Test-Driven Development with Chef

training@chef.io

2.1.8

Introductions

2.1.8

Instructor Introduction

- **Name:** Dan Robinson
- **Current job role:** Solutions Engineer, Consulting
- **Previous job roles/background:** Consulting Engineer with Opsware/HP for 5 years prior to joining Chef, working with HP Server Automation. Background also includes various consulting and engineering roles in email security, interactive television, and Internet services
- **Experience with Chef/Config Management:** 3 years working with (and working at) Chef
- **Favorite Text Editor:** SublimeText

Introduce Yourselves

- Name
- Current job role
- Previous job roles/background
- Experience with Chef and/or config management
- Favorite Text Editor

Course Objectives and Style

2.1.8

Course Objectives

- Upon completion of this course you will be able to
  - Utilize test-kitchen for local testing of cookbooks
  - Unit testing of recipes with Chefspec
  - Integration testing of recipes with Serverspec
  - Linting of cookbooks using Foodcritic and Rubocop

Training is really a discussion

- We will be doing things the **hard way**
- We're going to do **a lot** of typing
- You can't be:
  - Absent
  - Late
  - Left Behind
- The result is you reaching fluency fast

Training is really a discussion

- I'll post objectives at the beginning of a section
- Ask questions when they come to you
- Ask for help when you need it
- Class slides: https://s3.amazonaws.com/tdd-training-bucket/chefconf_tdd_workshop.pdf

Cookbook Style & Correctness

The real benefits of Infrastructure As Code

v1.1.4

Devops is a Two-Way Street

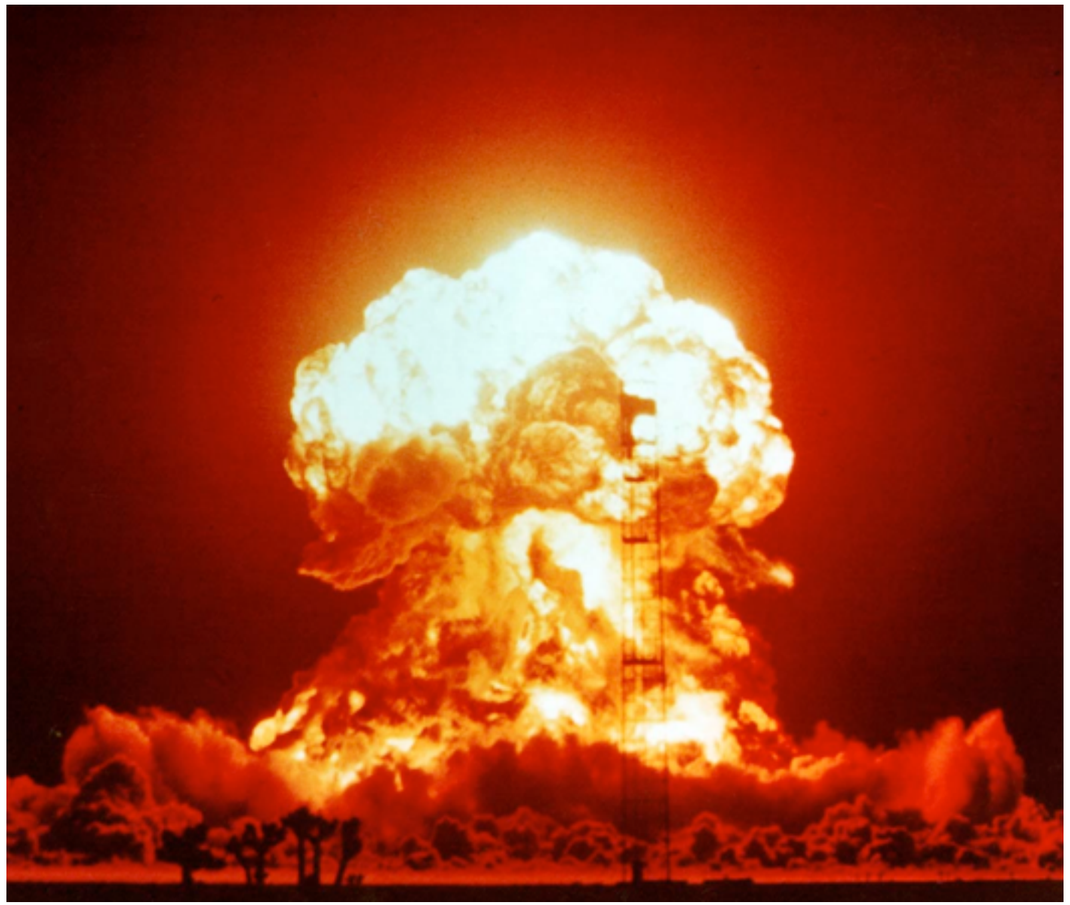
- It’s great when developers care about
  - **uptime**!
  - **scaling!**
  - **deployment!**
- Put them on call! etc. etc. etc.

Devops is a Two-Way Street

- **Operations** also has as much or more to learn from developers as well!

Old Software Development Workflow

- Write some code
- <ad-hoc verification here>
- Go to pre-production
- <ad-hoc verification here>
- Go to production
- **Production failure**



New Software Development Workflow

- Write some code
- Write and run some **unit tests**
- Go to pre-production
- Run some **integration/ acceptance tests**
- Go to production
- **Lowered chance of production failure**

Old Chef Cookbook Workflow

- Write some cookbook code
- <ad-hoc verification here>
- Go to pre-production
- <ad-hoc verification here>
- Go to production
- **Whoops, broke production**

New Chef Cookbook Workflow

- Write some cookbook code
- Check for **code correctness**
- Write and run some **unit tests**
- Go to pre-production
- Run some **integration tests**
- Go to production

Tools of the Trade

- **Code correctness**: Foodcritic, Rubocop
- **Unit tests**: ChefSpec
- **Integration tests**: Test Kitchen, ServerSpec, BATS

Workstation Setup

Getting started

2.1.8

Lesson Objectives

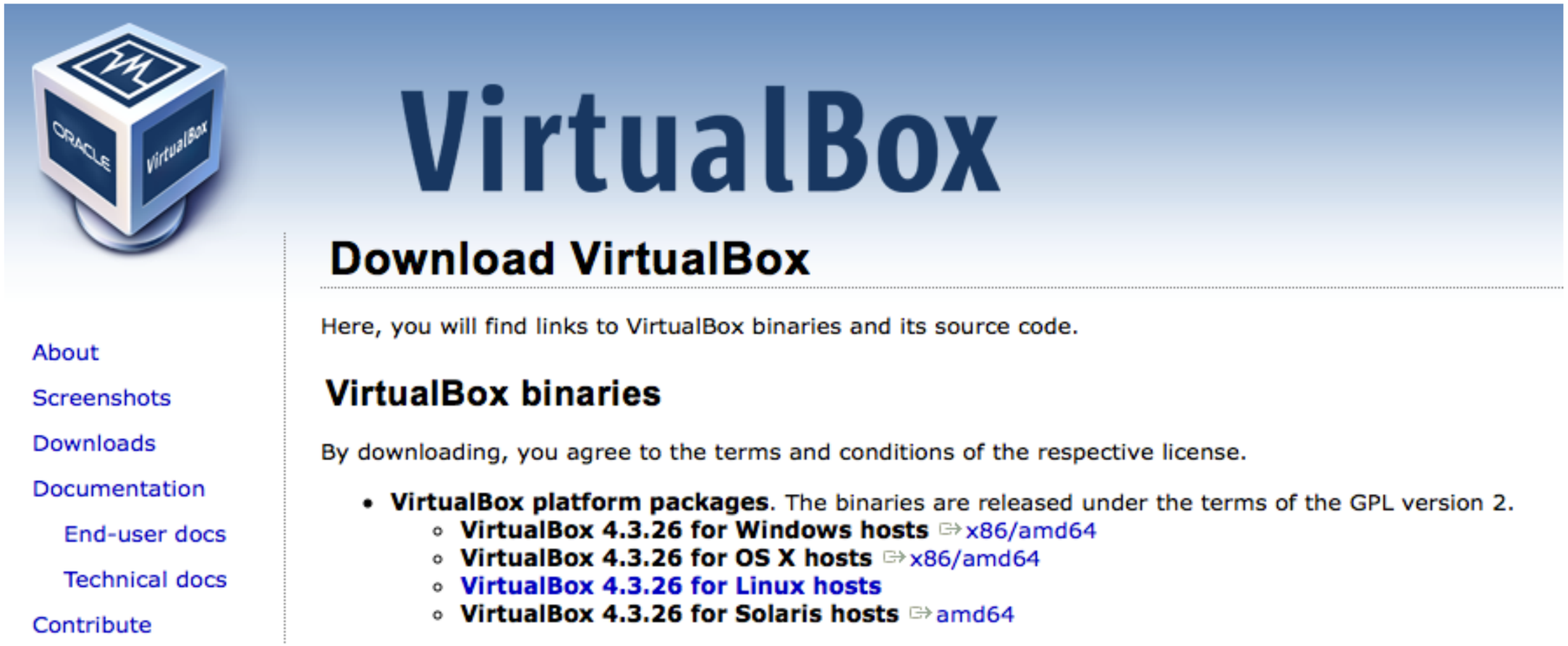
- After completing the lesson, you will be able to
  - Install Chef Development Kit
  - Install VirtualBox
  - Install Vagrant
  - Install a programmer's text editor (if you don't have one already)

Install Chef Development Kit

http://downloads.chef.io/chef-dk

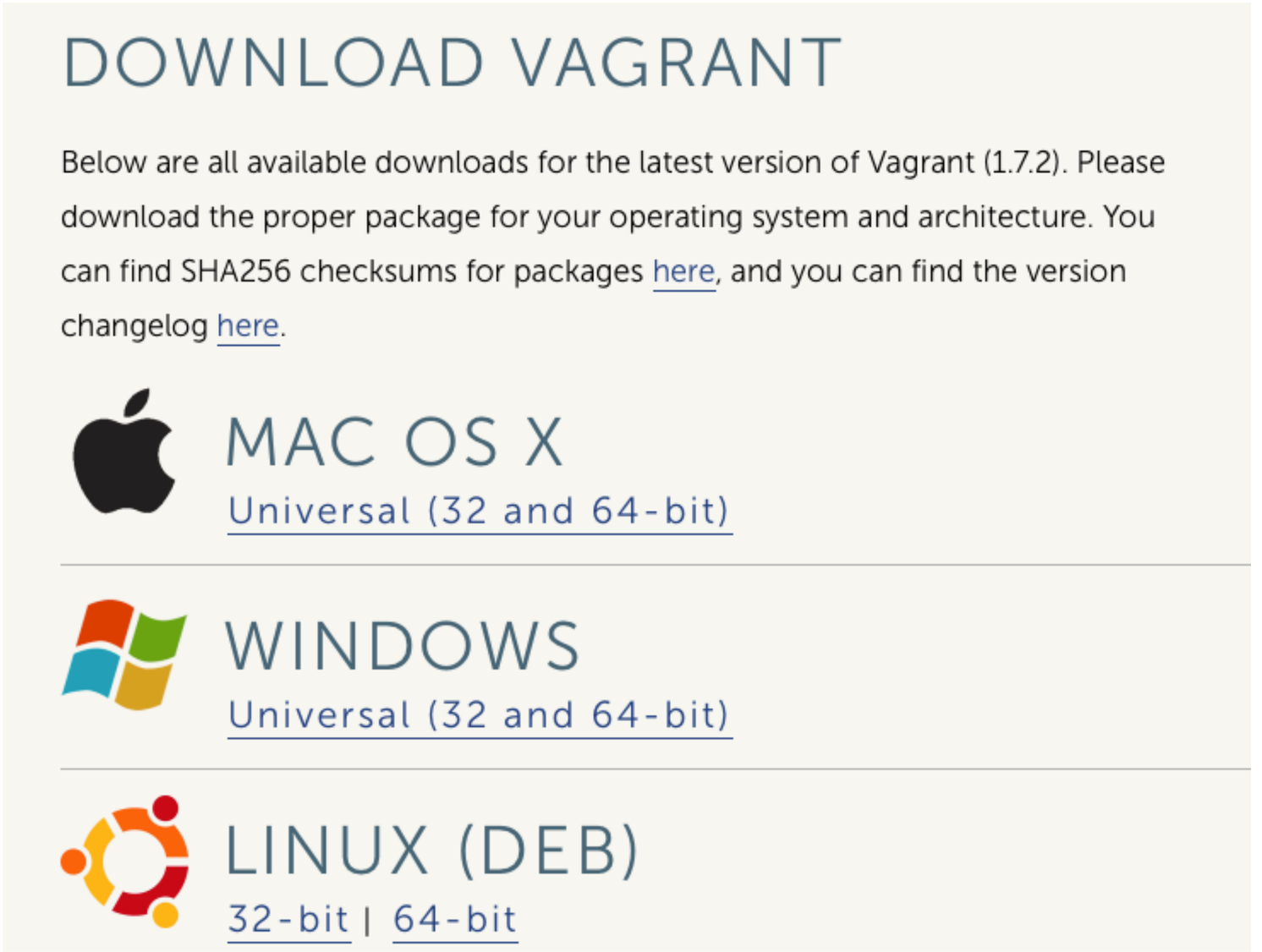
Install VirtualBox

https://www.virtualbox.org/wiki/Downloads



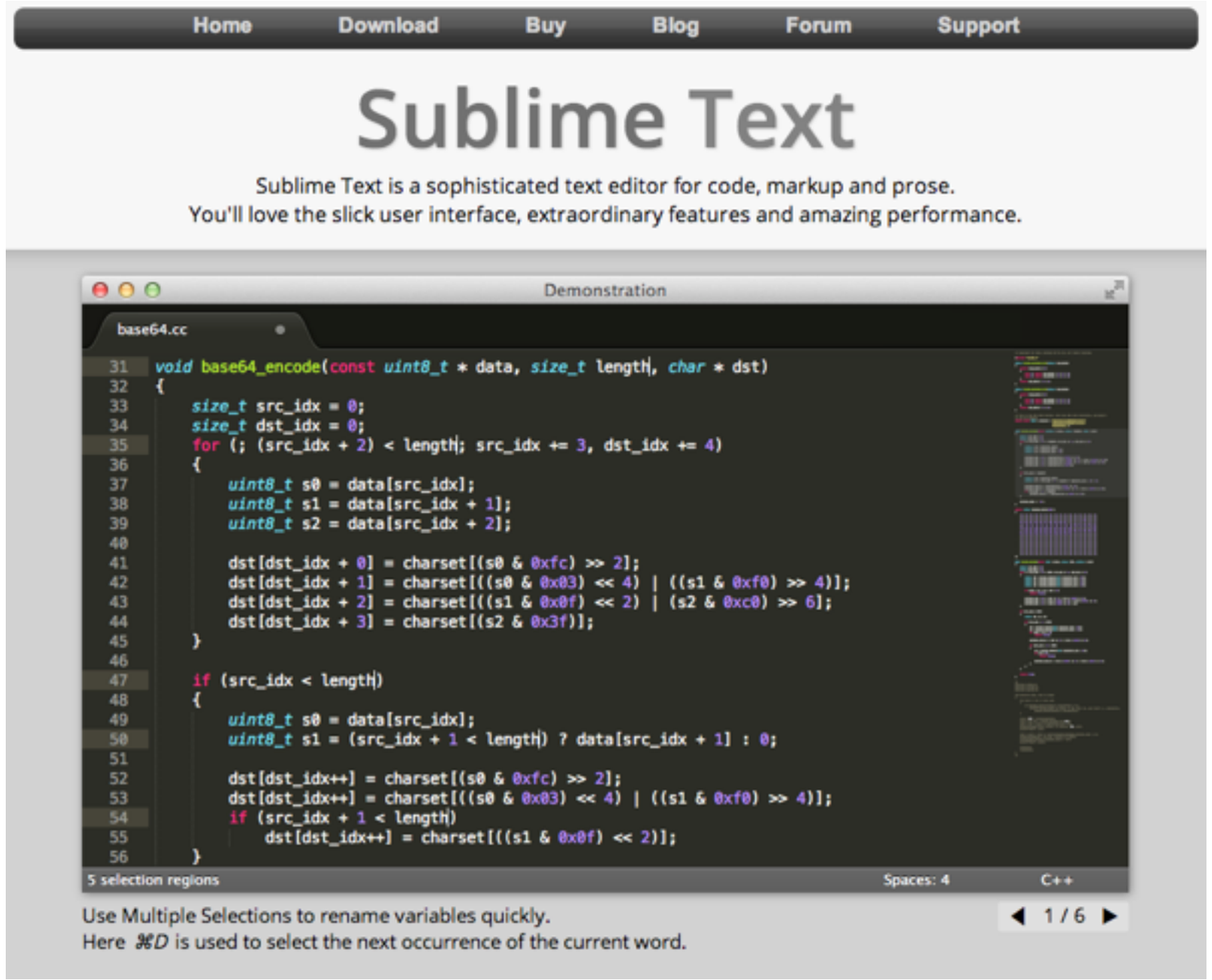
Install Vagrant

https://www.vagrantup.com



If you do not have a preferred text editor on your workstation already...

- Download Sublime Text
  - Free trial, not time bound
  - Works on every platform
- sublimetext.com



Test Kitchen

Creating a sandbox environment

2.1.8

Lesson Objectives

- After completing the lesson, you will be able to
  - Explain the value of functional testing
  - Create a sandbox environment using Test Kitchen
  - Inject test data to create a functional environment
  - Add networking configuration to the Test Kitchen configuration file

What is functional testing?

- Identify a function the cookbook is expected to perform
- Create input data
- Determine expected output
- Create a test case
- Execute the test case
- Compare actual and expected outputs

Test Kitchen

- Test Kitchen is a tool that ships with ChefDK and aids in rapid cookbook development and testing
- It's a pluggable testing framework that lets you
  - Create a sandbox environment
  - Configure the environment
  - Test the environment
- All in an automated fashion

Create Sandbox

- Test Kitchen sandbox environments can be created in many places
- Target is specified by the Test Kitchen ***driver*** and include
  - On your development workstation (virtual box, vagrant)
  - In a container (docker, lxc)
  - On a cloud instance
    - ec2, rackspace, openstack, digital ocean, etc

Configure Sandbox Environment

- Multiple ways to configure sandbox environments
- Specified by Test Kitchen ***provisioner***
  - chef_zero
  - chef_solo

Execute Tests

- Multiple testing frameworks supported
- Specified by Test Kitchen ***busser***
  - Serverspec*
  - Bats
  - Minitest

- *Serverspec is covered in this class

The Problem and the Success Criteria

- **The Problem:** We need to create a place where we can test our Apache cookbook without elaborate infrastructure
- **Success Criteria:** Be able to create, converge, and destroy a sandbox environment

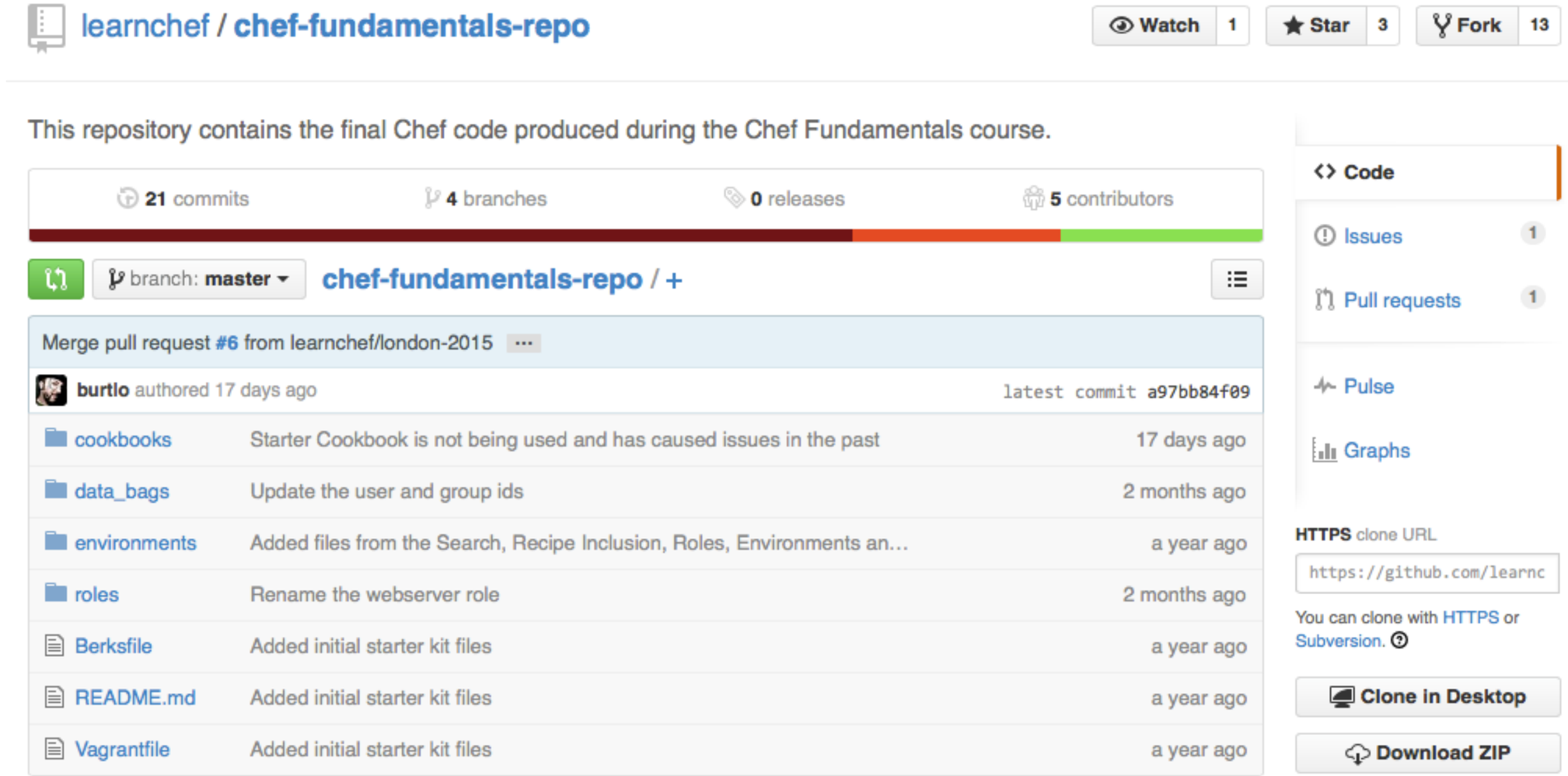
Exercise: Clone Repository

```
$ git clone https://github.com/learnchef/chef-fundamentals-repo
```

```
Cloning into 'chef-fundamentals-repo'...
remote: Counting objects: 575, done.
remote: Total 575 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (575/575), 363.35 KiB | 0
bytes/s, done.
Resolving deltas: 100% (126/126), done.
Checking connectivity... done.
```

Exercise: Clone Repository

https://github.com/learnchef/chef-fundamentals-repo



Exercise: Examine repository

```
$ cd chef-fundamentals-repo
```

```
.
├── Berksfile
├── cookbooks/
├── data_bags/
├── environments/
├── .git/
├── README.md
├── roles/
└── Vagrantfile

5 directories, 3 files
```

Exercise: Change to apache cookbook directory

```
$ cd $HOME/chef-fundamentals-repo/cookbooks/apache
```

```
.
├── attributes
├── CHANGELOG.md
├── files
├── metadata.rb
├── README.md
├── recipes
└── templates
```

Exercise: Configure Shell

```
$ eval "$(chef shell-init SHELL_NAME)"
```

- Configure your shell to use the ChefDK Ruby
- Required each time you start a new shell
- See output of `chef shell-init` for additional instructions
- Not necessary on Windows - Windows may require adjustments to your PATH.

Test Kitchen

- Test Kitchen is all about testing cookbooks
- **Perform all commands in the apache cookbook directory**

Test Kitchen Syntax

- Test Kitchen uses the following syntax
- `kitchen <command> [subcommand] [options]`

Test Kitchen Commands

- Key Test Kitchen commands
  - Help
  - List
  - Create
  - Login
  - Converge
  - Verify
  - Destroy
  - Test

Exercise: View Test Kitchen help

```
$ 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 diagnose [INSTANCE|REGEXP|all]   # Show computed diagnostic configuration
  kitchen driver                           # Driver subcommands
  kitchen driver create [NAME]             # Create a new Kitchen Driver gem project
  kitchen driver discover                  # Discover Test Kitchen drivers published on RubyGems
  kitchen driver help [COMMAND]            # Describe subcommands or one specific subcommand
  kitchen help [COMMAND]                   # Describe available commands or one specific command

...
```

Exercise: Add Test Kitchen Support

```
$ kitchen init
```

```
      create  .kitchen.yml
      create  test/integration/default
      create  .gitignore
      append  .gitignore
      append  .gitignore
         run  gem install kitchen-vagrant from "."
Fetching: kitchen-vagrant-0.15.0.gem (100%)
Successfully installed kitchen-vagrant-0.15.0
Parsing documentation for kitchen-vagrant-0.15.0
Installing ri documentation for kitchen-vagrant-0.15.0
Done installing documentation for kitchen-vagrant after 0 seconds
1 gem installed
```

Exercise: List Instances

```
$ kitchen list
```

```
Instance                Driver   Provisioner  Last Action
default-ubuntu-1204     Vagrant  ChefSolo     <Not Created>
default-centos-64       Vagrant  ChefSolo     <Not Created>
```

Exercise: Open Test Kitchen configuration file

OPEN IN EDITOR: .kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_solo

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

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

Editing YAML files

- YAML files must be “tab free”
- Spacing indicates hierarchy
- Two spaces is community preference
- **You will get errors if you use tabs, or do not indent properly**

Exercise: Edit Test Kitchen configuration file

OPEN IN EDITOR: .kitchen.yml

```
---
driver:
  name: vagrant

provisioner:
  name: chef_solo

platforms:
  - name: centos65
    driver_config:
      box: /local/path/to/your/centos65.box


.
```

SAVE FILE!

Exercise: Create sandbox instance

```
$ kitchen create
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Creating <default-centos-65>...
       Step 0 : FROM centos:6.4
        ---> 539c0211cd76
       Step 1 : RUN yum clean all
        ---> Running in 2b81a3a6cf17
...
 "StartedAt": "2014-06-21T03:07:35.468219224Z"
           },
           "Volumes": {},
           "VolumesRW": {}
       }
       ]
       Finished creating <default-centos-65> (0m28.37s).
-----> Kitchen is finished. (0m28.43s)
```

Exercise: Login to sandbox instance

```
$ kitchen login
```

```
Last login: Thu Oct 16 04:51:43 2014 from 10.0.2.2
[vagrant@default-centos65 ~]$exit
logout
Connection to localhost closed.
```

Exercise: Destroy sandbox instance

```
$ kitchen destroy
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Destroying <default-centos-64>...
       [{
           "Args": [
        "-D",
        "-o",
        "UseDNS=no",
...
 fec37e758e9a617a2183860044f797a6fc99aebdc9016f1d319e3dccb7284849
       fec37e758e9a617a2183860044f797a6fc99aebdc9016f1d319e3dccb7284849
       Finished destroying <default-centos-65> (0m0.38s).
-----> Kitchen is finished. (0m0.46s)
```

Exercise: Converge sandbox instance

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Creating <default-centos-65>...
       Step 0 : FROM centos:centos6
       Pulling repository centos
        ---> 68edf809afe7

...
       Finished creating <default-centos-65> (0m41.56s).
```

What just happened?

- Test Kitchen commands implicitly include prerequisite commands
  - Create
  - Converge
  - Setup
  - Verify
  - Destroy
  - Test

Converge implies create

Test implies create, converge, setup, verify, destroy

Feedback!

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Creating <default-centos-64>...
       Step 0 : FROM centos:6.4
        ---> 539c0211cd76
...
[2014-06-20T23:19:04:00] FATAL: Chef::Exceptions::ChildConvergeError: Chef run
process exited unsuccessfully (exit code 1)
>>>>>> Converge failed on instance <default-centos-64>.
>>>>>> Please see .kitchen/logs/default-centos-64.log for more details
>>>>>> ------Exception-------
>>>>>> Class: Kitchen::ActionFailed
>>>>>> Message: SSH exited (1) for command: [sudo -E chef-solo --config /tmp/
kitchen/solo.rb --json-attributes /tmp/kitchen/dna.json  --log_level info]
>>>>>> ----------------------
```

FAIL!

Feedback!

```
$ kitchen converge
```

```
[2014-10-06T22:27:44+01:00] FATAL: Stacktrace dumped to /tmp/kitchen/cache/chef-
stacktrace.out
Chef Client failed. 7 resources updated in 11.509284742 seconds
[2014-10-06T22:27:44+01:00] ERROR:

Chef::Mixin::Template::TemplateError (undefined method `[]' for nil:NilClass) on
line #3:

1: <html>
2:   <body>
3:     <h1>Welcome to <%= node["motd"]["company"] %></h1>
4:     <h2>We love <%= @site_name %></h2>
5:     <%= node["ipaddress"] %>:<%= @port %>
```

Undefined Attributes

- When you see `undefined method '[]' for nil:NilClass` it often means you have an undefined attribute
- Let's look in the apache cookbook attribute file to see if it's defined

Exercise: Verify attribute file

OPEN IN EDITOR: attributes/default.rb

```
default["apache"]["indexfile"] = “index1.html"
default["apache"]["sites"]["clowns"] = { "port" => 80 }
default["apache"]["sites"]["bears"] = { "port" => 81 }
```

- The motd attributes are in the motd cookbook

Testing Cookbooks

- Good cookbooks can be used in isolation
- They set ***reasonable defaults*** for all attributes used
- A good testing framework allows you to mock input data
- Test Kitchen lets us inject test attributes

Exercise: Edit Test Kitchen configuration file

OPEN IN EDITOR: .kitchen.yml

```
platforms:
  - name: centos65
    driver_config:
      box: /local/path/to/your/centos65.box

suites:
  - name: default
    run_list:
      - recipe[apache::default]
    attributes:
      motd:
        company: Chef
```

SAVE FILE!

Exercise: Converge sandbox instance

```
$ kitchen converge
```

```
--> Starting Kitchen (v1.2.2.dev)
--> Converging <default-centos-64>...
      Preparing files for transfer
      Preparing current project directory as a cookbook

...
      Chef Client finished, 5/10 resources updated in 5.49
 seconds
      Finished converging <default-centos-64> (0m7.50s).
 --> Kitchen is finished. (0m7.57s)
```

WIN!

Exercise: Configure Networking

OPEN IN EDITOR: cookbooks/apache/.kitchen.yml

```
provisioner:
  name: chef_solo

platforms:
  - name: centos65
    driver_config:
      box: /local/path/to/your/centos65.box
      network:
      - ["forwarded_port", {guest: 80, host: 8880}]
      - ["forwarded_port", {guest: 81, host: 8881}]

suites:
  - name: default
    run_list:
      - recipe[apache::default]
```

Forward ports to the sandbox environment

SAVE FILE!

Exercise: Rebuild Sandbox

```
$ kitchen destroy
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Destroying <default-centos-65>......

...

Finished destroying <default-centos-65> (0m0.87s).
-----> Kitchen is finished. (0m0.93s)
```

- Changing the network configuration requires destroy/create

Exercise: Converge Sandbox

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Creating <default-centos-65>...
       Step 0 : FROM centos:6.4
        ---> 539c0211cd76
       Step 1 : RUN yum clean all
        ---> Using cache
        ---> 737b6e6a416b
...
Chef Client finished, 11/11 resources updated in 33.065734092 seconds
       Finished converging <default-centos-65> (0m52.62s).
-----> Kitchen is finished. (0m53.00s)
```

Exercise: Observe network changes

```
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 80 => 8880 (adapter 1)
    default: 81 => 8881 (adapter 1)
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
```

Exercise: Verify clowns & bears

```
$ curl localhost:8880
```

```
<html>
  <body>
    <h1>Welcome to Chef</h1>
    <h2>We love clowns</h2>
    172.17.0.12:80
  </body>
</html>
```

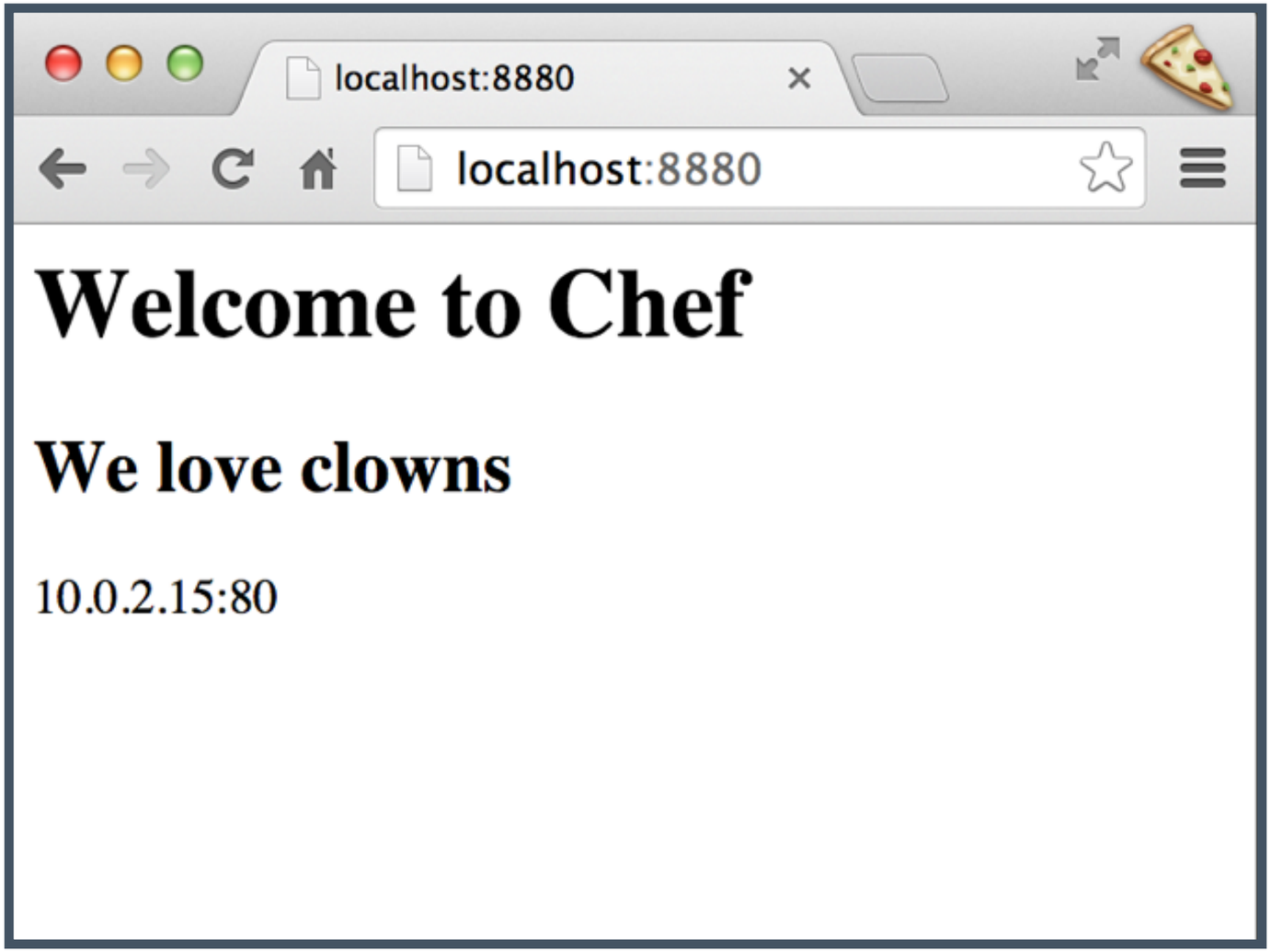
Exercise: Verify clowns & bears

```
$ curl localhost:8881
```

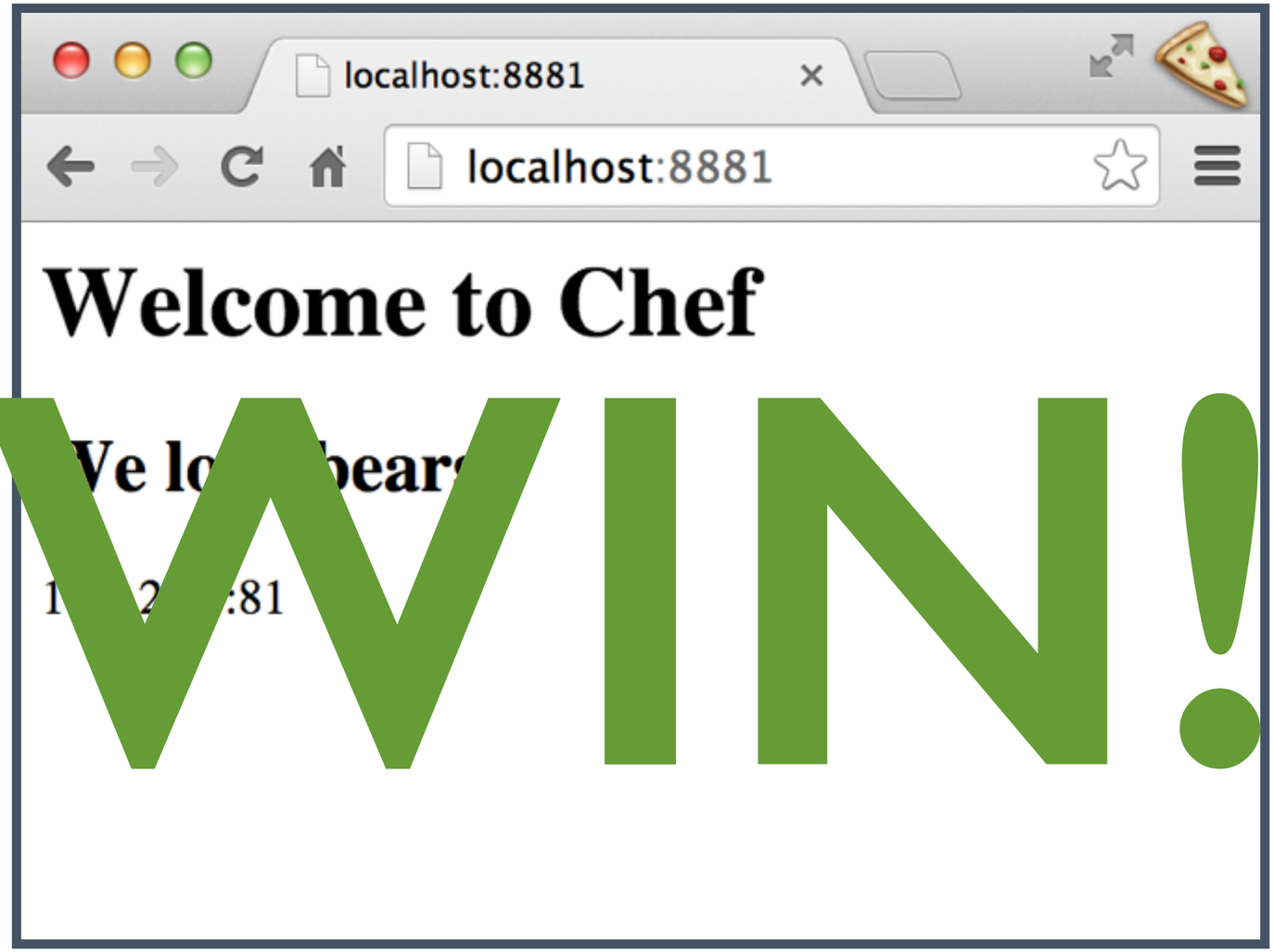
```
<html>
  <body>
    <h1>Wel ome    Che /h >
    <h2>We  ve be rs< n2>
    172.17.0  81
  </body>
</html>
```

WIN!

Exercise: Verify clowns



Exercise: Verify bears



More Resources

- Test Kitchen - http://kitchen.ci/
- Chef docs - https://docs.getchef.com/kitchen.html

Review Questions

- What is the name of the Test Kitchen configuration file?
- What is the name of the Test Kitchen component responsible for executing the tests?
- What steps are included in `kitchen test`?

Serverspec

Automatically verifying your results

2.1.8

Lesson Objectives

- After completing this lesson you will be able to
  - Use Test Kitchen to Create Serverspec tests using the ‘Expect’ form
  - Execute Serverspec tests using Test Kitchen
  - Find additional information about Serverspec commands and matchers

Serverspec

- Serverspec lets you write RSpec tests for checking your servers are configured correctly
- You are testing the **actual state** by executing commands on the target server
- Supports multiple transports SSH, WinRM, Docker API

Serverspec tests

- Test Kitchen looks in the `test/integration` directory for test-related files
- Test Kitchen creates this for you when you run `kitchen init`

Exercise: Review cookbooks/apache

```
.
├── attributes/
├── CHANGELOG.md
├── files/
├── metadata.rb
├── README.md
├── recipes/
├── templates/
└── test
    └── integration
```

Suite directory

- Test Kitchen supports multiple test suites
- Each suite can use a different framework
- Test Kitchen requires additional directories for our Serverspec tests

```
└─test
   └─integration
      └─ <suite_name>
```

Exercise: Find suite name

OPEN IN EDITOR: .kitchen.yml

```
...
platforms:
  - name: centos65

suites:
  - name: default
    run_list:
      - recipe[apache::default]
    attributes:
      motd:
        company: Chef
```

Suite directory

Our suite name is `default`

```
└── test
    └── integration
        └── default
```

Busser directory

- The next directory level denotes the ***test plugin***.
- A Test Kitchen plugin is called a ***busser***. We're using the serverspec ***busser***

```
└─test
   └─integration
      └─default
         └─ serverspec
```

Hostname directory

- Serverspec supports testing remote nodes, so yet another directory level denotes the ***hostname***
- We won't be using the remote testing capability, so it should be ***localhost***

```
└─test
  └─integration
    └─default
      └─serverspec
        └─localhost
```

- **Note**: the hostname directory is optional if testing localhost

Exercise: Create directories

```
$ mkdir -p test/integration/default/serverspec/localhost
```

Test files

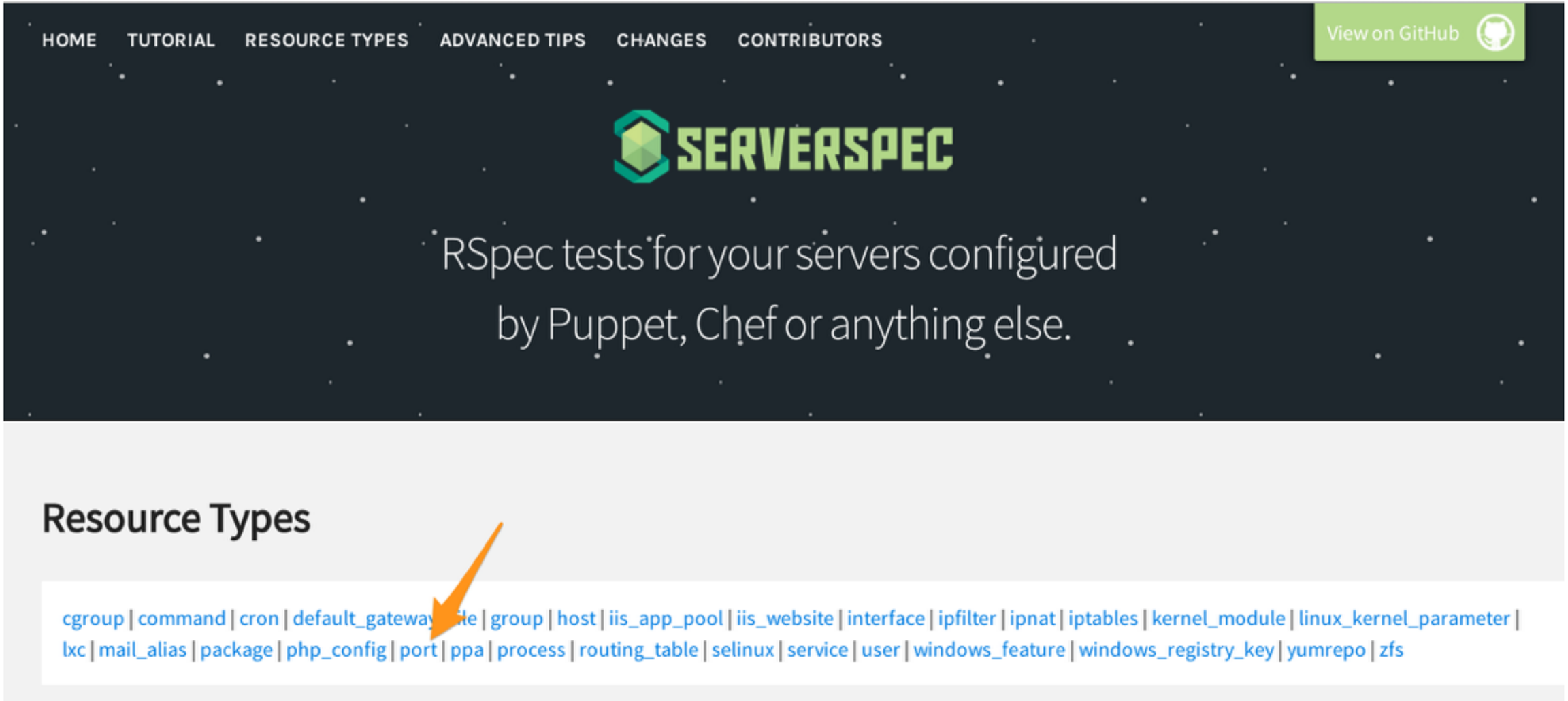
- Test Kitchen expects to find tests in files named like `*_spec.rb`
- Test kitchen will execute all files in the `test/integration` tree that end in `_spec.rb`
- Tests are described using a modified RSpec notation form

Serverspec commands & matchers

- Serverspec provides a wide variety of ***matchers*** for each ***command***
- Serverspec commands are documented here

http://serverspec.org/resource_types.html

Serverspec resources



Port resource

port

Port resource type.

be_listening

In order to test a given port is listening, you should use **be_listening** matcher.

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

You can also specify `tcp` , `udp` , `tcp6` , or `udp6` .

The Problem and the Success Criteria

- **The Problem:** We need an automated way to test that our Apache cookbook works properly
- **Success Criteria:** Create functional tests that verify each site returns the expected data

Kitchen setup

- Run `kitchen setup` before executing tests
- `kitchen setup` loads and configures the files necessary to run the test plugins on the node
- The component that manages Test Kitchen plugins is called the **Busser**

Exercise: kitchen setup

```
$ kitchen setup
```

```
-----> Starting Kitchen (v1.2.2.dev)
-----> Setting up <default-centos-65>...
Fetching: thor-0.19.0.gem (100%)
Fetching: busser-0.6.2.gem (100%)
Successfully installed thor-0.19.0
Successfully installed busser-0.6.2
2 gems installed
-----> Setting up Busser
       Creating BUSSER_ROOT in /tmp/busser
Creating busser binstub
       Plugin serverspec installed (version 0.2.6)
-----> Running postinstall for serverspec plugin
       Finished setting up <default-centos-65> (0m22.46s).
-----> Kitchen is finished. (0m22.53s)
```

Exercise: Test clowns site

OPEN: test/integration/default/serverspec/localhost/clowns_spec.rb

```
require 'serverspec'

set :backend, :exec

describe 'clowns site' do
  it 'responds on port 80' do
    expect(port 80).to be_listening 'tcp'
  end
end
```

SAVE FILE!

Kitchen Verify

- The `kitchen verify` command will run all of your `*_spec.rb` tests in the `test/integration` tree

Exercise: kitchen verify

```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.2.1)
-----> Verifying <default-centos-64>...
...
       clowns s  e
         respond  on  or  30

       Finished i  .06427  conds  ile  took  17 seconds to load)
       1 example,  failure
       Finished verifying <default-centos-64> (0m1.38s).
-----> Kitchen is finished. (0m1.69s)
```

WIN!

Kitchen Verify

- Are we really sure this test does something useful?
- Let’s modify the port in our test case and see if we can make it fail

Exercise: Verify our tests

OPEN: test/integration/default/serverspec/localhost/clowns_spec.rb

```
require 'serverspec'

set :backend, :exec

describe 'clowns site' do
  it 'responds on port 85' do
    expect(port 85).to be_listening 'tcp'
  end
end
```

SAVE FILE!

Exercise: kitchen verify

```
$ kitchen verify
```

```
clowns site
  responds on port 85 (FAILED - 1)

Failures:

  1) clowns site responds on port 85
     Failure/Error: expect(port(85)).to be_listening('tcp')
       expected Port "85" to be listening on "tcp"
       /bin/sh -c "netstat -tln | grep -p -E :85

     # /tmp/busser/suites/serverspec/localhost_default_spec.rb:8:in `block (2 levels)
in <top (required)>'

Finished in 0.07091 seconds (files took 0.24667 seconds to load)
1 example, 1 failure
```

WIN!

Exercise: Verify our tests

OPEN: test/integration/default/serverspec/localhost/clowns_spec.rb

```
require 'serverspec'

set :backend, :exec

describe 'clowns site' do
  it 'responds on port 80' do
    expect(port 80).to be_listening 'tcp'
  end
end
```

SAVE FILE!

Exercise: kitchen verify

```
$ kitchen verify
```

```
-----> Starting Kitchen (v1.2.1)
-----> Verifying <default-centos-64>...
...
       clowns s  e
         respond  on  or  30

       Finished i  .06427  conds  ile  took  17 seconds to load)
       1 example,  failure
       Finished verifying <default-centos-64> (0m1.38s).
-----> Kitchen is finished. (0m1.69s)
```

WIN!

Exercise: Test bears site

OPEN: test/integration/default/serverspec/localhost/bears_spec.rb

```
require 'serverspec'

set :backend, :exec

describe 'bears site' do
  it 'responds on port 81' do
    expect(port 81).to be_listening 'tcp'
  end
end
```

SAVE FILE!

Exercise: kitchen verify

```
$ kitchen verify
```

```
       bears site
          responds on port 81

        clowns s  e
          respond  on  or  80

        Finished i  07885  conds  ile  took  03 seconds to load)
        2 examples,  failu  s
        Finished verifying <default-centos-64> (0m1.81s).
-----> Kitchen is finished. (0m2.25s)
```

WIN!

Code Cleanup

- Common code can be moved to a file called `spec_helper.rb`

- Test Kitchen automatically looks for `spec_helper.rb in test/integration/default/serverspec`

Exercise: Create spec_helper.rb

OPEN: `test/integration/default/serverspec/spec_helper.rb`

```
require 'serverspec'

set :backend, :exec
```

SAVE FILE!

Exercise: Clean-up clowns site

OPEN: test/integration/default/serverspec/localhost/clowns_spec.rb

```
require 'spec_helper'

set :backend, :exec

describe 'clowns site' do
  it 'responds on port 80' do
    expect(port 80).to be_listening 'tcp'
  end
end
```

Remove this line

SAVE FILE!

Exercise: Clean-up bears site

OPEN: `test/integration/default/serverspec/localhost/bears_spec.rb`

```
require 'spec_helper'

set :backend, :exec

describe 'bears site' do
  it 'responds on port 81' do
    expect(port 81).to be_listening 'tcp'
  end
end
```

Remove this line

SAVE FILE!

Exercise: Verify with spec_helper

```
$ kitchen verify
```

```
       bears site
          responds on port 81

        clowns s  e
          respond  on  or  80

        Finished i  07885  conds  ile  took  03 seconds to load)
        2 examples,  failu  s
        Finished verifying <default-centos-64> (0m1.81s).
-----> Kitchen is finished. (0m2.25s)
```

WIN!

Exercise: Test bears content

OPEN: test/integration/default/serverspec/localhost/bears_spec.rb

```
require 'spec_helper'

describe 'bears site' do
  it 'responds on port 81' do
    expect(port 81).to be_listening 'tcp'
  end

  it 'returns bears in the HTML body' do
    expect((command 'curl localhost:81').stdout).to match(/bears/)
  end
end
```

SAVE FILE!

Exercise: Test Package Installation

OPEN: test/integration/default/serverspec/localhost/server_spec.rb

```
require 'spec_helper'

describe 'server' do
  it 'has apache installed' do
    expect(package 'httpd').to be_installed
  end
end
```

SAVE FILE!

Exercise: Verify all tests

```
$ kitchen verify
```

```
      bears site
        responds on port 81
        returns bears in the HTML body

      clowns site
        responds on port 81
        returns clowns in the HTML body

      server
        has apache installed

      Finished in 0.19711 seconds (files took 0.24900 seconds to load)
      5 examples, 0 failures
      Finished verifying <default-centos-64> (0m2.74s).
-----> Kitchen is finished. (0m3.06s)
```

WIN!

Kitchen test

- Kitchen commands implicitly include prerequisite commands
- Running kitchen test result in

```
kitchen destroy     (if necessary)
kitchen create
kitchen converge
kitchen setup
kitchen verify
kitchen destroy
```

Kitchen test

- Use `kitchen test` as your final check after creating tests, or making changes before committing to source control
- `kitchen test` Can be useful in a Continuous Delivery pipeline

Exercise: Verify all tests

```
$ kitchen test
```

```
----> Starting Kitchen (v1.2.1)
-----> Cleaning up any prior instances of <default-centos-64>
-----> Destroying <default-centos-64>...
...

     Finished in 0.621 seconds (files took 0.500 seconds to
load)
       5 examples, 0 failures
       Finished verifying <default-centos-64> (0m5.01s).
-----> Destroying <default-centos-64>...
```

WIN!

Additional Information

- Look at community cookbooks for additional ideas and advanced uses of Serverspec
- The Jenkins cookbook as some of the best examples and techniques available.

Review Questions

- Where does Test Kitchen look for tests?
- Serverspec tests are written using commands and _____?
- What is one characteristic of a good testing framework?
- What is the name of the file where we store code that is common to multiple test cases?

Foodcritic

Lint for Chef

2.1.8

Lesson Objectives

- After completing this lesson you will be able to
  - Execute Foodcritic tests
  - Configure Foodcritic to include or exclude specific tests
  - Install and test using additional rules

Foodcritic

- Foodcritic helps you detect problems that will cause a runtime failure
- Helps you enforce desirable behavior
- Detects common anti-patterns
- Works quickly without requiring you to run chef-client and converge a node

Exercise: Execute Foodcritic

```
$ foodcritic .
```

```
FC003: Check whether you are running with chef server before using
server-specific features: ./recipes/ip-logger.rb:1
FC008: Generated cookbook metadata needs updating: ./metadata.rb:2
FC008: Generated cookbook metadata needs updating: ./metadata.rb:3
```

More Feedback

- Fast feedback
- Foodcritic tests your cookbook against a set of rules
- Rules are documented at

  http://acrmp.github.io/foodcritic

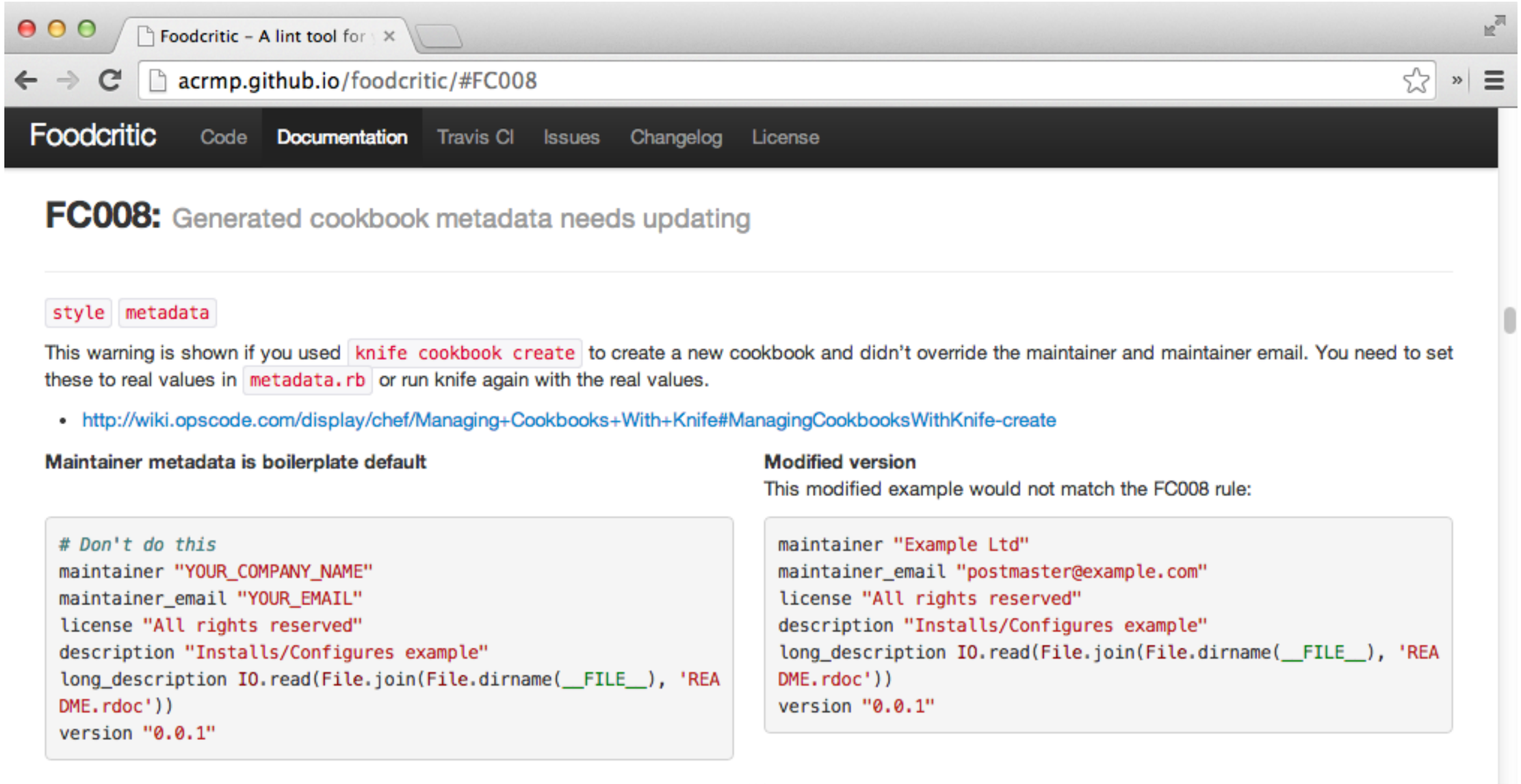
- The default rules are a good start
- You can add your own

Default Foodcritic Rules

Rules Cultivating a refined palate

- FC001: Use strings in preference to symbols to access node attributes
- FC002: Avoid string interpolation where not required
- FC003: Check whether you are running with chef server before using server-specific features
- FC004: Use a service resource to start and stop services
- FC005: Avoid repetition of resource declarations
- FC006: Mode should be quoted or fully specified when setting file permissions
- FC007: Ensure recipe dependencies are reflected in cookbook metadata
- FC008: Generated cookbook metadata needs updating
- FC009: Resource attribute not recognised
- FC010: Invalid search syntax
- FC011: Missing README in markdown format
- FC012: Use Markdown for README rather than RDoc
- FC013: Use file_cache_path rather than hard-coding tmp paths
- FC014: Consider extracting long ruby_block to library
- FC015: Consider converting definition to a LWRP
- FC016: LWRP does not declare a default action
- FC017: LWRP does not notify when updated
- FC018: LWRP uses deprecated notification syntax
- FC019: Access node attributes in a consistent manner
- FC020: Conditional execution string attribute looks like Ruby
- FC021: Resource condition in provider may not behave as expected
- FC022: Resource condition within loop may not behave as expected
- FC023: Prefer conditional attributes
- FC024: Consider adding platform equivalents
- FC025: Prefer chef_gem to compile-time gem install
- FC026: Conditional execution block attribute contains only string
- FC027: Resource sets internal attribute
- FC028: Incorrect #platform? usage
- FC029: No leading cookbook name in recipe metadata
- FC030: Cookbook contains debugger breakpoints
- FC031: Cookbook without metadata file
- FC032: Invalid notification timing
- FC033: Missing template
- FC034: Unused template variables
- FC035: Template uses node attribute directly
- FC037: Invalid notification action
- FC038: Invalid resource action
- FC039: Node method cannot be accessed with key
- FC040: Execute resource used to run git commands
- FC041: Execute resource used to run curl or wget commands
- FC042: Prefer include_recipe to require_recipe
- FC043: Prefer new notification syntax
- FC044: Avoid bare attribute keys
- FC045: Consider setting cookbook name in metadata

What about FC008?



Exercise: Fix FC008

OPEN IN EDITOR: metadata.rb

```
name                'apache'
maintainer          'YOUR_COMPANY_NAME'
maintainer_email    'YOUR_EMAIL'
license             'All rights reserved'
description         'Installs/Configures apache'
long_description
IO.read(File.join(File.dirname(__FILE__),
'README.md'))
version             '0.2.0'
```

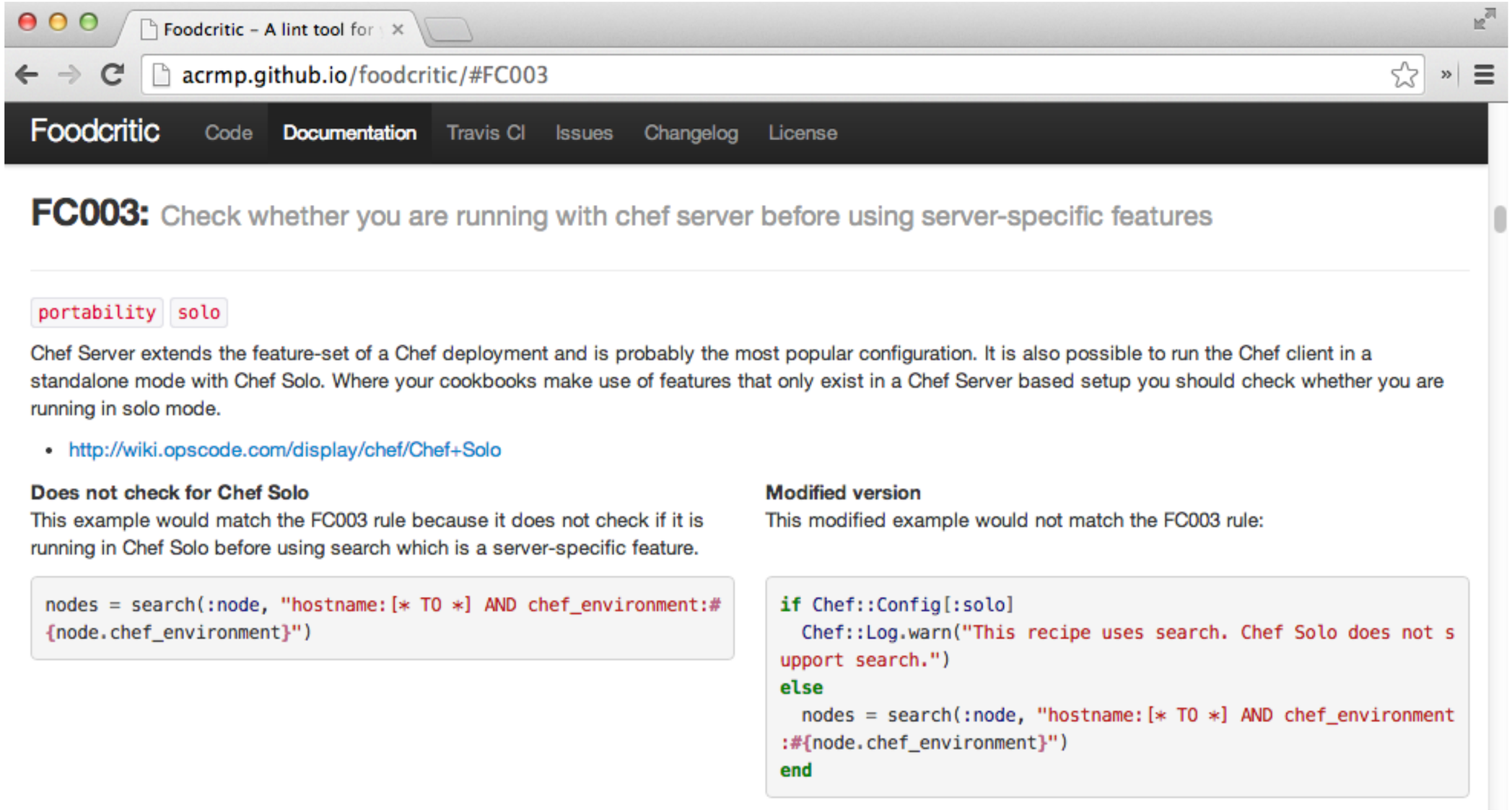
SAVE FILE!

Exercise: Execute Foodcritic

```
$ foodcritic .
```

```
FC003: Check whether you are running with chef server
before using server-specific features: ./recipes/ip-
logger.rb:1
```

What about FC003?



Ignoring Rules

- Rules can be ignored
- FC003 was a bigger deal before chef_zero.
- Some rules may not make sense for your scenario
- Use `--tags <TAGS>` to specify rules
- Test only specific rules
  - `foodcritic --tags FC001,FC002,FC003`
- Exclude a specific rule with ~
  - `foodcritic --tags ~FC003`

Exercise: Execute Foodcritic

```
$ foodcritic --tags ~FC003 .
```

WIN!

Review Questions

- What is one advantage Foodcritic has over Serverspec?
- How can you tell Foodcritic to exclude specific tests?
- Where do Foodcritic tests execute?

Rubocop

Analyze against Ruby community style guide

2.1.8

Lesson Objectives

- After completing this lesson, you will be able to
  - Evaluate your Ruby coding style against community standards
  - Improve your code by iterating through style warnings
  - Prevent specific rules from generating warnings

Rubocop

- New Ruby developers often ask for guidance on writing idiomatic Ruby
- Rubocop gives the same kind of feedback for your Ruby style that Foodcritic gives for your Chef Code
- Evaluates your Chef code for Ruby best practices, not for Chef style
- Documentation located here

https://github.com/bbatsov/rubocop

Exercise: Use additional rules

```
$ rubocop
```

```
recipes/default.rb:53:7: C: Use the new Ruby 1.9 hash syntax.
      :site_name => site_name,
      ^^^^^^^^^^^^^
recipes/default.rb:54:7: C: Use the new Ruby 1.9 hash syntax.
      :port => site_data["port"]
      ^^^^^^^^
recipes/default.rb:54:26: C: Prefer single-quoted strings when you don't
need string interpolation or special symbols.
      :port => site_data["port"]
                         ^^^^^^

7 files inspected, 38 offences detected
```

Fixing Rubocop offenses

- We can mask offenses using the rubocop configuration file `.rubocop.yml`
- Rubocop can automatically generate a configuration called file named `.rubocop_todo.yml` that contains entries to mask all of your offenses
- We can use `.rubocop_todo.yml` to iterate through the offenses

Fixing Rubocop offenses

- `.rubocop_todo.yml` settings are documented at

https://github.com/bbatsov/rubocop/blob/master/README.md

Exercise: Generate .rubocop_todo.yml

```
$ rubocop --auto-gen-config
```

```
recipes/default.rb:54:7: C: Use the new Ruby 1.9 hash syntax.
      :port => site_data["port"]
      ^^^^^^^^
recipes/default.rb:54:26: C: Prefer single-quoted strings when you don't
need string interpolation or special symbols.
      :port => site_data["port"]
                         ^^^^^^

7 files inspected, 38 offences detected
Created .rubocop_todo.yml.
Run rubocop with --config .rubocop_todo.yml, or
add inherit_from: .rubocop_todo.yml in a .rubocop.yml file.
```

Exercise: Review .rubocop_todo.yml

OPEN IN EDITOR: .rubocop_todo.yml

```
 This configuration was generated by `rubocop --auto-gen-config`
# on 2014-10-09 10:42:37 -0400 using RuboCop version 0.18.1.
# The point is for the user to remove these configuration records
# one by one as the offences are removed from the code base.
# Note that changes in the inspected code, or installation of new
# versions of RuboCop, may require this file to be generated again.

# Offence count: 4
# Cop supports --auto-correct.
# Configuration parameters: SupportedStyles.
HashSyntax:
  EnforcedStyle: hash_rockets

# Offence count: 2
LineLength:
  Max: 127
```

Exercise: Review .rubocop_todo.yml

OPEN IN EDITOR: `.rubocop_todo.yml`

```
# Offence count: 2
# Cop supports --auto-correct.
SpaceInsideBrackets:
  Enabled: false

# Offence count: 28
# Cop supports --auto-correct.
# Configuration parameters: EnforcedStyle, SupportedStyles.
StringLiterals:
  Enabled: false

# Offence count: 1
# Cop supports --auto-correct.
TrailingWhitespace:
  Enabled: false
```

Exercise: Update .rubocop.yml

OPEN IN EDITOR: `.rubocop.yml`

```
inherit_from: .rubocop_todo.yml
```

SAVE FILE!

Exercise: Re-run rubocop

```
$ rubocop
```

```
Inspecting 7 files
.......

7 files inspected, no offences detected
```

Work through the issues

- We now have a configuration file that masks all of our offenses
- We want to fix some of the offenses and mask the others
- Move the configuration rules for the offenses we want to continue masking from
- `.rubocop_todo.yml` to `.rubocop.yml`

Exercise: Update Rubocop configuration

OPEN IN EDITOR: cookbooks/apache/.rubocop.yml

```
#inherit_from: .rubocop_todo.yml  <<- Comment out this line

Encoding:
  Enabled: false

LineLength:
  Max: 200

HashSyntax:
  EnforcedStyle: hash_rockets

StringLiterals:
  Enabled: false
```

SAVE FILE!

Exercise: Re-generate .rubocop_todo.yml

```
$ rubocop --auto-gen-config
```

```
recipes/default.rb:15:11: C: Space inside square brackets detected.
  action [ :enable, :start ]
          ^
recipes/default.rb:15:27: C: Space inside square brackets detected.
  action [ :enable, :start ]
                          ^
recipes/default.rb:19:87: C: Trailing whitespace detected.
execute "mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/welcome.conf.disabled" do
                                                                                      ^

7 files inspected, 4 offences detected
Created .rubocop_todo.yml.
Run rubocop with --config .rubocop_todo.yml, or
add inherit_from: .rubocop_todo.yml in a .rubocop.yml file.
```

Fixing Rubocop offenses

- We decide we want to fix these offenses
  1. Add `inherit_from:` to `.rubocop.yml`
  2. Remove an entry from `.rubocop_todo.yml`
  3. Fix offenses
  4. Verify fix by running Rubocop
  5. Lather-Rinse-Repeat (steps 2-4)
  6. Remove `.rubocop_todo.yml`

Exercise: Update .rubocop.yml

OPEN IN EDITOR: `.rubocop.yml`

```
inherit_from: .rubocop_todo.yml
```

<<- Un-comment this line

```
Encoding:
  Enabled: false

LineLength:
  Max: 200

HashSyntax:
  EnforcedStyle: hash_rockets

StringLiterals:
  Enabled: false
```

SAVE FILE!

Exercise: Delete entry from .rubocop_todo.yml

OPEN IN EDITOR: .rubocop_todo.yml

```
# Offence count: 1
# Cop supports --auto-correct.
SpaceAfterComma:
  Enabled: false

# Offence count: 2
# Cop supports --auto-correct.
SpaceInsideBrackets:
  Enabled: false

# Offence count: 1
# Cop supports --auto-correct.
TrailingWhitespace:
  Enabled: false
```

SAVE FILE!

Exercise: Re-run rubocop

```
$ rubocop
```

```
Offences:

recipes/default.rb:19:87: C: Trailing whitespace
detected.
execute "mv /etc/httpd/conf.d/welcome.conf /etc/httpd/
conf.d/welcome.conf.disabled" do

^
```

Exercise: Fix default recipe

OPEN IN EDITOR: recipes/default.rb

```
package "httpd" do
  action :install
end

service "httpd" do
  action [ :enable, :start ]
end

# Disable the default virtual host
execute "mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/
welcome.conf.disabled" do
```

Remove trailing whitespace from line 19

SAVE FILE!

Exercise: Re-run rubocop

```
$ rubocop
```

```
Inspecting 7 fi
.......

7 files insp  ted,  off  ce  dete  d
```

WIN!

Review Questions

- What guidelines does Rubocop help you enforce?
- Why might you want to exclude certain cops?
- What is the name of Rubocop configuration file?

ChefSpec

Unit testing your cookbook

2.1.8

Lesson Objectives

- After completing this lesson, you will be able to
  - Create ChefSpec tests using Expect notation
  - Execute ChefSpec tests
  - Find additional information and examples of ChefSpec commands and matchers

ChefSpec

- ChefSpec provides a framework for performing unit tests on your cookbooks
- It's *FAST*
- If you start with a ChefSpec, you end up with runnable documentation for your cookbook
- You end up with a set of tests that help uncover bugs as you refactor

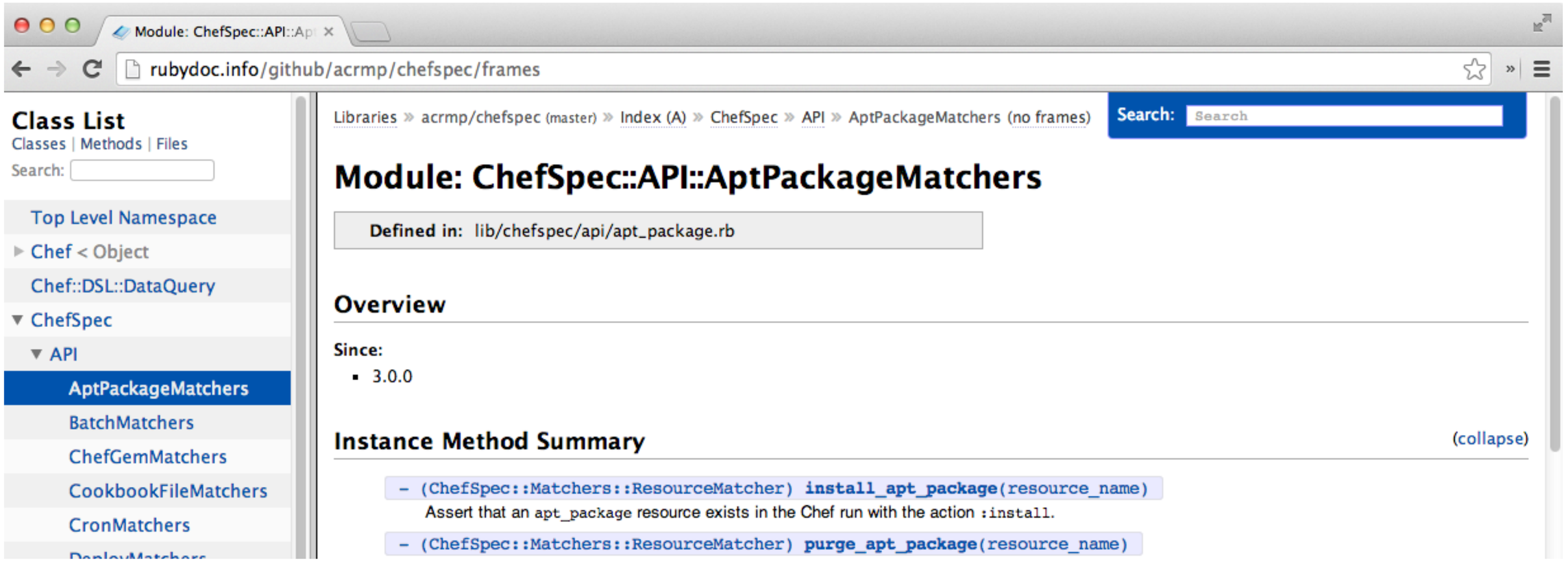
ChefSpec

- Builds on RSpec just like Serverspec
- ChefSpec looks for tests in the `spec/` directory of your cookbook
- Just like Serverspec, ChefSpec looks for files in the form `*_spec.rb`
- ChefSpec matcher documentation

http://rubydoc.info/github/acrmp/chefspec/frames

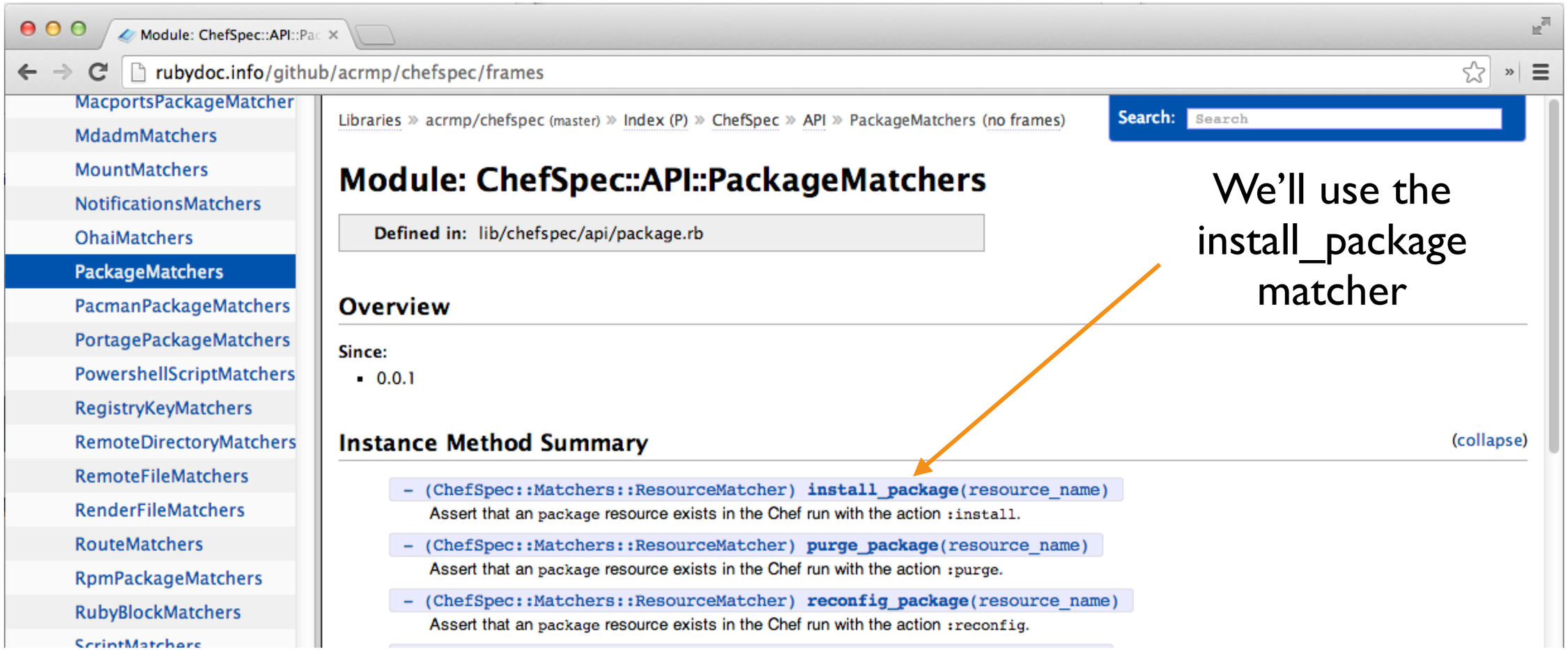
ChefSpec

- The important information is under ChefSpec->Api



ChefSpec

- Each API has an example



Exercise: Create spec/ directory

```
$ mkdir spec
```

Exercise: Test apache::default recipe

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'

describe 'apache::default' do
  chef_run =
ChefSpec::ServerRunner.new.converge('apache::default')

  it 'installs apache2' do
    expect(chef_run).to install_package('httpd')
  end
end
```

SAVE FILE!

Exercise: Execute tests

```
$ rspec --color -fd
```

```
apache::default
  installs apache2

Finished in 0.007 seconds (files took 1.79 seconds
to load)
1 example, 0 failures
```

WIN!

What just happened?

```
1: require 'chefspec'
2:
3: describe 'apache::default' do
```

- We used rspec to execute the tests. There's not a separate executable.
- 1: Include the chefspec gem. The chefspec gem includes all of the custom matchers
- 3: The following tests describe the apache::default recipe

What just happened?

```
4:  chef_run =
ChefSpec::ServerRunner.new.converge('apache::default') 5:
6:  it 'installs apache2' do
7:    expect(chef_run).to install_package('httpd')
8:  end
```

- 4: Perform a fake Chef run using a run_list of recipe[apache::default]. Store the resulting resource collection in the variable chef_run
- 6: Identify the test
- 7: Use custom commands and matchers to verify the content of chef_run

Is this for real?

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'

describe 'apache::default' do
  chef_run =
ChefSpec::ServerRunner.new.converge('apache::default')

  it 'installs apache2' do
    expect(chef_run).to install_package('bad_httpd')
  end
end
```

SAVE FILE!

Exercise: Execute tests

```
$ rspec --color -fd
```

```
Failures:

  1) apache::default installs apache2
     Failure/Error: expect(chef_run).to install_package('load_httpd')
       expected "package[load_httpd]" with action :install to be in Chef run.
Other package installs:

         package[httpd]

     # ./spec/default_spec.rb:8:in `block (2 levels) in <top (required)>'

Finished in 0.00087 seconds (files took 1.83 seconds to load)
1 example, 1 failure
```

WIN!

Revert to working test

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'

describe 'apache::default' do
  chef_run =
ChefSpec::ServerRunner.new.converge('apache::default')

  it 'installs apache2' do
    expect(chef_run).to install_package('httpd')
  end
end
```

SAVE FILE!

Exercise: Execute tests

```
$ rspec --color -fd
```

```
apache::default
  installs apache2

Finished in 0.007 seconds (files took 1.79 seconds
to load)
1 example, 0 failures
```

WIN!

Optimize a little

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'

describe 'apache::default' do
  let(:chef_run) \
  { ChefSpec::ServerRunner.new.converge(described_recipe) }

  it 'installs apache2' do
    expect(chef_run).to install_package('httpd')
  end
end
```

Lazy evaluation

macro

SAVE FILE!

Huh! What does that mean?

- `let` blocks are not evaluated until the first time they're called
- This lets us used the `described_recipe` macro to evaluate and substitute the recipe name.
- Reduces the need to type the recipe name in multiple places

Exercise: Execute tests

```
$ rspec --color -fd
```

```
apache::default
  installs apache2

Finished in 0.00077 seconds (files took 1.79 seconds
to load)
1 example, 0 failures
```

Add a resource report

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'

at_exit { ChefSpec::Coverage.report! }

describe 'apache::default' do
  let(:chef_run) \
  { ChefSpec::ServerRunner.new.converge(described_recipe) }

  it 'installs apache2' do
    expect(chef_run).to install_package('httpd')
  end
end
```

SAVE FILE!

Exercise: Execute tests

```
$ rspec --color -fd
```

```
ChefSpec Coverage report generated...

  Total Resources:   9
  Touched Resources: 1
  Touch Coverage:    11.11%
Untouched Resources:

  service[httpd]                          /recipes/default.rb:14
  execute[mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/
welcome.conf.disabled]   /recipes/default.rb:19
  template[/etc/httpd/conf.d/clowns.conf]   /recipes/default.rb:32
```

create_file matcher

- Our clowns site needs a `clowns.conf` file to work properly
- We want to verify that our recipe is written to create this file

Exercise: Add test for clowns.conf

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'
at_exit { ChefSpec::Coverage.report! }

describe 'apache::default' do
  let(:chef_run) \
  { ChefSpec::ServerRunner.new.converge(described_recipe) }
...
  it 'creates clowns.conf' do
    expect(chef_run).to \
      create_file('/etc/httpd/conf.d/clowns.conf')
  end
end
```

SAVE FILE!

Exercise: Execute tests

```
$ rspec --color -fd
```

```
Finished in 0.03302 seconds (files took 1.81 seconds
to load)
2 examples, 1 failur

Failed exam les:

rspec ./spec/recipes/default_spec.rb:11 #
apache::default creates clowns.conf
```

FAIL!

What just happened?

- Think of ChefSpec as runnable documentation
- The test specifies the `create_file` matcher
- ChefSpec only verifies that you ***told*** Chef to create a file. It doesn't actually change the system and inspect the results
- Our cookbook creates a file, but it uses the `template` resource

Exercise: Fix test for clowns.conf

OPEN IN EDITOR: spec/default_spec.rb

```
require 'chefspec'
at_exit { ChefSpec::Coverage.report! }

describe 'apache::default' do
  let(:chef_run) \
  { ChefSpec::ServerRunner.new.converge(described_recipe) }
...
  it 'creates clowns.conf' do
    expect(chef_run).to \
      create_template('/etc/httpd/conf.d/clowns.conf')
  end
end
```

SAVE FILE!

Exercise: Execute tests

```
$ rspec --color -fd
```

```
apache::default
  installs apache2
  creates clowns.conf

Finished in (.04069 seconds (file) to .79 seconds to load)
2 examples, 0 failures

ChefSpec Coverage report generat...

  Total Resources:   9
  Touched Resources: 2
  Touch Coverage:    22.22%
```

WIN!

spec_helper.rb

- Similar to Serverspec, common code can be moved to a file called `spec_helper.rb`
- ChefSpec recurses through the `spec/*` directories looking for tests. You can create any directory structure you wish
- We'll create `spec/spec_helper.rb`
- We'll create a new directory `spec/recipes` and move `default_spec.rb` there

Exercise: Create spec_helper.rb

OPEN IN EDITOR: spec/spec_helper.rb

```
require 'chefspec'

at_exit { ChefSpec::Coverage.report! }
```

SAVE FILE!

Exercise: Remove common code

OPEN IN EDITOR: spec/default_spec.rb

```
require 'spec_helper'

at_exit { ChefSpec::Coverage.report! }     <— remove this line

describe 'apache::default' do
  let(:chef_run) \
  { ChefSpec::ServerRunner.new.converge(described_recipe) }

  it 'installs apache2' do
    expect(chef_run).to install_package('httpd')
  end
end
```

SAVE FILE!

Exercise: Create recipes directory

```
$ mkdir spec/recipes
$ mv spec/default_spec.rb spec/recipes
```

Exercise: Execute tests

```
$ rspec --color -fd
```

```
apache::default
  installs apache2
  creates clowns.conf

Finished in (.04069 seconds (file) to .79 seconds to load)
2 examples, 0 failures

ChefSpec Coverage report generat...

  Total Resources:   9
  Touched Resources: 2
  Touch Coverage:    22.22%
```

WIN!

Additional Resources

- Many of the the community cookbooks have good examples of ChefSpec. Check out the `spec/` directory in your favorites.

Review Questions

- Where do ChefSpec tests run?
- How can ChefSpec improve your cookbook development process?
- Where does ChefSpec look for tests?
- Who tests their cookbooks in production?

Congratulations!