

Refactoring Cookbooks with Tests



We explored the process of developing a test first but to explore the full Test Driven Development (TDD) cycle we need to refactor the code that we wrote.

Refactoring is the process of making changes to the implementation while maintaining the original intention. Without having tests that capture the original intention how do you know if the new implementation did not change the original intention? Fortunately for us we have defined a test that will allow us to make the changes confident that we have not destroyed that original intention.

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**

Refactoring is the often forgotten step in the TDD cycle. When we are able to get our expectations to pass we immediately want to move to our next requirement or next cookbook.

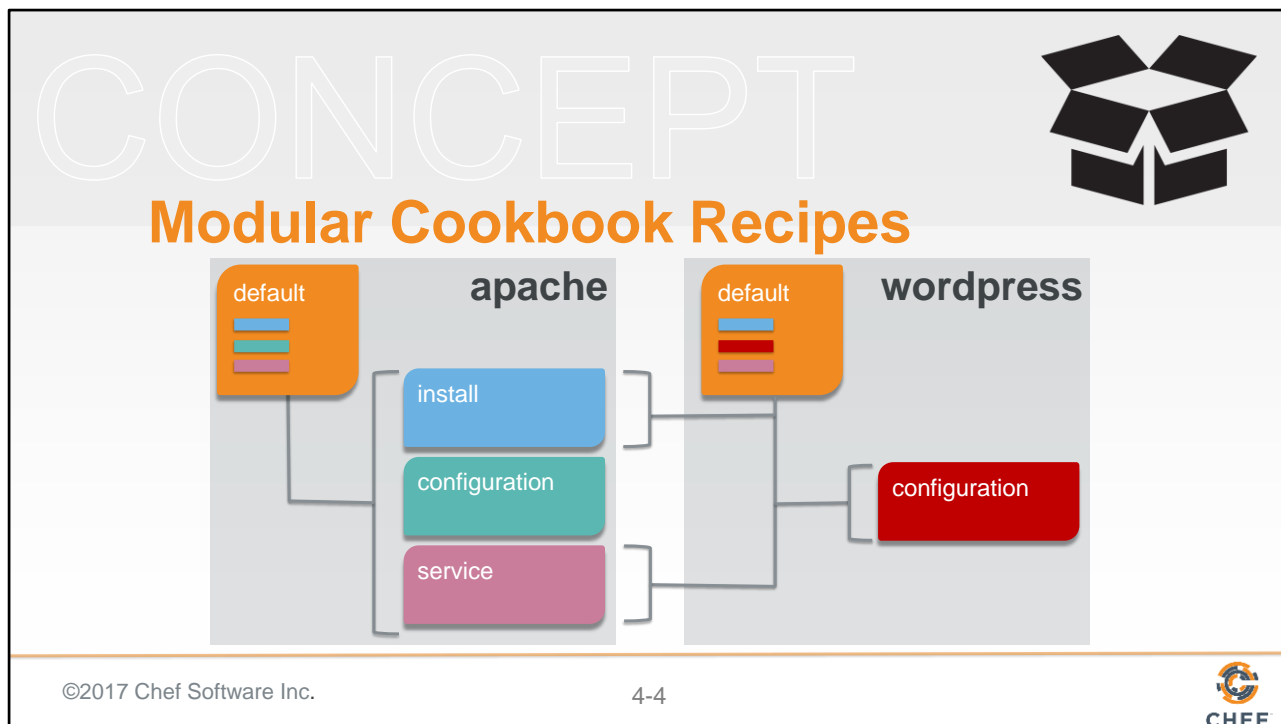
This step is incredibly important. Within it we are able to reflect on the unit of code and tests that we have written and evaluate them. How you evaluate the code may vary based on your experience, the standards defined by the team you work with, or if the code will be shared with the Chef community.

Objectives

After completing this module, you should be able to:

- Refactor a recipe using `include_recipe`
- Use Test Kitchen to validate the code you refactored
- Explain when to use `kitchen converge`, `kitchen verify` and `kitchen test`.

In this module you will learn how to refactor a cookbook using the method 'include_recipe', verify the changes with Test Kitchen, and then explain in what scenarios you would choose to use 'kitchen converge', 'kitchen verify' and 'kitchen test'.



Our initial implementation of the default recipe for the apache cookbook defined the entire installation, configuration, and management of the service within a single recipe. This implementation has the benefit of being entirely readable from a single recipe. However, it does not easily allow for other cookbooks that may want to use the apache cookbook to easily choose the components that it may need.

An example of this is that we may deploy wordpress or some other web application that relies on the apache webserver installed and running. In this new cookbook we would like to re-use the resources that installs apache and the resources that manage the service. We most likely do not want to setup a test page that greets people. We are likely going to replace it with application code.

CONCEPT



include_recipe


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

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

The 'include_recipe' method can be used to include recipes from the same cookbook or external cookbooks. It allows us to accomplish what we saw previously. This gives us the ability to build recipes in more modular ways promoting better re-use patterns within the cookbooks we write.

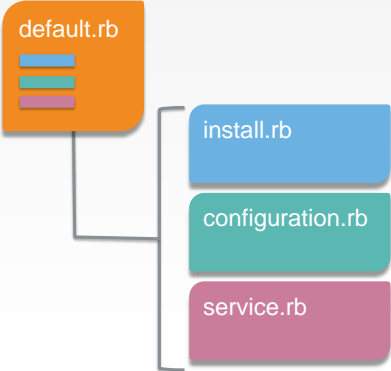
CONCEPT

Recipe Organization



recipes/default.rb


```
include_recipe 'cookbook::install'
include_recipe 'cookbook::configuration'
include_recipe 'cookbook::service'
```



```
graph LR
    default[default.rb] --- install[install.rb]
    default --- config[configuration.rb]
    default --- service[service.rb]
```

©2017 Chef Software Inc.

4-6



To allow better re-use we can choose to refactor a single recipe into more modular recipes that focus on their individual concerns. Then these recipes can be included into the original single recipe through the 'include_recipe' method.

EXERCISE



Refactor to Modular Recipes

*This is why we **can** have nice things!*

Objective:

- ☐ Refactor the installation into a separate recipe
- ☐ Converge the cookbook and execute the tests

You called?



This more modular approach to recipes is very common as the complexity of the cookbook continues to grow. The complexity of the cookbook we are developing is not there, nor will it ever be there for the entirety of this course. However, we are still going to use this opportunity to prematurely optimize to demonstrate the refactoring of a cookbook.

Together we will work through creating a recipe that manages the installation of the webserver.

Ask Chef About Generating a Recipe



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

```
Usage: chef generate recipe [path/to/cookbook] NAME [options]
```

-C, --copyright COPYRIGHT	Name of the copyright holder...
-m, --email EMAIL	Email address of the author...
-a, --generator-arg KEY=VALUE	Use to set arbitrary arguments...
-I, --license LICENSE	all_rights, apache2, mit, ...
-g GENERATOR_COOKBOOK_PATH, --generator-cookbook	Use GENERATOR_COOKBOOK_PATH...

First let's return to the chef generator tool and it what information it needs to generate a recipe within a cookbook. The recipe generator can be run from within a cookbook or outside of it. If you are within a cookbook you do not need to specify a path to the cookbook; it's optional.

Generate an Install Recipe



```
> chef generate recipe install
```

```
Recipe: code_generator::recipe
  * directory[/home/chef/apache/spec/unit/recipes] action create
  (up to date)
  * cookbook_file[/home/chef/apache/spec/spec_helper.rb] action
  create_if_missing (up to date)
  * template[/home/chef/apache/spec/unit/recipes/install_spec.rb]
  action create_if_missing
    - create new file
    /home/chef/apache/spec/unit/recipes/install_spec.rb
    - update content in file
    /home/chef/apache/spec/unit/recipes/install_spec.rb from none to
    187413
```

Since we are within the cookbook directory you simply need to provide it the name of the recipe you want created.

Instructor Note: The generator will create the recipe file with the recipes directory and also a spec file within the unit test directory. Unit testing is a topic that we will discuss in the next module.

Removing the Generated Test File



```
> rm test/smoke/default/install.rb
```

When you use chef, the command-line tool, to generate a recipe it will create three files. First is the recipe file found in the recipes directory. Second is the unit test file found in the 'spec/unit/recipes' directory. Third is the integration test file found in 'test/smoke/default'.

The test file automatically generate for us contains those same examples we saw in the 'default_test.rb'. We do not want to verify the root user is present and we definitely do not want to verify that port 80 is not listening. So we want to remove this file.

Write the Install Recipe



```
~/apache/recipes/install.rb
```

```
#  
# Cookbook Name:: apache  
# Recipe:: install  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
package 'httpd'
```

The installation of the web server can be expressed with this one resource. Within the new recipe add the following resource.

Remove the Resource from the Default Recipe

 ~/apache/recipes/default.rb

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2015 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
```

Now that we have defined the installation of the webserver in a separate recipe it is time to remove the installation from the default recipe.

Include the Install Recipe



~/apache/recipes/default.rb

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
include_recipe 'apache::install'  
  
file '/var/www/html/index.html' do  
  content '<h1>Welcome Home!</h1>'  
end  
  
service 'httpd' do  
  action [:enable, :start]  
end
```

Replacing it with the 'include_recipe' method that retrieves the contents of that recipe and includes it here.

EXERCISE



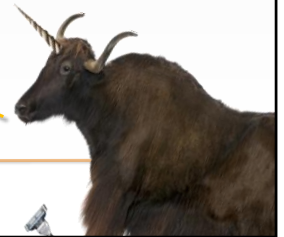
Refactor to Modular Recipes

*This is why we **can** have nice things!*

Objective:

- ✓ Refactor the installation into a separate recipe
- ❑ Converge the cookbook and execute the tests

I see what you did there.



The default recipe has changed. It is now time to ensure that we did everything right by converging the latest changes against the test instance and then verifying the changes by executing our tests.

Re-Converge the Test Instance



```
> kitchen converge
```

```
----> Starting Kitchen (v1.11.1)
----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
...
----> Installing Chef Omnibus (install only if missing)
Downloading https://www.chef.io/chef/install.sh to file...
resolving cookbooks for run list: ["apache::default"]
...
Finished converging <default-centos-67> (0m27.64s) .
----> Kitchen is finished. (0m28.58s)
```

Whenever a change is made to a recipe or component of the cookbook it is important to converge the latest cookbook against the test instance.

If an error occurs that likely means that you have a typo within your default recipe or the install recipe.

Re-Verify the Test Instance



```
> kitchen verify
```

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

Target:  ssh://kitchen@localhost:32770

  ✓ Port 80 should be listening
  ✓ Command curl localhost stdout should match /Welcome Home/

Summary: 2 successful, 0 failures, 0 skipped
```

If everything converges successfully it is time to verify the state of the instance with the test that we have defined.

EXERCISE



Refactor to Modular Recipes

*This is why we **can** have nice things!*

Objective:

- ✓ Refactor the installation into a separate recipe
- ✓ Converge the cookbook and execute the tests

Nice shave!



Together we were able to refactor the cookbook while implementing the installation recipe.

LAB



The Configuration

- ☐ Create a configuration recipe that defines the policy:

The file named `'/var/www/html/index.html'` contains the content `'<h1>Welcome Home!</h1>'`

- ☐ Delete the automatically generated InSpec test
- ☐ Within the default recipe replace the file resource with an include recipe
- ☐ Converge and verify the test instance to ensure there are no failures

Now it is your turn to do the same thing for the webserver configuration. The only configuration that we currently perform for the webserver is write out a new default home page. We still want to move that resource to a separate recipe and ensure that we made the change correctly.

When you are done we will review the next few slides together to review your work.

Instructor Note: Another exercise follows this one to manage the service.

Instructor Note: Allow 5 minutes to complete this exercise.

Generate a Service Recipe



```
> chef generate recipe configuration
```

```
Recipe: code_generator::recipe
  * directory[/home/chef/apache/spec/unit/recipes] action create
    (up to date)
  * cookbook_file[/home/chef/apache/spec/spec_helper.rb] action
    create_if_missing (up to date)
  *
  template[/home/chef/apache/spec/unit/recipes/configuration_spec.rb
] action create_if_missing
    - create new file
      /home/chef/apache/spec/unit/recipes/configuration_spec.rb
    - update content in file
      /home/chef/apache/spec/unit/recipes/configuration_spec.rb from
```

Generate the configuration recipe within the webserver cookbook.

Remove the Generated Test File



```
> rm test/smoke/default/configuration.rb
```

Remove the automatically generated test file as we are not interested in the sample tests that it generates for us.

Write the Configuration Recipe



```
~/apache/recipes/configuration.rb
```

```
#  
# Cookbook Name:: apache  
# Recipe:: configuration  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
file '/var/www/html/index.html' do  
  content '<h1>Welcome Home!</h1>'  
end
```

Define all the resources that are related to the configuration of the webserver within this new recipe

Remove the Resource from the Default Recipe



~/apache/recipes/default.rb

```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
include_recipe 'apache::install'

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

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

Remove the resources, that are now defined in the configuration recipe, from the default recipe

Include the Configuration Recipe



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

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
include_recipe 'apache::install'  
include_recipe 'apache::configuration'  
  
service 'httpd' do  
  action [:enable, :start]  
end
```

Replace the resources that you have removed with an 'include_recipe' that brings the newly defined configuration recipe.

Re-Converge the Test Instance



```
> kitchen converge
```

```
----> Starting Kitchen (v1.11.1)
----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
...
----> Installing Chef Omnibus (install only if missing)
Downloading https://www.chef.io/chef/install.sh to file...
resolving cookbooks for run list: ["apache::default"]
...
Finished converging <default-centos-67> (0m27.64s) .
----> Kitchen is finished. (0m28.58s)
```

The recipe changed so it is important to converge the instance.

Re-Verify the Test Instance



```
> kitchen verify
```

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

Target:  ssh://kitchen@localhost:32770

  ✓ Port 80 should be listening
  ✓ Command curl localhost stdout should match /Welcome Home/

Summary: 2 successful, 0 failures, 0 skipped
```

If everything converges successfully it is time to verify the state of the instance with the test that we have defined.

LAB



The Configuration

- ✓ Create a configuration recipe that defines the policy:

The file named `'/var/www/html/index.html'` contains the content `'<h1>Welcome Home!</h1>'`

- ✓ Delete the automatically generated InSpec test
- ✓ Within the default recipe replace the file resource with an include recipe
- ✓ Converge and verify the test instance to ensure there are no failures

Congratulations you have successfully refactored the webserver configuration into its own recipe.

LAB



The Service

- ☐ Create a service recipe that defines the policy:

The service named 'httpd' is started and enabled

- ☐ Delete the automatically generated InSpec test
- ☐ Within the default recipe replace the service resource with an include recipe
- ☐ Converge and verify the test instance to ensure there are no failures

One last time!



Now it is your turn to do the same thing for the webserver service.

When you are done we will review the next few slides together to review your work.

Instructor Note: Allow 5 minutes to complete this exercise.

Generate a Service Recipe



```
> chef generate recipe service
```

```
Recipe: code_generator::recipe
  * directory[/home/chef/apache/spec/unit/recipes] action create
    (up to date)
  * cookbook_file[/home/chef/apache/spec/spec_helper.rb] action
    create_if_missing (up to date)
  * template[/home/chef/apache/spec/unit/recipes/service_spec.rb]
    action create_if_missing
    - create new file
      /home/chef/apache/spec/unit/recipes/service_spec.rb
    - update content in file
      /home/chef/apache/spec/unit/recipes/service_spec.rb from none to
      1f669c
```

Generate the service recipe within the webserver cookbook.

Remove the Generated Test File



```
> rm test/smoke/default/service.rb
```

Remove the automatically generated test file as we are not interested in the sample tests that it generates for us.

Write the Services Recipe



```
~/apache/recipes/service.rb
```

```
#  
# Cookbook Name:: apache  
# Recipe:: service  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
service 'httpd' do  
  action [:enable, :start]  
end
```

Define all the resources that are related to the service of the webserver within this new recipe

Remove the Resource from the Default Recipe



~/apache/recipes/default.rb

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
include_recipe 'apache::install'  
include_recipe 'apache::configuration'  
  
service 'httpd' do  
  action [:enable, :start]  
end
```

Remove the resources, that are now defined in the service recipe, from the default recipe

Remove the Resource from the Default Recipe



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

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
include_recipe 'apache::install'  
include_recipe 'apache::configuration'  
include_recipe 'apache::service'
```

Replace the resources that you have removed with an 'include_recipe' that brings the newly defined service recipe.

Re-Converge the Test Instance



```
> kitchen converge
```

```
----> Starting Kitchen (v1.11.1)
----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
...
----> Installing Chef Omnibus (install only if missing)
Downloading https://www.chef.io/chef/install.sh to file...
resolving cookbooks for run list: ["apache::default"]
...
Finished converging <default-centos-67> (0m27.64s) .
----> Kitchen is finished. (0m28.58s)
```

The recipe changed so it is important to converge the instance.

Re-Verify the Test Instance



```
> kitchen verify
```

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

Target:  ssh://kitchen@localhost:32770

  ✓ Port 80 should be listening
  ✓ Command curl localhost stdout should match /Welcome Home/

Summary: 2 successful, 0 failures, 0 skipped
```

If everything converges successfully it is time to verify the state of the instance with the test that we have defined.

LAB



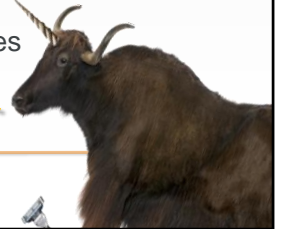
The Service

- ✓ Create a service recipe that defines the policy:

The service named 'httpd' is started and enabled.

- ✓ Delete the automatically generated InSpec test
- ✓ Within the default recipe replace the service resource with an include recipe
- ✓ Converge and verify the test instance to ensure there are no failures

My hair will grow back!



Congratulations you have successfully refactored the webserver service into its own recipe.

DISCUSSION



Do Our Tests Really Work?

What if we removed code from within the recipes and ran the tests?

During the group exercise and the lab we made changes to the recipes that we were able to verify on the test instance. If you accidentally or purposefully created a typo for yourself you would have seen the converge or the verification fail. However, what if removed code from the recipes that we wrote?

The omission (or in this case removal of code) of resources could have happened. When we refactored the default recipe we may have remembered to remove the resources that manage the configuration but forgot to use the 'include_recipe' to ensure we loaded the new recipe. Or it is possible that we created a service recipe that we never populated but made all the appropriate changes to the default recipe.

CONCEPT



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.

Removing code sabotages the policy that you have defined. If you used Test Kitchen to converge and verify the cookbook and saw a failure you can sleep soundly at night knowing your tools have you covered. On the other hand, if Test Kitchen were to return success, after such a change, then it might cause you to break out in a cold sweat.

Removing code from a recipe or recipes is a small change. So is introducing a typo into the code, specifying a different resource name or changing the value of a resource attribute. The process of modifying the code in small ways and then executing the test suite against it is often times referred to as mutation testing.

EXERCISE



Heckle That Code

It could be a game show. Maybe on Twitch?

Objective:

- ☐ Remove / Comment source code
- ☐ Converge the cookbook and execute the tests

Before we leave this module, let's do a little mutation testing, to ensure the test that we have defined is good enough.

Comment Out Key Code Within the Default Recipe



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

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
# include_recipe 'apache::install'  
include_recipe 'apache::configuration'  
include_recipe 'apache::service'
```

Return to the default recipe and choose one line to remove or comment out. Here I have chosen to comment out the first line that includes the install recipe.

EXERCISE



Heckle That Code

It could be a game show. Maybe on Twitch?

Objective:

- ✓ Remove / Comment source code
- ☐ Converge the cookbook and execute the tests

Now with that small mutation in place it is time to converge the cookbook and execute the tests.

Re-Converge the Test Instance



```
> kitchen converge
```

```
-----> Converging <default-centos-67>...  
Synchronizing Cookbooks:  
  - apache (0.1.0)  
Compiling Cookbooks...  
Converging 3 resources  
Recipe: apache::configuration  
  (up to date)  
Recipe: apache::service  
  (up to date)  
    * service[apache] action enable (up to date)
```

When converging the updated recipe it no longer shows the install recipe being loaded. This has changed the number of resources that are converged on the test instance. Removing the recipe from the default recipe does not remove any of the components that it previously installed.

Re-Verify the Test Instance



```
> kitchen verify
```

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

Target:  ssh://kitchen@localhost:32770

  ✓ Port 80 should be listening
  ✓ Command curl localhost stdout should match /Welcome Home/

Summary: 2 successful, 0 failures, 0 skipped
```

Verification of the test instance will return a success. Despite removing the install recipe from the default recipe the test instance is still able to serving the default web page that our test is looking for when it requests data from the site.

CONCEPT



Kitchen Converge & Verify

Running converge or verify will create a new instance the first time it is run. The same instance is used for each additional converge or verify.

The test instance policy changed, but no resource explicitly removed or uninstalled the resources defined in the install recipe.

This is important feature and limitation of using Test Kitchen's 'converge' and 'verify'. Both of these commands will create a test instance the first time they are executed. Every time after these commands will use the same test instance again and again.

When we remove resources from a recipe we do not explicitly uninstall them from the test instance. We simply do not enforce their policy anymore. On an existing system, which this test instance is after the first run, this means it is actually in the desired state that we no longer define. That means that the webserver is still installed, the default web page has still been updated, and the service is still running.

To ensure our cookbook works on a new system it is important to delete the test instance and start over.



CONCEPT

Kitchen Destroy

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


Destroys one or more instances.



Test Kitchen provides the 'destroy' subcommand. Destroy is available at all stages and essentially cleans up the instance. This is useful when you make changes to the configuration policy you define and you want to ensure that it will work on a brand new instance.

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


CONCEPT




Kitchen Test

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

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



©2017 Chef Software Inc. 4-45



Test Kitchen also provides the subcommand 'test'. Test provides one command that wraps up all the stages in one command. It will destroy any test instance that exists at the start, create a new one, converge the run list on that instance, and the verify it. If everything passes the 'test' subcommand will finish by destroying that instance. If it fails at one of these steps it usually leaves the instance running to allow you to troubleshoot it.

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

CONCEPT



Kitchen Test

Destroying the instance ensures that the policy is being applied to a new instance.

The test instance is re-created and then the updated policy is applied to the new instance. The new policy is incomplete causing an error.

Running 'kitchen test' is useful if want to ensure the policy you defined works on a new instance.

Test the Cookbook Against a New Instance




```
> kitchen test
```

```
----> Starting Kitchen (v1.11.1)
----> Cleaning up any prior instances of <default-centos-67>
----> Destroying <default-centos-67>...
...
[2017-08-29T20:29:16+00:00] FATAL:
Chef::Exceptions::ChildConvergeError: Chef run process exited
unsuccessfully (exit code 1)
>>>>> -----Exception-----
>>>>> Class: Kitchen::ActionFailed
>>>>> Message: 1 actions failed.
>>>>> Converge failed on instance <default-centos-67>.
```

Running 'kitchen test' in this instance will expose the issue that we created by removing that installation of the webserver. This is because the new instance no longer installed the necessary packages so the file path was never created for the default HTML file and there are no services to run.

The test that you wrote correctly verifies the state of the system. What is important to notice is that there are important differences in the Test Kitchen commands.

Converge & Verify	Test
<p>Faster execution time</p> <p>Running converge twice will ensure your policy applies without error to existing instances</p>	<p>Slower execution time</p> <p>Running test will ensure your policy applies without error to any new instances</p>
©2017 Chef Software Inc.	<div><div>4-48</div><div>CHEF</div></div>

Using Test Kitchen to run 'kitchen converge' and 'kitchen verify' is much faster because you are essentially applying and verifying the policy that you have defined against an already running instance. The drawback is that only running 'converge' and 'verify' will not demonstrate for you how your policy will act on a brand new instance.

Using Test Kitchen to run 'kitchen test' is slower because every time you are recreating the test instance, installing chef, and applying the policy on that new instance. The drawback here is the longer feedback cycle and only running 'test' will not demonstrate for you how your policy will act on an existing instance.

EXERCISE



Heckle That Code

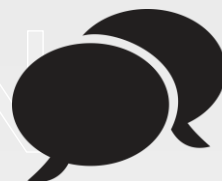
It could be a game show. Maybe on Twitch?

Objective:

- ✓ Remove / Comment source code
- ✓ Converge the cookbook and execute the tests

Removing code and causing a failure showed us some of the differences between 'kitchen converge and verify' and 'kitchen test'. To ensure that we understand these important differences let's have a discussion.

DISCUSSION



Discussion

What is happening when running `kitchen test`?


What types of bugs would `kitchen converge` & `kitchen verify` find when running?

What is the difference between `kitchen test` and running both `kitchen converge` & `kitchen verify` together?

How long do each of these approaches take?

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

DISCUSSION




Q&A


What questions can we answer for you?

©2017 Chef Software Inc.

4-51



Before we complete this section, let us pause for questions.

Morning	Afternoon
<div>Introduction</div> <div>Why Write Tests? Why is that Hard?</div> <div>Writing a Test First</div> <div>Refactoring Cookbooks with Tests</div>	<div>Faster Feedback with Unit Testing</div> <div>Testing Resources in Recipes</div> <div>Refactoring to Attributes</div> <div>Refactoring to Multiple Platforms</div>
<div>©2017 Chef Software Inc.</div>	<div>4-52</div> <div>CHEF</div>

You have performed the complete TDD cycle from start to finish. Now that you have seen this cycle and understand that we are simply going to continue to repeat it as we develop cookbooks it is important to talk about the amount of time it takes for us to get feedback. In the next section we are going to explore that further by introducing a new testing tool and language that promises to give us faster feedback.

