

5: Testing Cookbooks



Slide 2

Objectives




After completing this module, you should be able to

- Use Test Kitchen to verify your recipes converge on a virtual instance
- Read the InSpec documentation
- Write and execute tests


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

Slide 3



Can We Test Cookbooks?

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

©2015 Chef Software Inc. 5-3 


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

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

So how could we solve a problem like this?


Slide 4

DISCUSSION



Mandating Testing

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

©2015 Chef Software Inc. 5-4 


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

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

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

Slide 5

Steps to Verify Cookbooks



```
graph TD; A[Create Virtual Machine] --> B[Install Chef Tools]; B --> C[Copy Cookbooks]; C --> D[Run/Apply Cookbooks]; D --> E[Verify Assumptions]; E --> F[Destroy Virtual Machine];
```

©2015 Chef Software Inc. 5-5



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

Create a virtual machine or setup an instance that resembles your current production infrastructure

Install the necessary Chef tools

Copy the cookbooks to this new instance

Apply the cookbooks to the instance

Verify that the instance is the desired state by executing various commands

Clean up that instance by destroying it or rolling it back to a previous snapshot

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

Slide 6

Testing Cookbooks



We can start by first mandating that all cookbooks are tested

How often should you test your cookbook?

How often do you think changes will occur?


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

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

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


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

Slide 7



Code Testing


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

©2015 Chef Software Inc. 5-7 

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

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

Slide 8




Test Configuration

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

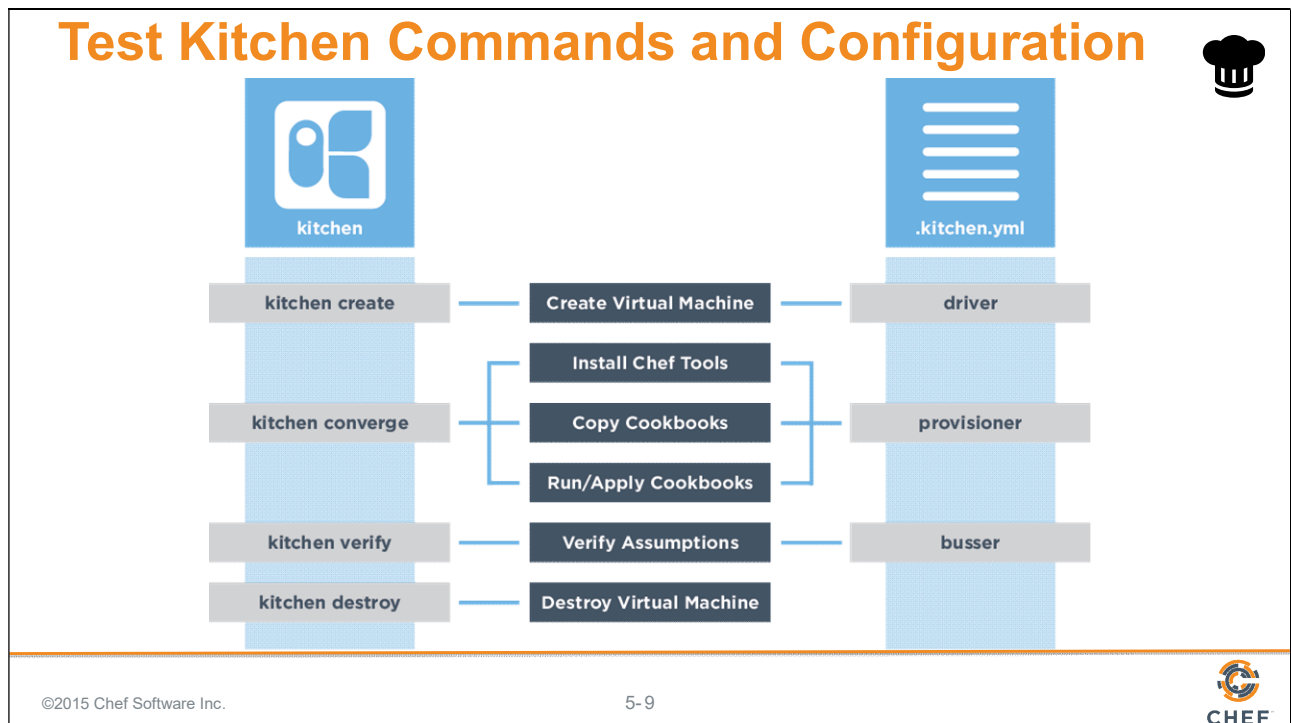
Objective:

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

©2015 Chef Software Inc. 5-8 

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

Slide 9



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

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

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

These commands the configuration will be explained in more detail.

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

What Can 'kitchen' Do?



```
$ kitchen --help
```

Commands:

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

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

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

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

What Can 'kitchen init' Do?



```
$ kitchen help init
```

Usage:

```
kitchen init
-D, [--driver=one two three]      # One or more Kitchen Driver gems ...
                                   # Default: kitchen-vagrant
-P, [--provisioner=PROVISIONER]  # The default Kitchen Provisioner to
use                               # Default: chef_solo
                                   # Whether or not to create a Gemfile
[--create-gemfile], [--no-create-gemfile]
```

Description:

Init will add Test Kitchen support to an existing project for convergence integration testing. A default `.kitchen.yml` file (which is intended to be customized) is created in the project's root directory and one or more gems will be added to the project's Gemfile.

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

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

Do We Have a .kitchen.yml?



```
$ tree /f cookbooks\workstation
```

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

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

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

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

What is Inside .kitchen.yml?



```
$ gc cookbooks\iis-demo\.kitchen.yml
```

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

# Uncomment the following verifier to leverage Inspec instead of Busser (the
# default verifier)
# verifier:
#   name: inspec

platforms:
```

The .kitchen.yml file defines a number of configuration entries that the kitchen command uses during execution.

Slide 14



.kitchen.yml

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

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

©2015 Chef Software Inc. 5-14 

We don't need to run `kitchen init` because we already have a default kitchen file. We may still need to update it to accomplish our objectives so let's learn more about the various fields in the configuration file.

Demo: The kitchen Driver

~\cookbooks\workstation\.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

# KITCHEN CONFIGURATION CONTINUES...
```

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

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

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

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

Demo: The kitchen Provisioner

 ~\cookbooks\workstation\.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

# KITCHEN CONFIGURATION CONTINUES...
```

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

The default and simplest approach is to use chef_zero.

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

Slide 17

Demo: The kitchen Platforms

 ~\cookbooks\workstation\.kitchen.yml

```
# TOP PART OF KITCHEN CONFIGURATION
```

```
platforms:
```

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

```
suites:
```

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

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

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

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

Demo: The kitchen Suites

 ~\cookbooks\workstation\.kitchen.yml

```
# TOP PART OF KITCHEN CONFIGURATION
```

```
platforms:
```

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

```
suites:
```

```
- name: default
```

```
  run_list:
```

- recipe[workstation::default]

```
  attributes:
```

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

We define a single suite named "default".

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

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

Demo: The kitchen Suites

 ~\cookbooks\workstation\.kitchen.yml

```
# TOP PART OF KITCHEN CONFIGURATION

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

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

The suite named "default" defines a run_list.

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

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



Kitchen Test Matrix

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

PLATFORMS x SUITES

Running kitchen list will show that matrix.

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

Slide 21

Example: View the Kitchen Test Matrix

```
$ kitchen list
```

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

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

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

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

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

Slide 22

Example: View the Kitchen Test Matrix

```
$ kitchen list
```

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

```
suites:
```

```
- name: default
```

```
  run_list:
```

```
    - recipe[workstation::default]
```

```
  attributes:
```

```
platforms:
```

```
- name: ubuntu-12.04
```

```
- name: centos-6.5
```

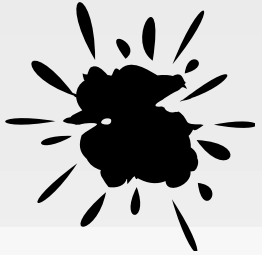
And the second centos 6.5 platform.

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

Slide 23


PROBLEM

Virtualization



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

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

©2015 Chef Software Inc. 4-23 

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

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

Slide 24

GE: Replace the Kitchen Config



```
$ cp kitchen-template.yml cookbooks\workstation\.kitchen.yml
```

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

GE: Add Your Name and Company

```
~\cookbooks\workstation\.kitchen.yml

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

Edit the file to insert your name and company information.

GE: Update to Test the Workstation Cookbook

 ~\cookbooks\workstation\.kitchen.yml

```
# ... TOP PART OF THE CONFIGURATION FILE ...

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

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

Slide 27

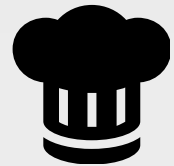
GE: Look at the Test Matrix



```
$ kitchen list
```

Instance	Driver	Provisioner	Verifier	Transport	Last Action
default-windows-2012r2	Ec2	ChefZero	Busser	Winrm	Converged

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



Test Configuration

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

Objective:

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

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

Slide 29



Kitchen Create



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

Create one or more instances.

©2015 Chef Software Inc. 5-29 CHEF

The first kitchen command is `kitchen create`.

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

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

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



Group Exercise: Kitchen Converge



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

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

©2015 Chef Software Inc. 5-30 

Creating an image gives us an instance to test our cookbooks but it still would leave us with the work of installing chef and applying the cookbook defined in our `.kitchen.yml` run list.

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

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

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

Instructor Note: It also, like the `'kitchen create'` commands, defaults to all instances when executed without any parameters. And is capable of accepting parameters to converge a specific instance or all instances that match the provided criteria.

GE: Converge the Cookbook



```
$ kitchen converge
```


```
-----> Starting Kitchen (v1.4.2)
-----> Creating <default-windows-2012r2>...
    Instance <i-1be58ae2> requested.
    EC2 instance <i-1be58ae2> created.
    Waited 0/300s for instance <i-1be58ae2> to become ready.      Waited
5/300s for instance <i-1be58ae2> to become ready.
    ...
    Waited 280/300s for instance <i-1be58ae2> to become ready.
    EC2 instance <i-1be58ae2> ready.
    [WinRM] Established
    ...
```

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

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

Slide 32


GE: Converge the Cookbook



```
$ kitchen login
```

The terminal window shows the command `$ kitchen login` at the prompt. Below the command is a large black rectangular area, likely representing a redacted output or a placeholder for a screenshot.

©2015 Chef Software Inc. 5-32



We can also login to the instance through the command ``kitchen login``. This is useful to see the state of the system to ensure that UAC has been properly disabled.

Slide 33



Test Configuration

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

Objective:

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

©2015 Chef Software Inc. 5-33 

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



Lab: Converge the Recipe for iis-demo

- ☐ Copy the template kitchen configuration into the 'iis-demo' cookbook
- ☐ Update the configuration to specify username and cookbook name
- ☐ Converge the 'iis-demo' cookbook

Do the same thing again for the 'iis-demo' cookbook. Update the .kitchen.yml file so that it converges the iis-demo cookbook's default recipe on a Windows 2012R2 instance through the EC2 driver.

Instructor Note: Allow 12 minutes to complete this exercise. Most of the time for this exercise is likely going to be spent waiting for the instance in Amazon EC2 to be ready.

Slide 35

Lab: Return the Home Directory



```
$ cd ~
```

First, return to the home directory.

Lab: Replace the Kitchen Config



```
$ cp kitchen-template.yml cookbooks/iis-demo/.kitchen.yml
```

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

Slide 37

Lab: Add Your Name and Company



```
~\cookbooks\iis-demo\.kitchen.yml
```

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

Edit the file to insert your name and company information.

Lab: Update to Test the 'iis-demo' Cookbook

```
~\cookbooks\iis-demo\.kitchen.yml
```

```
# ... TOP PART OF THE CONFIGURATION FILE ...

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

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

Lab: Return Home and Move into the Cookbook



```
$ cd cookbooks\iis-demo
```

Change into the 'iis-demo' cookbook folder.

Lab: Converge the Cookbook



```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.4.2)
-----> Creating <default-windows-2012r2>...
      Instance <i-1be58ae2> requested.
      EC2 instance <i-1be58ae2> created.
      Waited 0/300s for instance <i-1be58ae2> to become ready.      Waited
5/300s for instance <i-1be58ae2> to become ready.
      ...
      Waited 280/300s for instance <i-1be58ae2> to become ready.
      EC2 instance <i-1be58ae2> ready.
      [WinRM] Established
      ...
```

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




Lab: Converge the Recipe for iis-demo

- ✓ Copy the template kitchen configuration into the 'iis-demo' cookbook
- ✓ Update the configuration to specify username and cookbook name
- ✓ Converge the 'iis-demo' cookbook

Congratulations you successfully converged the 'iis-demo' cookbook on a test instance that you created in Amazon EC2.

Slide 42

DISCUSSION




Test Kitchen

What does this test when kitchen converges a recipe?

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

©2015 Chef Software Inc.

5-42


CHEF


So what does this test when kitchen converges a recipe?

What does it NOT test when kitchen converges a recipe?

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

Slide 43

DISCUSSION




Test Kitchen

What is left to validate to ensure that the cookbook successfully applied the policy defined in the recipe?

©2015 Chef Software Inc.


5-43



What is left to validate to ensure that the cookbook successfully applied the policy defined in the recipe?

Instructor Note: Converging the instance ensured that the recipe was able to install a package, write out a file, and start and enable a service. But what it was unable to check to see if the system was configured correctly -- is our instance serving up our custom home page.

Slide 44




The First Test

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

Objective:

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


©2015 Chef Software Inc. 5-44 

There is no automation that automatically understands the intention defined in the recipes we create. To do that we will define our own automated test.

Let's explore testing by adding an expectation that will validate that UAC has been properly disabled on our test instance.

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


Slide 45



InSpec

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

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

©2015 Chef Software Inc. 5-45 

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

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

Slide 46

Example: Is the Registry Key Set Correctly?

```
system_policies = 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System'  
  
describe registry_key('System Policies', system_policies) do  
  its('EnableLUA') { should eq 0 }  
end
```

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

https://docs.chef.io/inspec_reference.html#registry-key

Here is an InSpec example.

We define a local variable named 'system_policies' that will store the registry key that stores our particular key we want to verify.

We use RSpec's 'describe' to begin to illustrate the element of our system that we are testing.

InSpec provides a helper method named 'registry_key' that allows to provide two parameters. The first parameter is a common name for the registry key. The second is the registry key itself which can be found inside the variable that we have defined.

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

Slide 47

GE: Return Home and Move into the Cookbook



```
$ cd ~\cookbooks\workstation
```

Change into the workstation cookbook directory.

Rename ServerSpec Test Directories



```
$ mv test\integration\default\serverspec  
test\integration\default\inspec
```

Chef Development Kit 0.10.0 assumes we want to use ServerSpec so it generated a number of folders with files in the test directory. ServerSpec is a similar verifier to InSpec.

To use InSpec we will need to manually rename the folder from 'serverspec' to to be called 'inspec'.

Instructor Note: This may not be necessary in future versions of the ChefDK that automatically assume InSpec as the Test Kitchen verifier.



Where do Tests Live?

```
workstation/test/integration/default/inspec/default_spec.rb
```

Test Kitchen will look for tests to run under this directory. It allows you to put unit or other tests in test/unit, spec, acceptance, or wherever without mixing them up. This is configurable, if desired.

<http://kitchen.ci/docs/getting-started/writing-test>

Let's take a moment to describe the reason behind this long directory path. Within our cookbook we define a test directory and within that test directory we define another directory named 'integration'. This is the basic file path that Test Kitchen expects to find the specifications that we have defined.



Where do Tests Live?


```
workstation/test/integration/default/inspec/default_spec.rb
```

This corresponds exactly to the Suite name we set up in the .kitchen.yml file. If we had a suite called "server-only", then you would put tests for the server only suite under

<http://kitchen.ci/docs/getting-started/writing-test>

The next part the path, 'default', corresponds to the name of the test suite that is defined in the .kitchen.yml file. In our case the name of the suite is 'default' so when test kitchen performs a `kitchen verify` for the default suite it will look within the 'default' folder for the specifications to run.

Slide 51




Where do Tests Live?

```
workstation/test/integration/default/inspec/default_spec.rb
```


This tells Test Kitchen (and Busser) which Busser runner plugin needs to be installed on the remote instance.

<http://kitchen.ci/docs/getting-started/writing-test>

©2015 Chef Software Inc. 5-51 

'inspec' is the kind of tests that we want to define. Test Kitchen supports a number of testing frameworks.

Slide 52




Where do Tests Live?

```
workstation/test/integration/default/inspec/default_spec.rb
```

All test files (or specs) are named after the recipe they test and end with the suffix "_spec.rb". A spec missing that will not be found when executing `kitchen verify`.

<http://kitchen.ci/docs/getting-started/writing-test>

©2015 Chef Software Inc. 5-52 

The final part of the path is the specification file. This is a ruby file. The naming convention for this file is the recipe name with the appended suffix of `_spec.rb`. All specification files must end with `_spec.rb`.

Instructor Note: Without the `'_spec.rb'` extension RSpec will simply ignore that file.

Slide 53

GE: Replace the Existing Test File


```
~\cookbooks\workstation\test\integration\default\inspec\default_spec.rb
```

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

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

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

Slide 54




The First Test

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


Objective:

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


©2015 Chef Software Inc. 5-54 

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

Slide 55




Kitchen Verify



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

Create, converge, and verify one or more instances.

©2015 Chef Software Inc. 5-55 

The third kitchen command is `kitchen verify`.

To verify an instance means to:

Create a virtual or cloud instances, if needed

Converge the instance, if needed

And then execute a collection of defined tests against the instance

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

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

Slide 56



The diagram illustrates the 'Kitchen Destroy' process. At the top right is the Chef logo, a black open box. Below it, the title 'Kitchen Destroy' is written in orange. A sequence of three teal boxes labeled 'kitchen create', 'kitchen converge', and 'kitchen verify' are connected by a grey arrow pointing right. Red arrows point from each of these boxes down to a red box labeled 'kitchen destroy'. Below the red box is the command `$ kitchen destroy [INSTANCE|REGEXP|all]`. Underneath the command is the text 'Destroys one or more instances.'

Kitchen Destroy

kitchen create kitchen converge kitchen verify

`kitchen destroy`

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

Destroys one or more instances.

©2015 Chef Software Inc. 5-56 

The fourth kitchen command is `kitchen destroy`.

Destroy is available at all stages and essentially cleans up the instance.

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

Slide 57



Kitchen Test




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

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

©2015 Chef Software Inc.

5-57



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

Kitchen test ensures that if the instance was in any state - created, converged, or verified - that it is immediately destroyed. This ensures a clean instance to perform all of the steps: create; converge; and verify. `kitchen test` completes the entire execution by destroying the instance at the end.

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

Slide 58

GE: Running the Test



```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.4.2)
-----> Verifying <default-windows-2012r2>...
.

Finished in 0.68749 seconds (files took 0.14062 seconds to load)
1 example, 0 failures

    Finished verifying <default-windows-2012r2> (0m0.73s).
-----> Kitchen is finished. (0m5.22s)
```

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



The First Test

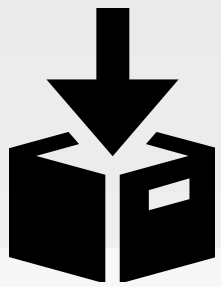
Converging seems to validate that the recipe runs successfully. But does it assert what actually is installed?

Objective:

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


Congratulations you wrote and executed your first test.

Slide 60



GE: Commit Your Work

```
$ cd ~\cookbooks\workstation  
$ git add .  
$ git status  
$ git commit -m "Added first test for the default  
recipe"
```

©2015 Chef Software Inc. 5-60 

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

Slide 61



Lab: Additional Test

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

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

Instructor Note: Allow 8 minutes to complete this exercise.

Slide 62

GE: Replace the Existing Test File

```
~\cookbooks\workstation\test\integration\default\inspec\default_spec.rb

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

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

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

Slide 63

GE: Running the Test



```
$ kitchen verify
```

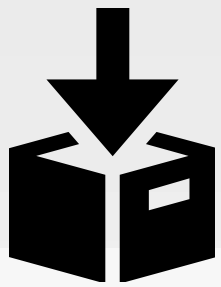
```
-----> Starting Kitchen (v1.4.2)
-----> Verifying <default-windows-2012r2>...
.

Finished in 0.68749 seconds (files took 0.14062 seconds to load)
2 example, 0 failures

    Finished verifying <default-windows-2012r2> (0m0.73s).
-----> Kitchen is finished. (0m5.22s)
```


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

Slide 64



GE: Commit Your Work

```
$ cd ~\cookbooks\workstation  
$ git add .  
$ git status  
$ git commit -m "Added second test for the default  
recipe"
```

©2015 Chef Software Inc. 5-64 

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

Slide 65



Lab: Additional Test


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

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

Instructor Note: Allow 8 minutes to complete this exercise.

Slide 66

DISCUSSION




Testing


What questions can we help you answer?

©2015 Chef Software Inc.

5-66



What questions can we help you answer?




Testing Our Webserver

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

Objective:


- ❑ Discuss and decide what should be tested with the iis-demo cookbook

©2015 Chef Software Inc. 5-67 

Now let's turn our focus towards testing the iis-demo cookbook.

Slide 68

DISCUSSION




Testing

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

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

©2015 Chef Software Inc.

5-68



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

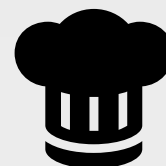
The iis-demo cookbook is similar to the workstation cookbook. It has a package and file which are things that we have already tested. The new thing is the service. We could review the InSpec documentation to find examples on how to test the service.

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

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

After applying the recipes in the past we visited the site through a browser or verified the content through running the command 'curl localhost'.

Is that something that we could test as well? Does InSpec provide the way for us to execute a command and verify the results?



Testing Our Webserver


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

Objective:

- ✓ Discuss and decide what should be tested with the iis-demo cookbook


Now that we have decided on the important things to test for our webserver. It is not time to write those tests.

Slide 70



Lab: Testing iis-demo

- ☐ Create a test file for the "iis-demo" cookbook's default recipe
- ☐ Add tests that validate a working web server
https://docs.chef.io/inspec_reference.html#port
https://docs.chef.io/inspec_reference.html#command
- ☐ Run kitchen verify
- ☐ Commit your changes

©2015 Chef Software Inc. 5-70 

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

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

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

Instructor Note: Allow 15 minutes to complete this exercise.

Lab: Change in the 'iis-demo' Cookbook Dir



```
$ cd ~\cookbooks\iis-demo  
$ mv test\integration\default\serverspec  
test\integration\default\inspec
```

Move into the iis-demo cookbook. Rename the 'serverspec' directory to be called 'inspec'

Slide 72

Lab: What Does the Webserver Say?



```
~\cookbooks\iis-demo\test\integration\default\inspect\default_spec.rb
```

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

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

Port 80 should be listening.

The standard out from the command 'curl http://localhost' should match 'Hello, world!'

Here we chose to validate that port 80 should be listening for incoming connections.

And we also validated that the standard out from the command 'curl http://localhost' should match 'Hello, world!'.

Lab: Verify the Webserver is Working



```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.4.2)
-----> Verifying <default-windows-2012r2>...
..

Finished in 2.17 seconds (files took 1.41 seconds to load)
2 examples, 0 failures

    Finished verifying <default-windows-2012r2> (0m3.49s).
-----> Kitchen is finished. (0m8.21s)
```

Slide 74

Lab: Commit Your Work

```
$ cd ~\cookbooks\iis-demo  
$ git add .  
$ git status  
$ git commit -m "Added tests for the default recipe"
```


©2015 Chef Software Inc.

5-74

Again, let's commit the work.

Slide 75

DISCUSSION



Discussion

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


Where would you define additional platforms?

Why would you define a new test suite?

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

©2015 Chef Software Inc.

5-75



Answer these questions.

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

Slide 76

DISCUSSION



Q&A

What questions can we help you answer?

- Test Kitchen
- kitchen commands
- kitchen configuration
- InSpec

©2015 Chef Software Inc.

5-76



What questions can we help you answer?

Generally or specifically about test kitchen, kitchen commands, kitchen configuration, InSpec.

Slide 77

