Testing Cookbooks

Validating Our Recipes in Virtual Environments

Objectives

After completing this module, you should be able to

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

Can We Test Cookbooks?

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

Mandating Testing

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

Steps to Verify Cookbooks

- Create Virtual Machine
- Install Chef Tools
- Copy Cookbooks
- Run/Apply Cookbooks
- Verify Assumptions
- Destroy Virtual Machine

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?

Code Testing

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

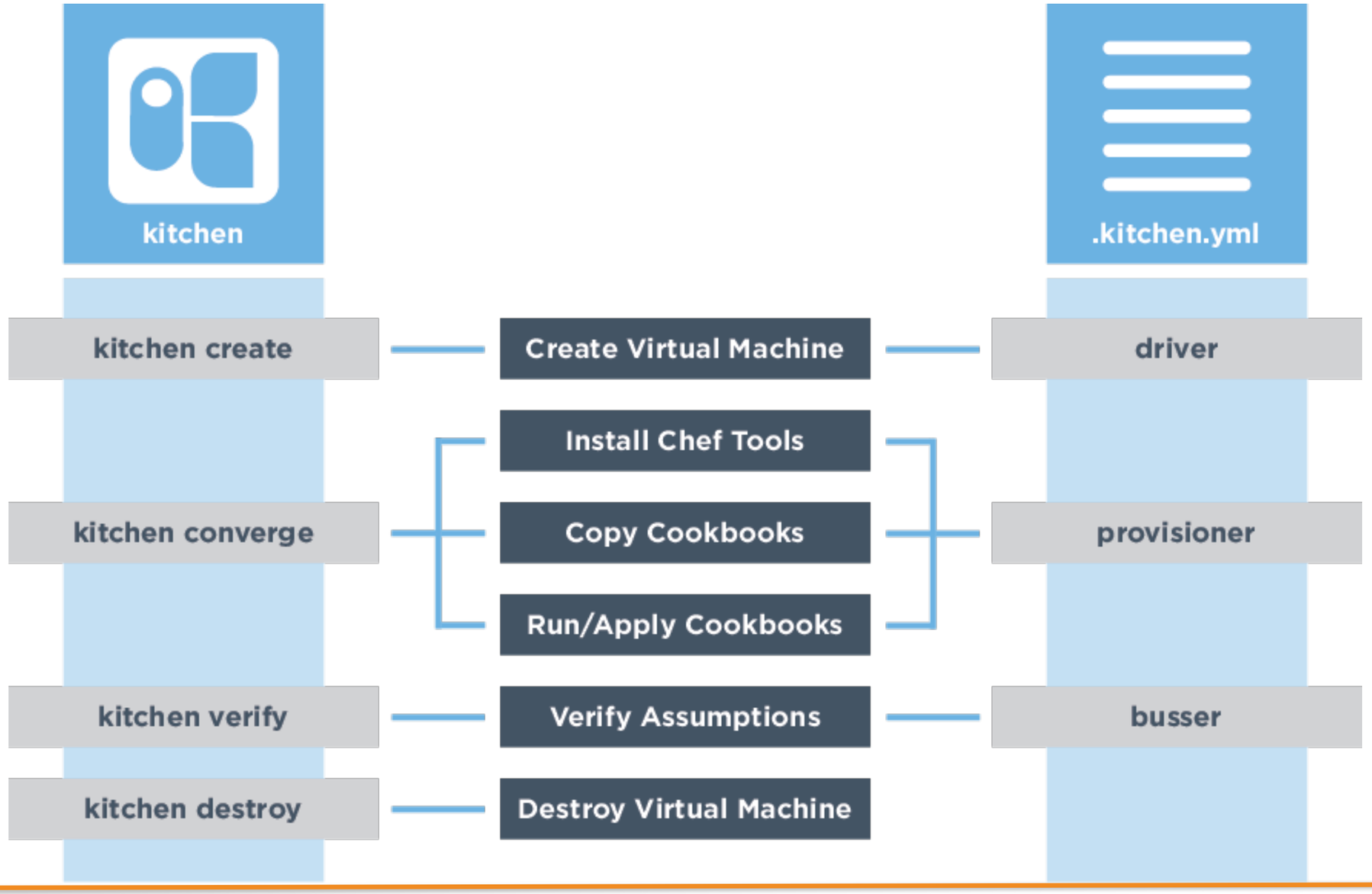
Test Configuration

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

Objective:

- ❑ Configure the "workstation" cookbook to test against the centos-6.7 platform
- ❑ Test the "workstation" cookbook on a virtual machine

Test Kitchen Commands and Configuration



CHEF

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
```

What Can 'kitchen init' Do?

```
$ kitchen help init
```

```
UsaGL:
  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
      [--create-gemfile], [--no-create-gemfile]     # Whether or not to create a Gemfi ...

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.
```

Do We Have a .kitchen.yml?

```
$ tree cookbooks/workstation -a -I .git
```

```
workstation
├── Berksfile
├── chefignore
├── .gitignore
├── .kitchen.yml
├── metadata.rb
├── README.md
├── recipes
│   ├── default.rb
│   └── setup.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
```

What is Inside .kitchen.yml?

```
$ cat cookbooks/workstation/.kitchen.yml
```

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

platforms:
  - name: ubuntu-12.04
  - name: centos-6.4

suites:
  - name: default
```

.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

Demo: The kitchen Driver

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

...
```

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

Example Drivers:

- docker
- vagrant

Demo: The kitchen Provisioner

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

...
```

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.

Demo: The kitchen Platforms

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

...
```

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

Demo: The kitchen Suites

~/cookbooks/workstation/.kitchen.yml

```
...

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".

Demo: The kitchen Suites

~/cookbooks/workstation/.kitchen.yml

```
...

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.

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.

Example: Kitchen Test Matrix

```
$ kitchen list

Instance                Driver   Provisioner  Verifier  Transport Last Action
default-ubuntu-1204     Vagrant  ChefZero     Busser    Ssh       <Not Created>
default-centos-65       Vagrant  ChefZero     Busser    Ssh       <Not Created>
```

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

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

Example: Kitchen Test Matrix

```
$ kitchen list

Instance              Driver   Provisioner  Verifier  Transport  Last Action
default-ubuntu-1204   Vagrant  ChefZero     Busser    Ssh        <Not Created>
default-centos-65     Vagrant  ChefZero     Busser    Ssh        <Not Created>
```

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

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

Group Exercise: Test Configuration

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

Objective:

- ❑ Configure the "workstation" cookbook's .kitchen.yml to use the Docker driver and centos 6.7 platform
- ❑ Use kitchen converge to apply the recipe on a virtual machine

GL: Move into the Cookbook's Directory

```
$ cd cookbooks/workstation
```

GL: Edit the Kitchen Configuration File

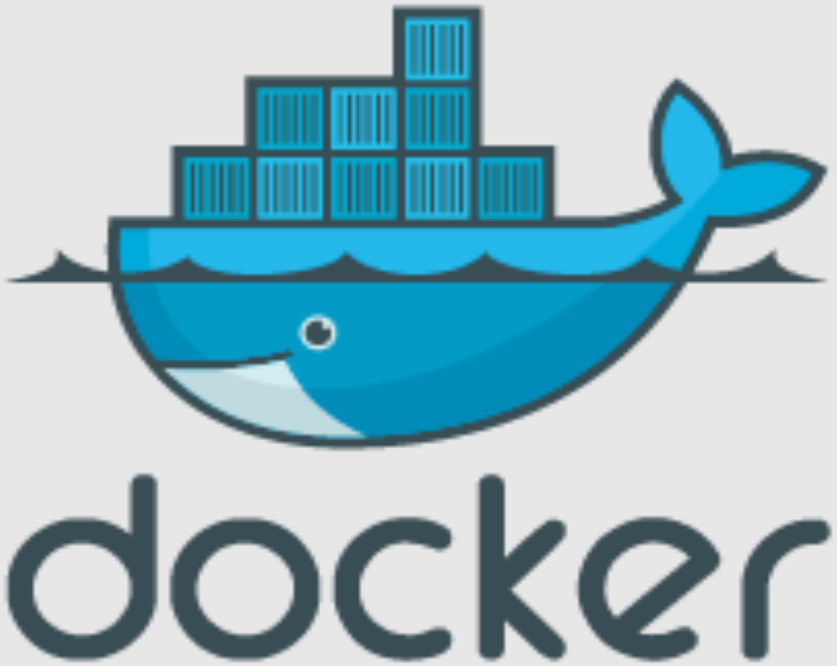
~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

platforms:
  - name: centos-6.7

suites:
# ... REMAINDER OF FILE ...
```



https://github.com/portertech/kitchen-docker

GL: Edit the Kitchen Configuration File

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

platforms:
  - name: centos-6.7

suites:
# ... REMAINDER OF FILE ...
```

https://www.centos.org

GL: Look at the Test Matrix

```
$ kitchen list
```

Instance	Driver	Provisioner	Verifier	Transport	Last Action
default-centos-67	Docker	ChefZero	Busser	Ssh	<Not Created>

Converging a Cookbook

Before I add features it really would be nice to test these cookbooks against the environments that resemble production.

Objective:

- ✓ Configure the "workstation" cookbook's .kitchen.yml to use the Docker driver and centos-6.7 platform
- ❑ Use kitchen converge to apply the recipe on a virtual machine

Kitchen Create


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

Create one or more instances.
```


CHEF

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.

GL: Converge the Cookbook

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.4.0)
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  2.56 kB
(skipping)
----->  Finished creating <default-centos-67> (1m18.32s).
-----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
(skipping)
Synchronizing Cookbooks:
         - workstation
       Compiling Cookbooks...
       Converging 0 resources
       Running handlers:
```

Lab: Converge the Recipe for Apache

We want to validate that our run-list installs correctly.

- ❑ Within the "apache" cookbook use kitchen converge for the default suite on the centos 6.7 platform.

Lab: Configuring Test Kitchen for Apache

~/cookbooks/apache/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

platforms:
  - name: centos-6.7

suites:
  - name: default
    run_list:
```

Lab: Return Home and Move into the Cookbook

```
$ cd ~/cookbooks/apache
```

Lab: Converge the Cookbook

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.4.0)
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  2.56 kB
       Sending build context to Docker daemon
(skipping)
 Installing Chef
       installing with rpm...
       warning: /tmp/install.sh.23/chef-12.4.1-1.el6.x86_64.rpm: Header V4 DSA/
SHA1 Signature, key ID 83ef826a: NOKEY
(skipping)
 Synchronizing Cookbooks:
         - apache
       Compiling Cookbooks...
```

Lab: Converge the Recipe for Apache

We want to validate that our run-list installs correctly.

- ✓ Within the "apache" cookbook use kitchen converge for the default suite on the centos 6.7 platform.

Test Kitchen

What is being tested when kitchen converges a recipe without error?

What is NOT being tested when kitchen converges the recipe without error?

Test Kitchen

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

The First Test

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

Objective:

- In a few minutes we'll write and execute a test that asserts that the tree package is installed when the "workstation" cookbook's default recipe is applied.

Kitchen Verify


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

Create, converge, and verify one or more instances.

Kitchen Destroy


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

Destroys one or more instances.

CHEF

Kitchen Test


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

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

ServerSpec

Serverspec tests your servers' actual state by executing command locally, via SSH, via WinRM, via Docker API and so on.

So you don't need to install any agent software on your servers and can use any configuration management tools, Puppet, Chef, CFEngine, Itamae and so on.

http://serverspec.org

Example: Is the 'tree' Package Installed?

```
describe package('tree') do
  it { should be_installed }
end
```

I expect the package tree should be installed.

http://serverspec.org/resource_types.html#package

GL: Requiring a Test Helper

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require 'spec_helper'

describe 'workstation::default' do

  describe package('tree') do
    it { should be_installed }
  end

end
```

Loads a helper file with that name in the same directory.

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

GL: Describing the Test Context

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require 'spec_helper'

describe 'workstation::default' do

  describe package('tree') do
    it { should be_installed }
  end

end
```

Describing a body of tests for the 'workstation' cookbook's default recipe.

https://relishapp.com/rspec/rspec-core/v/3-3/docs

GL: Our Assertion in a spec File

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require 'spec_helper'

describe 'workstation::default' do

  describe package('tree') do
    it { should be_installed }
  end

end
```

When we converge the workstation cookbook's default recipe we expect the package named tree to be installed.

http://serverspec.org/resource_types.html#package

Where do Tests Live?

`workstation/test/integration/default/serverspec/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

Where do Tests Live?

`workstation/test/integration/default/serverspec/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

Where do Tests Live?

`workstation/test/integration/default/serverspec/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

Where do Tests Live?

`workstation/test/integration/default/serverspec/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

GL: Return Home and Move into the Cookbook

```
$ cd ~/cookbooks/workstation
```

GL: Running the Specification

```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.4.0)
-----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
(skipping)
-----> Chef Omnibus installation detected (install only if missing)
       Transferring files to <default-centos-67>
       Starting Chef Client, version 12.4.1
(skipping)
       Running handlers:
       Running handlers complete
       Chef Client finished, 6/6 resources updated in 64.426896317 seconds
       Finished converging <default-centos-67> (1m9.02s).
-----> Kitchen is finished. (1m9.69s)
```

GL: Commit Your Work

```
$ cd ~/cookbooks/workstation
$ git add .
$ git status
$ git commit -m "Add first test for default test suite"
```

DISCUSSION

More Tests

What are other resources within the recipe that we could test?

Testing a File

ServerSpec can help us assert different characteristics about files on the file system. Like if it is a file, directory, socket or symlink.

The file's mode owner or group. If the file is readable, writeable, or executable. It is even able to verify the data contained within the file.

http://serverspec.org/resource_types.html#file

CHEF

Example: The File Contains Data

```
describe file('/etc/passwd') do
  it { should be_file }
end
```

I expect the file named '/etc/passwd' to be a file (as opposed to a directory, socket or symlink).

http://serverspec.org/resource_types.html#file

Example: The File Contains Specific Content

```
describe file('/etc/httpd/conf/httpd.conf') do
  its(:content) { should match /ServerName www.example.jp/ }
end
```

I expect the file named '/etc/httpd/conf/httpd.conf' to have content that matches 'ServerName www.example.jp'

http://serverspec.org/resource_types.html#file

Example: The File is Owned by a Particular User

```
describe file('/etc/sudoers') do
  it { should be_owned_by 'root' }
end
```

I expect the file named '/etc/sudoers' to be owned by the 'root' user.

Lab: More Tests

- ❑ Add tests that validate that the remaining package resources have been installed (http://serverspec.org/resource_types.html#package)
- ❑ Add tests that validate the file resource (http://serverspec.org/resource_types.html#file)
- ❑ Run kitchen verify to validate the test meets the expectations that you defined
- ❑ Commit your changes

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
require 'spec_helper'

describe 'workstation::default' do
  # ... other tests for packages ...

  describe package('tree') do
    it { should be_installed }
  end

  describe package('git') do
    it { should be_installed }
  end

end
```

The package named 'git' is installed.

http://serverspec.org/resource_types.html#package

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/integration/default/serverspec/default_spec.rb

```
...

  describe package('git') do
    it { should be_installed }
  end

  describe file('/etc/motd') do
    it { should be_owned_by 'root' }
  end

end
```

The file named '/etc/motd' should be owned by 'root'.

http://serverspec.org/resource_types.html#file

GL: Return to the Cookbook Directory

```
$ cd ~/cookbooks/workstation
```

Lab: Running the Specification

```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.4.0)
-----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
(skipping)
-----> Chef Omnibus installation detected (install only if missing)
       Transferring files to <default-centos-67>
       Starting Chef Client, version 12.4.1
(skipping)
       Running handlers:
       Running handlers complete
       Chef Client finished, 6/6 resources updated in 64.426896317 seconds
       Finished converging <default-centos-67> (1m9.02s).
-----> Kitchen is finished. (1m9.69s)
```

Lab: Commit Your Work

```
$ cd ~/cookbooks/workstation
$ git add .
$ git status
$ git commit -m "Add additional tests for default test suite"
```

Lab: More Tests

- ✓ Add tests that validate that the remaining package resources have been installed (http://serverspec.org/resource_types.html#package)
- ✓ Add tests that validate the file resource (http://serverspec.org/resource_types.html#file)
- ✓ Run kitchen verify to validate the test meets the expectations that you defined
- ✓ Commit your changes

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 apache cookbook

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?

Lab: Testing Apache

- ❑ Create a test file for the "apache" cookbook's default recipe
- ❑ Add tests that validate a working web server

  http://serverspec.org/resource_types.html#port
  http://serverspec.org/resource_types.html#command
- ❑ Run kitchen verify
- ❑ Commit your changes

Lab: Return Home and 'cd cookbooks/apache'

```
$ cd ~/cookbooks/apache
```

Lab: What Does the Webserver Say?

~/cookbooks/apache/test/integration/default/serverspec/default_spec.rb

```
require 'spec_helper'

describe 'apache::default' do
  describe port(80) do
    it { should be_listening }
  end

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

Port 80 should be listening.

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

Lab: Commit Your Work

```
$ cd ~/cookbooks/apache
$ git add .
$ git status
$ git commit -m "Add tests for default test suite"
```

Lab: Testing Apache

- ✓ Create a test file for the "apache" cookbook's default recipe
- ✓ Add tests that validate a working web server

  http://serverspec.org/resource_types.html#port
  http://serverspec.org/resource_types.html#command
- ✓ Run kitchen verify
- ✓ Commit your changes

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?

CHEF™