI = Continuous Integration
- CD = Continuous Delivery
- Automates the process of testing and publishing code
- Focus on removing human intervention from the process
- CI and CD processes can happen independently of each other
- CI/CD pipelines are often a collection of tools that work together, i.e. SCM, pipeline, virtualization, etc.

CI/CD Pipeline options

- Countless options are available, both commercial and open-source
- Jenkins is a popular open-source option
- Azure Pipelines is offered by Microsoft
- AWS DevOps is offered by Amazon
- The best option is usually the one that integrates with your existing tools

Concept

Common CI/CD Pipeline Operations

- Pull updated code from branch
- Validate version number was incremented
- Run unit and integration tests
- Send notification if there are any failures
- Create pull request

Questions?

Q&A

What questions can we answer for you?

Exercise

CI/CD pipeline demo

Installation, configuration, and building a CI/CD pipeline is outside the scope of this course. We do have a simple example to share.