Intro to Chef

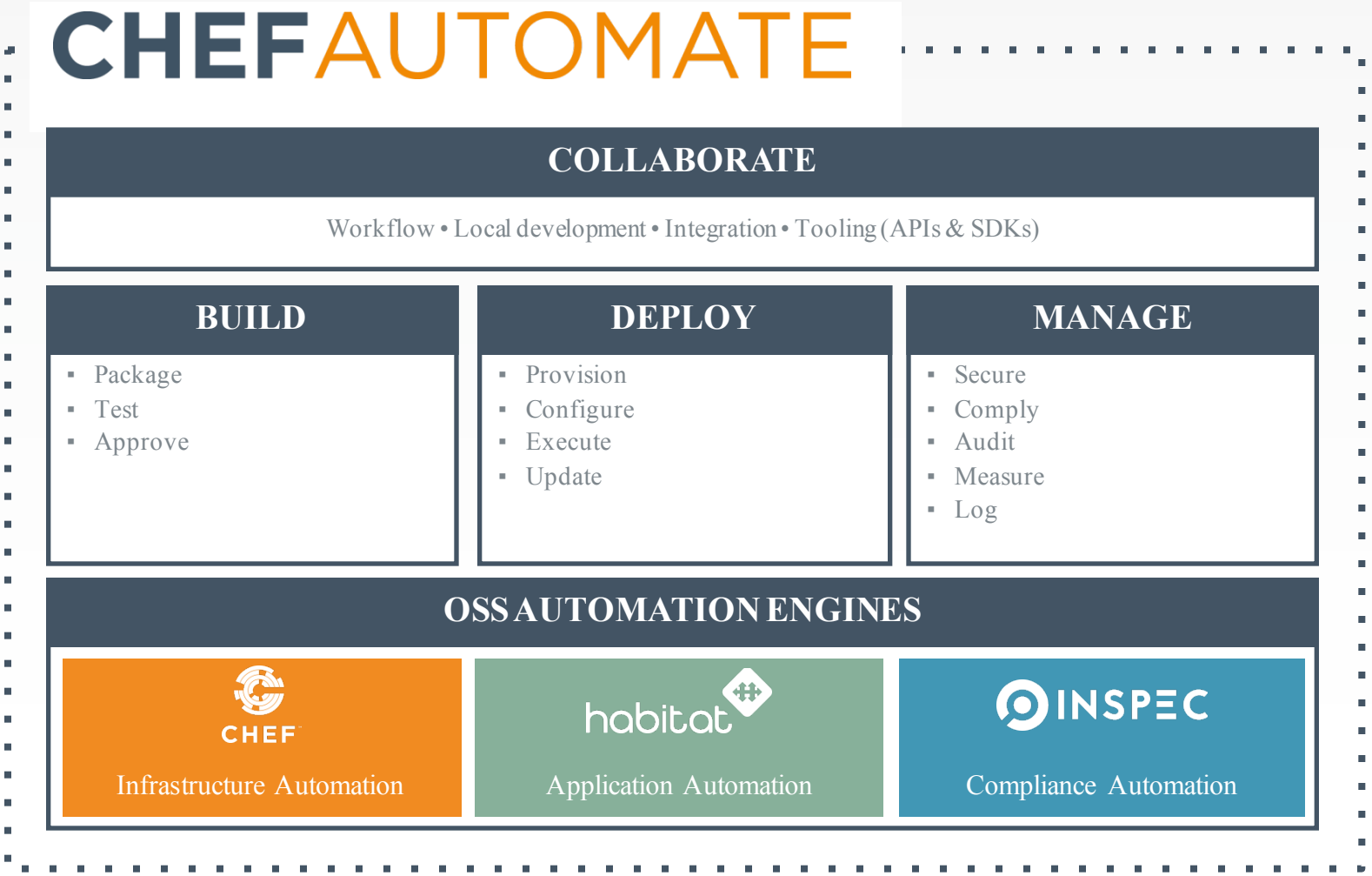
Jessica DeVita, Evangelist

@ubergeekgirl jessica@chef.io

CHEF™

The Chef Automate Platform

Continuous Automation for High Velocity IT



Increase Speed

- Package infrastructure and app configuration as code
- Continuously automate infrastructure and app updates

Improve Efficiency

- Define and execute standard workflows and automation
- Audit and measure effectiveness of automation

Decrease Risk

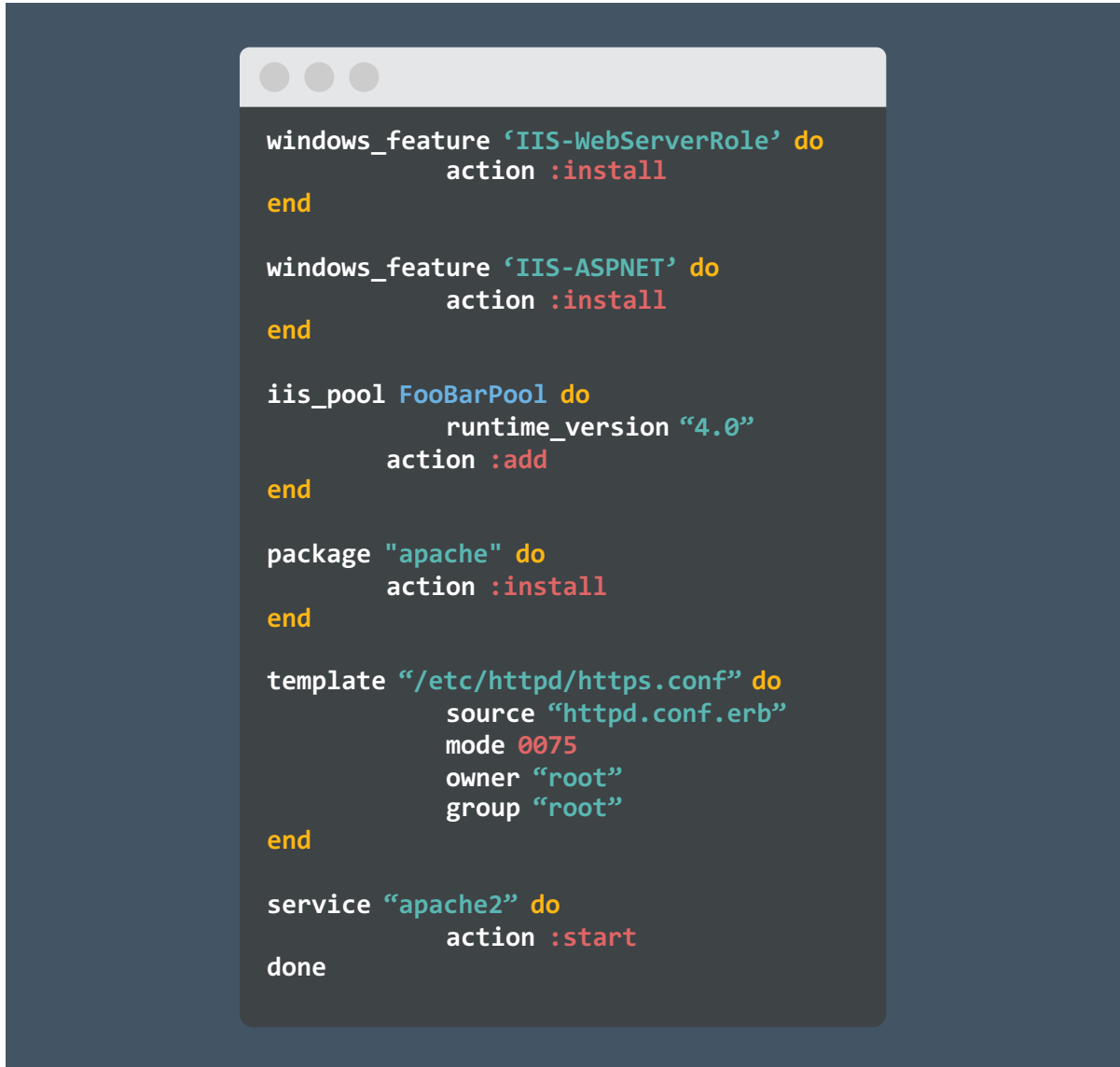
- Define compliance rules as code
- Deliver continuous compliance as part of standard workflow

CHEF

Chef

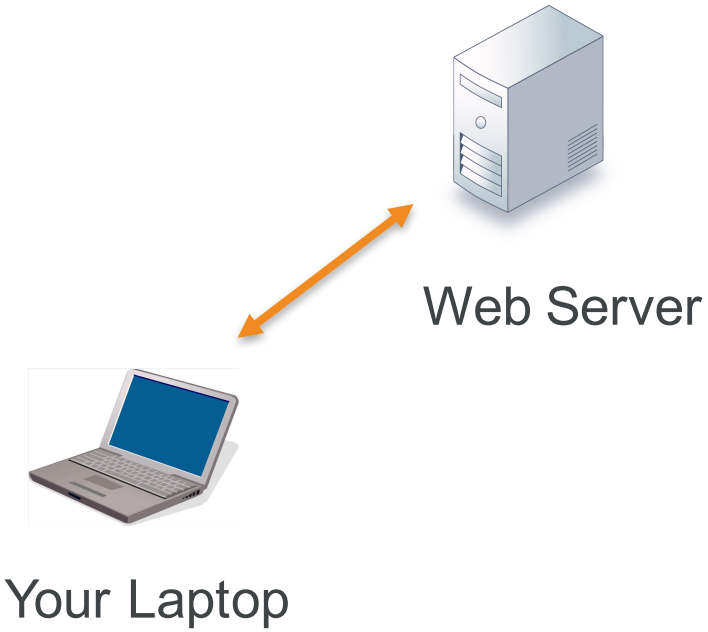
Infrastructure automation and delivery at scale

- Manages **deployment** and on-going automation
- Define **reusable** resources and infrastructure state as code
- Scale elegantly from one to tens of thousands of **managed** nodes across multiple complex environments
- Community, Certified Partner, and Chef supported content available for **all common automation** tasks



Managing Additional Systems

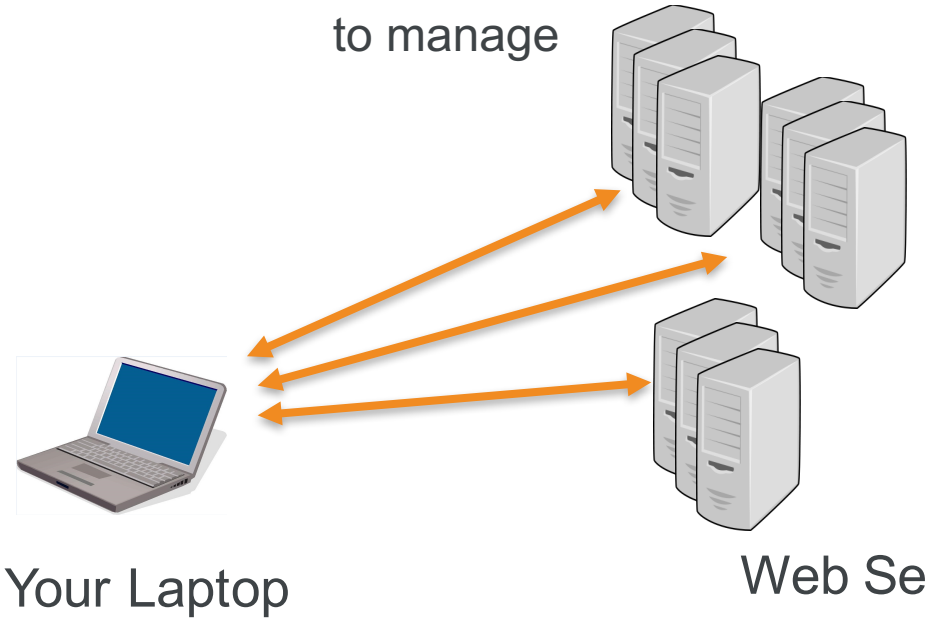
Now



Future

More complex

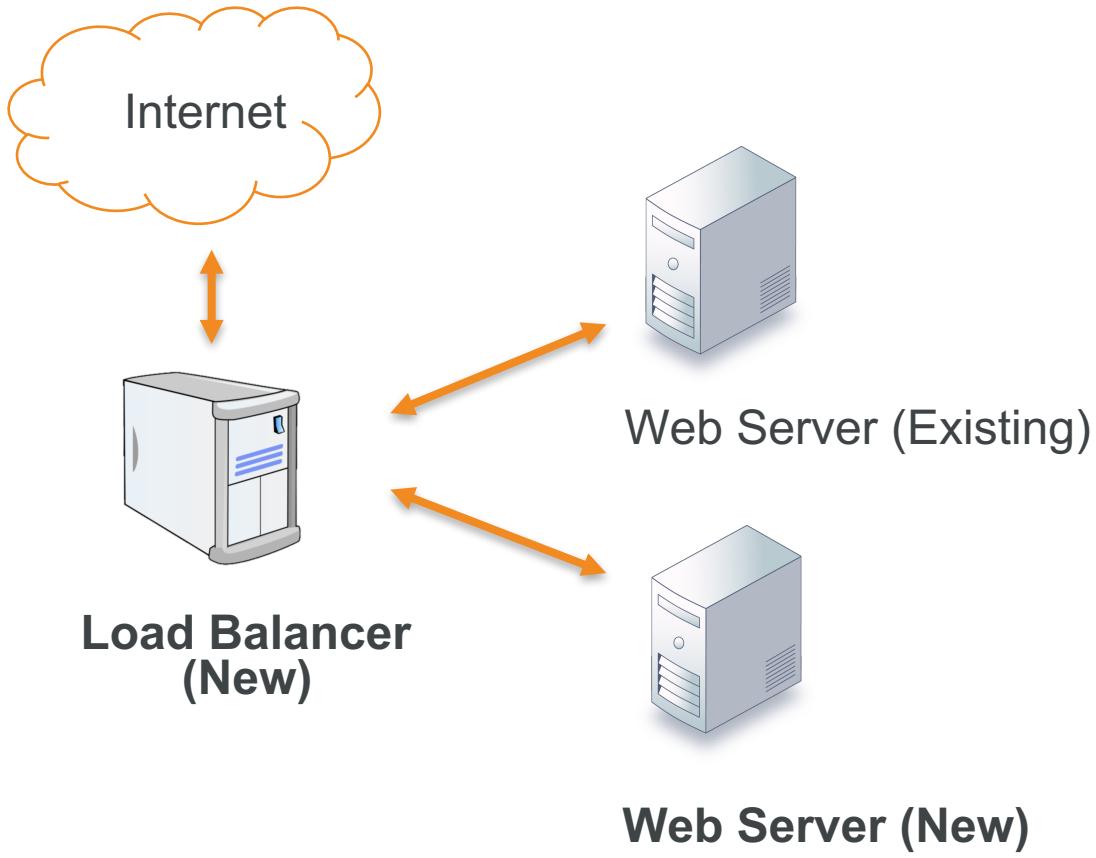
to manage



CHEF

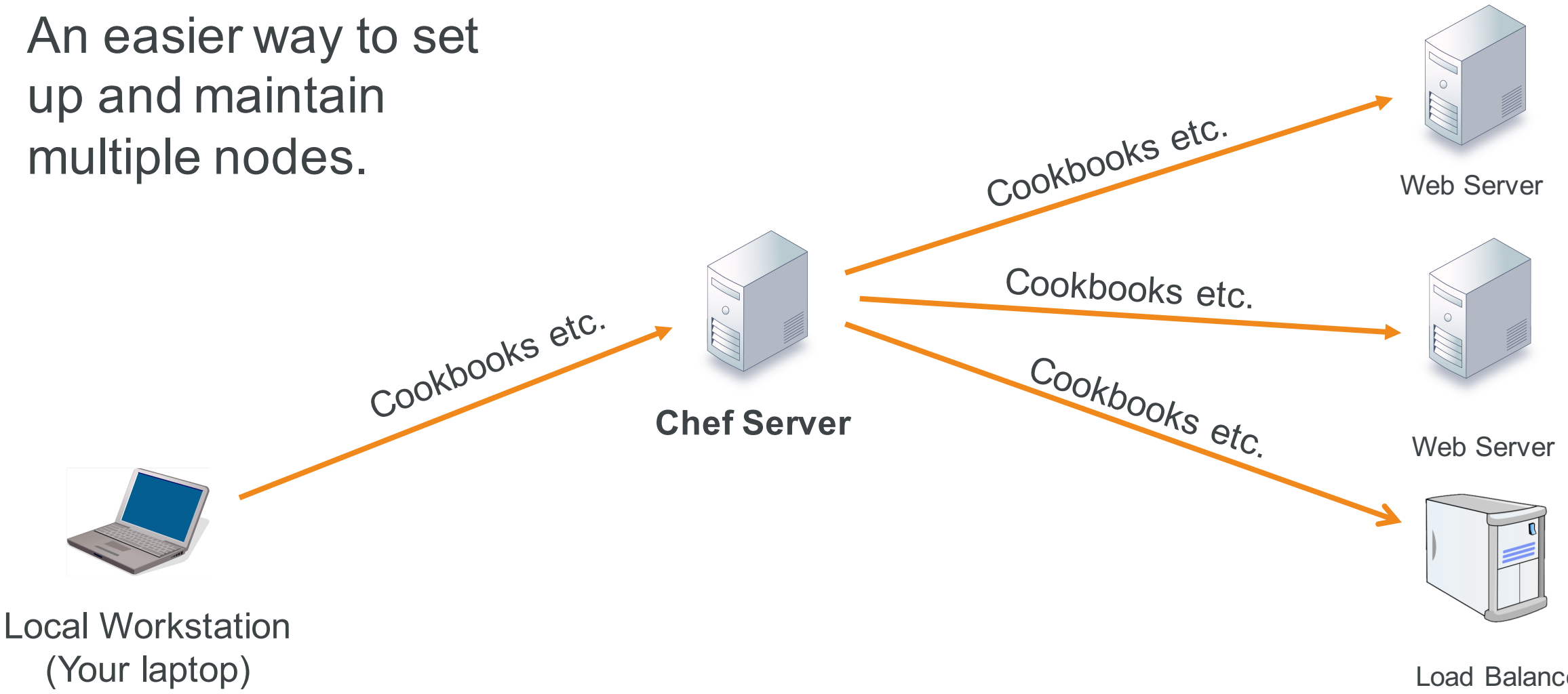
Managing User Traffic

Today you will set up a new load balancer that will direct web requests to similarly-configured nodes.



The Chef Server

An easier way to set up and maintain multiple nodes.

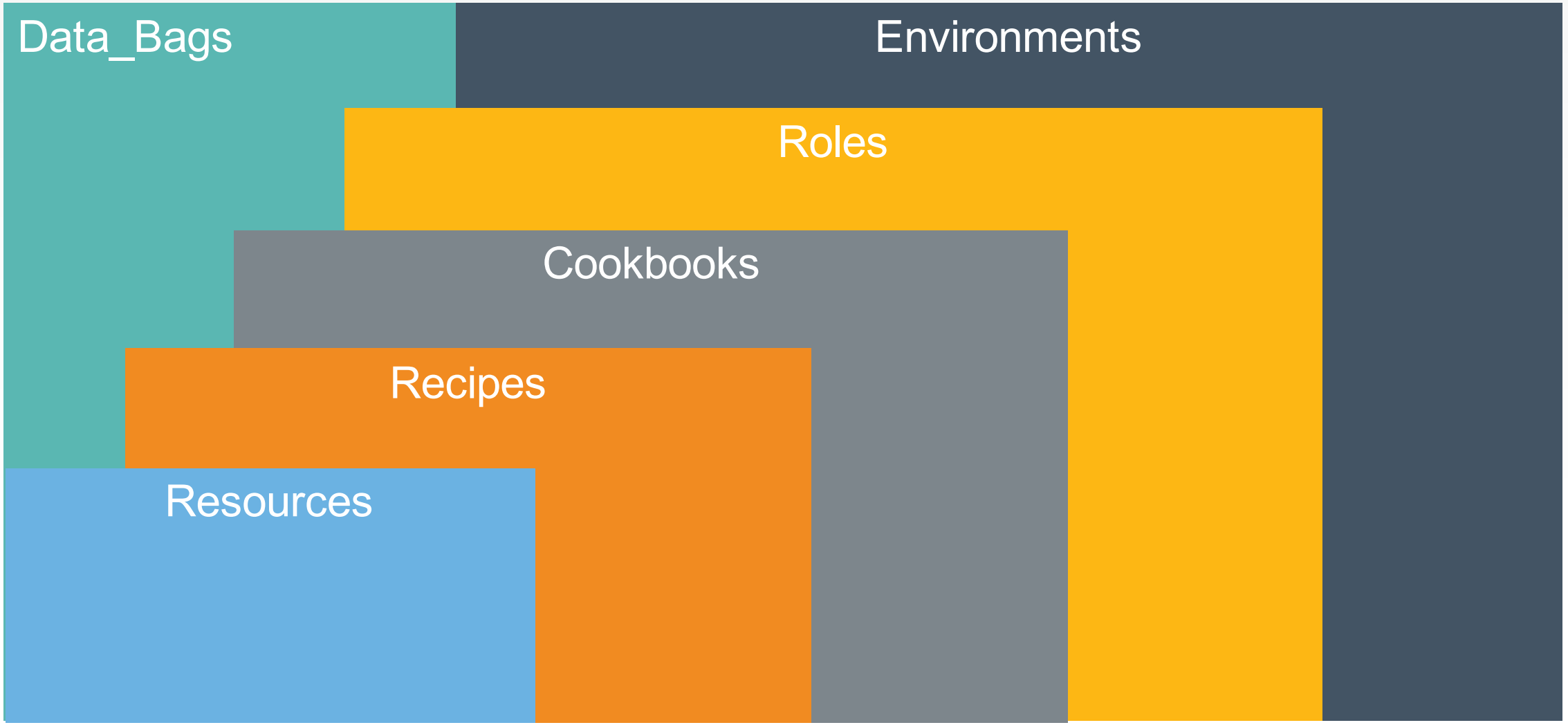


Learning Chef

One of the best ways to learn a technology is to apply the technology in every situation that it can be applied.

A number of chef tools are installed on the system so lets put them to use.

Building Blocks



CHEF

Building Blocks: What is a Resource?

- A Resource is a system state you define

  Example: Package installed, state of a service, configuration file existing

- You declare what the state of the resource is

  Chef automatically determine HOW that state is achieved

```
windows_feature "IIS-WebServerRole" do
  action :install
end
```

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

CHEF

Building Blocks: What is a Recipe?

- A recipe is a collection of Resources
- Resources are executed in the order they are listed

```
windows_feature "IIS-WebServerRole" do
  action :install
end

template 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm.erb"
  rights :read, "Everyone"
end

service "w3svc" do
  action [ :enable, :start ]
end
```

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

template "/var/www/index.html" do
  source "index.html.erb"
  mode "0644"
end

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

CHEF

Building Blocks: What is a Cookbook?

- A cookbook is a set of recipes
- A cookbook is a defined set of items and different outcomes that you expect to address

  A cookbook could have a recipe to install apache2/httpd but also another set of recipes to activate modules required.

```
./attributes
./attributes/default.rb
./CHANGELOG.md
./metadata.rb
./README.md
./recipes
./recipes/application.rb
./recipes/balancer.rb
./recipes/database.rb
./recipes/default.rb
./recipes/webserver.rb
./templates
./templates/default
./templates/default/mysite.conf.erb
```

Building Blocks: What is a role?

- Define reusable roles for Infrastructure Code

```
chef_type:              role
default_attributes:
  my-app:
    application:
      version:      1.5.6
description:            Role for my application
json_class:             Chef::Role
name:                   my_application_role
run_list:
  role[base]
  recipe[my-app::application]
```

CHEF

Building Blocks: What is an Environment?

- Define a reusable environments for Infrastructure Code

```
chef_type:              environment
cookbook_versions:
  database: 2.2.0
default_attributes:
  myapp:
    application:
      version: 1.2.3
description:            Our production environment
json_class:             Chef::Environment
name:                   production
```

CHEF

Building Blocks: What is a Data_Bag?

- Define global variables accessible to all the things

```
deploy_key:
  cipher:         aes-256-cbc
  encrypted_data:
lIpW3sqd69wXt7+MB+uGXr0GfcrEf6rOnHLMA7H00ZCbTxMcEypguGD22w23
  qzEZSzCf2ahv67CtcfrDGvUoWS57Gp5/QoR4WBYKQQUplY0LPtXMZFDISCLU
  a0aNhrzrXhT9eDKNpru7hpuEkOZPRNstx1121bdMZ9lm1/6BPBeIWUYrxAeS
  . . . . . . . . . . . . . . .
  iv:             tpz6zFz9xkscoi36kRw4JQ==
  version:        1
id:         jenkins_ssh_key
```

CHEF

Building Blocks: What is a Resource?

- A Resource is a system state you define

  Example: Package installed, state of a service, configuration file existing

- You declare what the state of the resource is

  Chef automatically determine HOW that state is achieved

```
windows_feature "IIS-WebServerRole" do
  action :install
end
```

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

CHEF

Example: Package

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

The package named 'httpd' is installed.

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

Example: Service

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

The service named 'ntp' is enabled (start on reboot) and started.

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

Example: File

```
file '/etc/motd' do
  content 'This computer is the property...'
end
```

The file name '/etc/motd' is created with content 'This computer is the property ...'

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

Example: File

```
file '/etc/php.ini.default' do
  action :delete
end
```

The file name '/etc/php.ini.default' is deleted.

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

Building Blocks: What is a Recipe?

- A recipe is a collection of Resources
- Resources are executed in the order they are listed

```
windows_feature "IIS-WebServerRole" do
  action :install
end

template 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm.erb"
  rights :read, "Everyone"
end

service "w3svc" do
  action [ :enable, :start ]
end
```

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

template "/var/www/index.html" do
  source "index.html.erb"
  mode "0644"
end

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

CHEF

GL: Use Your Editor to Open the Recipe

```
$ nano moo.rb
```


CHEF

GL: Update the Moo Recipe

~/moo.rb

```
package 'cowsay' do
  action :install
end
```

GL: Apply the Moo Recipe

```
$ sudo chef-client --local-mode moo.rb
```

```
Starting Chef Client, version 12.13.37
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Installing Cookbook Gems:
Compiling Cookbooks...
[2016-08-22T20:20:45+00:00] WARN: Node ip-172-31-9-151.ec2.internal has an empty run list.
Converging 1 resources
Recipe: @recipe_files::/home/chef/moo.rb
  * yum_package[cowsay] action install
    - install version 3.03-8.el6 of package cowsayRunning handlers:

Running handlers complete
Chef Client finished, 1/1 resources updated in 01 minutes 25 seconds
```


CHEF

GL: Run cowsay with a Message

```
$ cowsay will moo for food
```

```
 _____
< will moo for food >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```


CHEF

Discussion

1. What would happen if you applied the recipe again?

2. What would happen if the package were to become uninstalled?

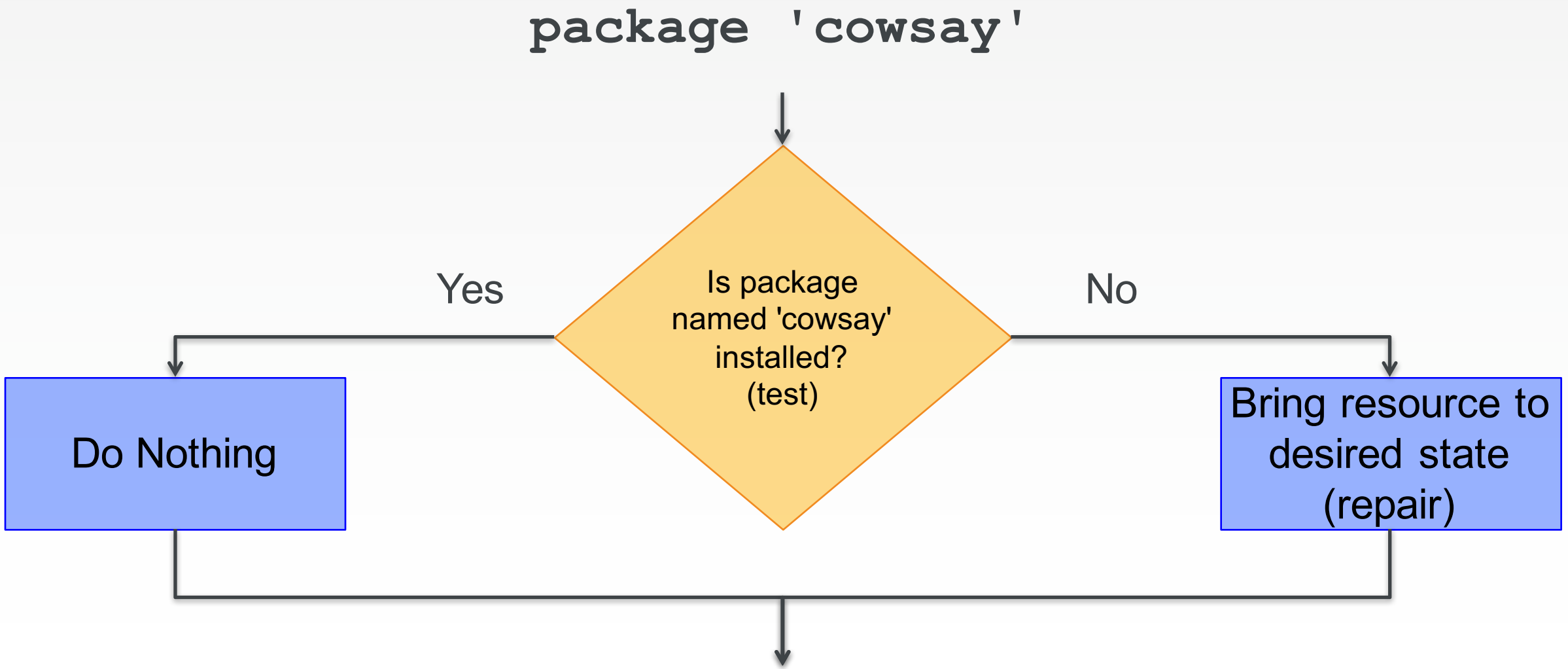
CONCEPT

Test and Repair

`chef-client` takes action only when it needs to. Think of it as test and repair.

Chef looks at the current state of each resource and takes action only when that resource is out of policy.

Test and Repair



Building Blocks: What is a Cookbook?

- A cookbook is a set of recipes
- A cookbook is a defined set of items and different outcomes that you expect to address

  A cookbook could have a recipe to install apache2/httpd but also another set of recipes to activate modules required.

```
./attributes
./attributes/default.rb
./CHANGELOG.md
./metadata.rb
./README.md
./recipes
./recipes/application.rb
./recipes/balancer.rb
./recipes/database.rb
./recipes/default.rb
./recipes/webserver.rb
./templates
./templates/default
./templates/default/mysite.conf.erb
```

GL: Create a Cookbooks Directory

```
$ mkdir cookbooks
```


CHEF

Lab: Create a cookbook called workstation

```
$ chef generate cookbook cookbooks/workstation
Generating cookbook workstation
- Ensuring correct cookbook file content
- Committing cookbook files to git
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content
- Adding delivery configuration to feature branch
- Adding build cookbook to feature branch
- Merging delivery content feature branch to master

Your cookbook is ready. Type `cd cookbooks/workstation` to enter it.

There are several commands you can run to get started locally developing and
testing your cookbook.
Type `delivery local --help` to see a full list.
```

Lab: Create a recipe for workstation and call it setup

```
$ chef generate recipe cookbooks/workstation setup
```


CHEF

Lab: Use a text editor to edit the setup recipe

~/cookbooks/workstation/setup.rb

```
package 'tree' do
  action :install
end

file '/etc/motd' do
  content 'Property of
...'
end
```

The package named 'tree' is installed.

The file named '/etc/motd' is created with the content 'Property of ...'.

GL: CD into the workstation Cookbook Directory

```
$ cd cookbooks/workstation
```


CHEF

GL: Apply the Recipe File

```
$ sudo chef-client -z setup.rb
```

```
Converging 2 resources
Recipe: @recipe_files::/home/chef/setup.rb
  * yum_package[tree] action install
    - install version 1.5.3-3.el6 of package tree
  * file[/etc/motd] action create
    - update content in file /etc/motd from e3b0c4 to d100eb
    --- /etc/motd       2010-01-12 13:28:22.000000000 +0000
    +++ /etc/.motd20160224-8754-1xczeyn 2016-02-24 16:57:57.203844958 +0000
    @@ -1 +1,2 @@
    +Property of ...
Running handlers:
Running handlers complete
Chef Client finished, 2/2 resources updated in 17 seconds
```

Lab: Create a cookbook called apache

```
$ chef generate cookbook cookbooks/apache
```

```
Generating cookbook apache
- Ensuring correct cookbook file content
- Committing cookbook files to git
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content
- Adding delivery configuration to feature branch
- Adding build cookbook to feature branch
- Merging delivery content feature branch to master

Your cookbook is ready. Type `cd cookbooks/apache` to enter it.

There are several commands you can run to get started locally developing and
testing your cookbook.
Type `delivery local --help` to see a full list.
```

Lab: Create a recipe for apache and call it server

```
$ chef generate recipe cookbooks/apache server
Compiling Cookbooks...
Recipe: code_generator::recipe
  * directory[cookbooks/apache/spec/unit/recipes] action create (up to date)
  * cookbook_file[cookbooks/apache/spec/spec_helper.rb] action create_if_missing
(up to date)
  * template[cookbooks/apache/spec/unit/recipes/server_spec.rb] action
create_if_missing
    - create new file cookbooks/apache/spec/unit/recipes/server_spec.rb
    - update content in file cookbooks/apache/spec/unit/recipes/server_spec.rb
from none to a43970
    (diff output suppressed by config)
  * template[cookbooks/apache/recipes/server.rb] action create
    - create new file cookbooks/apache/recipes/server.rb
    - update content in file cookbooks/apache/recipes/server.rb from none to
3d6b92
```

Lab: Use a text editor to edit the server recipe

~/cookbooks/apache/recipes/server.rb

```
package 'httpd'

file '/var/www/html/index.html' do
  content '<h1>Hello, world!</h1>'
end

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

Lab: Apply the Server Recipe

```
$ sudo chef-client -z cookbooks/apache/recipes/server.rb
```

```
Converging 3 resources
Recipe: @recipe_files::/home/chef/cookbooks/apache/recipes/server.rb
  * yum_package[httpd] action install
    - install version 2.2.15-47.el6.centos.3 of package httpd
  * file[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to 17d291
    --- /var/www/html/index.html    2016-02-24 21:41:45.494844958 +0000
    +++ /var/www/html/.index.html20160224-10036-6y8on7    2016-02-24
21:41:45.493844958 +0000
    @@ -1 +1,2 @@
    +<h1>Hello, world!</h1>
  * service[httpd] action enable
    - enable service service[httpd]
```

Demo: The Cookbook Has a Folder for Recipes

```
$ tree cookbooks/workstation
```

```
workstation
├── Berksfile
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
| ...
6 directories, 8 files
```

GL: The Cookbook Has a Default Recipe

```
$ cat cookbooks/workstation/recipes/default.rb
```

```
# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
```

DOCS

include_recipe

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

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

CHEF

GL: The Default Recipe Includes the Setup Recipe

~/cookbooks/workstation/recipes/default.rb

```
#
# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.

include_recipe 'workstation::setup'
```

Lab: The Default Recipe Includes the Apache Recipe

~/cookbooks/apache/recipes/default.rb

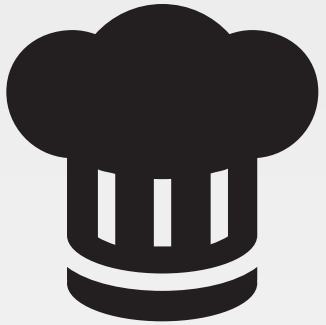
```
#
# Cookbook Name:: apache
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.

include_recipe 'apache::server'
```

Managing a Large Number of Servers

Have you ever had to manage a large number of servers that were almost identical?

How about a large number of identical servers except that each one had to have host-specific information in a configuration file?



Details About the Node

Displaying system details in the MOTD definitely sounds useful.

Objective:

- ❑ Update the MOTD file contents, in the "workstation" cookbook, to include node details

Demo/EXAMPLE: Adding the CPU

~/cookbooks/workstation/recipes/setup.rb

```
file '/etc/motd' do
  content 'Property of ...


  IPADDRESS: 104.236.192.102
  HOSTNAME : banana-stand
  MEMORY   : 502272 kB
  CPU      : 2399.998 MHz
'
  mode '0644'
  owner 'root'
  group 'root'
end
```

Hard Coded Values

The values that we have derived at this moment may not be the correct values when we deploy this recipe again even on the same system!

Data In Real Time

How could we capture this data in real-time?

CONCEPT

Ohai!

Ohai is a tool that already captures all the data that we similarly demonstrated finding.

http://docs.chef.io/ohai.html

Ohai!

```
$ ohai
```


```
{
  "kernel": {
    "name": "Linux",
    "release": "2.6.32-431.1.2.0.1.el6.x86_64",
    "version": "#1 SMP Fri Dec 13 13:06:13 UTC 2013",
    "machine": "x86_64",
    "os": "GNU/Linux",
    "modules": {
      "veth": {
        "size": "5040",
        "refcount": "0"
      },
      "ipt_addrtype": {
```


CHEF

CONCEPT

ohai + chef-client = <3

chef-client and chef-apply automatically executes ohai and stores the data about the node in an object we can use within the recipes named node.

http://docs.chef.io/ohai.html

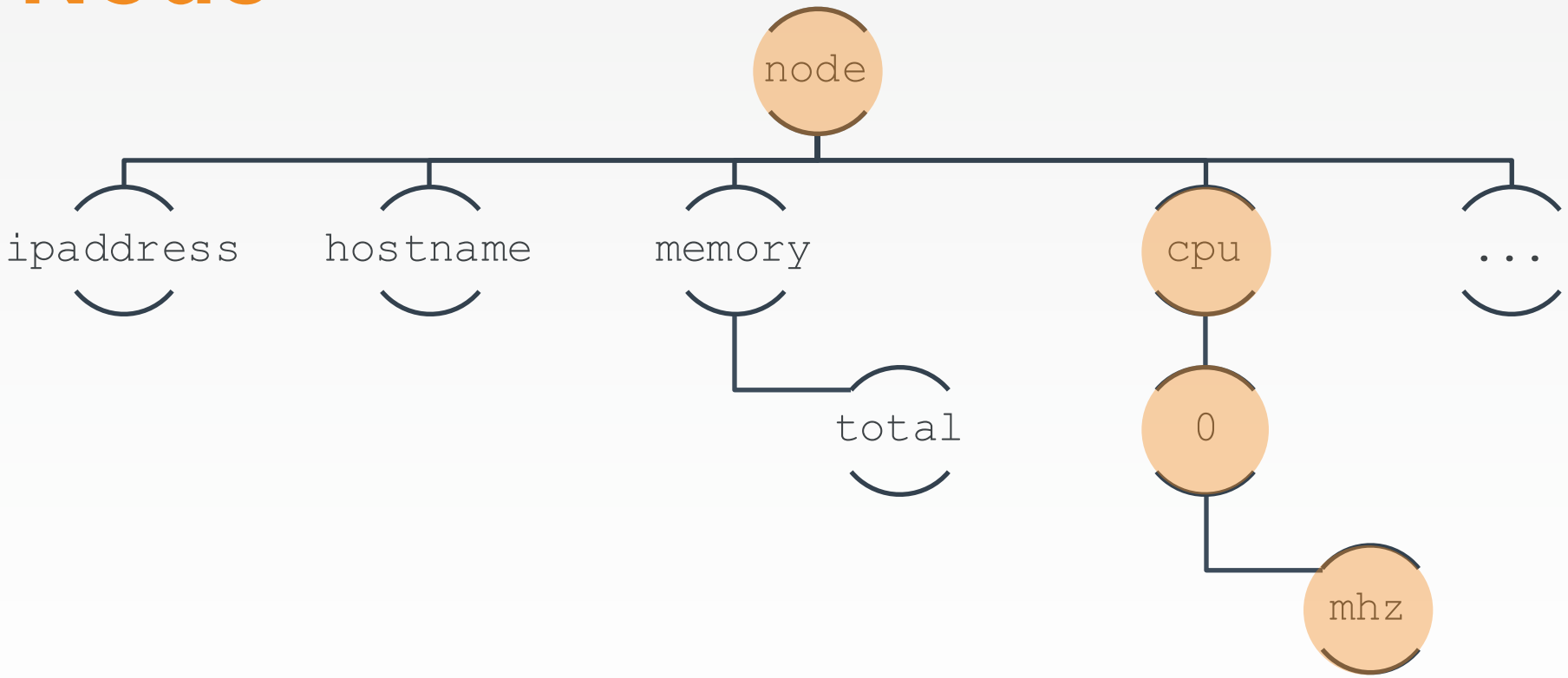
CONCEPT

The Node Object

The node object is a representation of our system.
It stores all the attributes found about the system.

http://docs.chef.io/nodes.html#attributes

The Node




```
CPU: 2399.998MHz
```

```
"CPU: #{node['cpu']['0']['mhz']}"
```

GL: Update recipe with the Node's Attributes

~/cookbooks/workstation/recipes/setup.rb

```
# ... PACKAGE RESOURCES ...
file '/etc/motd' do
  content "Property of ...

  IPADDRESS: #{node['ipaddress']}
  HOSTNAME : #{node['hostname']}
  MEMORY   : #{node['memory']['total']}
  CPU      : #{node['cpu']['0']['mhz']}
"
  mode '0644'
  owner 'root'
  group 'root'
end
```

Note the changes to "Property of..."

GL: The Cookbook Has Some Metadata

```
$ tree cookbooks/workstation
```

```
workstation
├── Berksfile
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
| ...
6 directories, 8 files
```

GL: Let's Take a Look at the Metadata

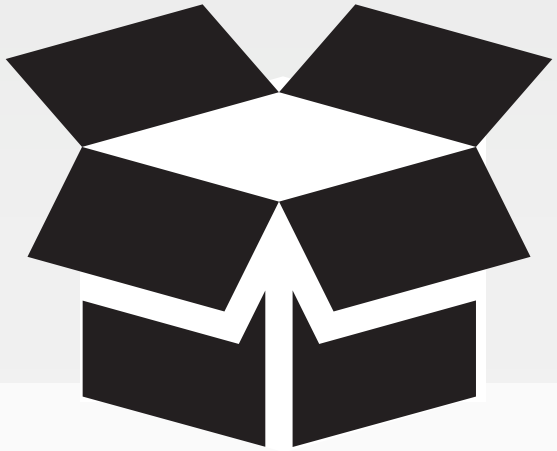
```
$ cat cookbooks/workstation/metadata.rb
```

```
name             'workstation'
maintainer       'The Authors'
maintainer_email 'you@example.com'
license          'all_rights'
description      'Installs/Configures workstation'
long_description 'Installs/Configures workstation'
version          '0.1.0'

# If you upload to Supermarket you should set this so your cookbook
# gets a `View Issues` link
# issues_url 'https://github.com/<insert_org_here>/workstation/issues' if
respond_to?(:issues_url)
```

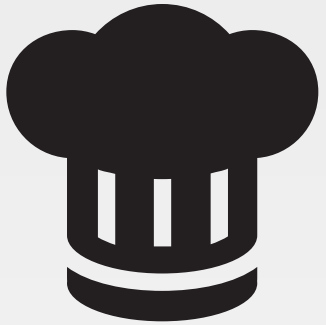
CONCEPT

Cookbook Versions



A cookbook version represents a set of functionality that is different from the cookbook on which it is based.

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



Changes Mean a New Version

Let's bump the version number and check in the code to source control.

Objective:

- ❑ Update the version of the "workstation" cookbook
- ❑ Commit the changes to the "workstation" cookbook to version control

CONCEPT

Semantic Versions

Given a version number **MAJOR**.**MINOR**.**PATCH**, increment the:

- **MAJOR** version when you make incompatible API changes
- **MINOR** version when you add functionality in a backwards-compatible manner
- **PATCH** version when you make backwards-compatible bug fixes

http://semver.org

Major, Minor, or Patch?

What kind of changes did you make to the cookbook?

GL: Update the Cookbook Version

~/cookbooks/workstation/metadata.rb

```
name              'workstation'
maintainer        'The Authors'
maintainer_email  'you@example.com'
license           'all_rights'
description       'Installs/Configures
workstation'
long_description  'Installs/Configures
workstation'
version           '0.2.0'
```

Lab: Verify That the Website is Available

```
$ curl localhost
```

```
<h1>Hello, world!</h1>
```


CHEF

Demo: Using 'chef-client' to Locally Apply Recipes

```
$ sudo chef-client --local-mode \
  -r "recipe[workstation::setup],recipe[apache::server]"
```

Applying the following recipes locally:

- The 'setup' recipe from the 'workstation' cookbook
- The 'server' recipe from the 'apache' cookbook

GL: Apply Both Recipes Locally

```
$ sudo chef-client --local-mode \
  -r "recipe[apache::server],recipe[workstation::setup]"
```

```
[2016-09-15T15:17:27+00:00] WARN: No config file found or specified on
command line, using command line options.
Starting Chef Client, version 12.13.37
resolving cookbooks for run list: ["apache::server","workstation::setup"]
Synchronizing Cookbooks:
  - apache
  - workstation
Compiling Cookbooks...

Running handlers:
...
```


CHEF

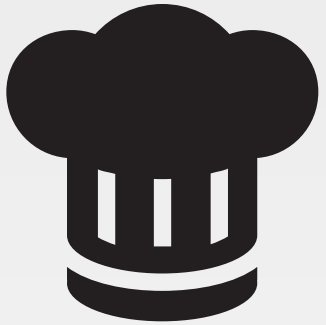
Lab: Applying the apache Default Recipe

```
$ sudo chef-client --local-mode -r "recipe[apache]"
```

```
[2016-09-15T15:23:18+00:00] WARN: No config file found or specified on command
line, using command line options.
Starting Chef Client, version 12.13.37
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache (0.1.0)
Compiling Cookbooks...
Converging 3 resources

Running handlers:
Running handlers complete
Chef Client finished, 0/3 resources updated in 3.310768509 seconds
```


CHEF



Changes Mean a New Version

Let's bump the version number

Objective:

- ❑ Update the version of the "workstation" cookbook

Discussion

Why would you want to apply more than one recipe at a time?

What are the benefits and drawbacks of using "include_recipe" within a recipe?

Do default values make it easier or harder to learn?

Q&A

What questions can we help you answer?

- chef-client
- local mode
- run list
- include_recipe

Testing Cookbooks

Validating Our Recipes in Virtual Environments

CHEF

Remember…

Chef recipes need testing

- ↳ Linting
- ↳ Static Analysis
- ↳ Unit Testing
- ↳ Integration Testing

“Infrastructure as Code” should be tested like ANY other codebase.

Chef DK - The Chef Development Kit

Definitive tooling for local development of Chef code & Infrastructure as Code development

FAST INEXPENSIVE TESTING

Foodcritic

Test Your "Chef Style"

- Validate your Chef code against Chef best practices
- Extend with rules to enforce organizational Chef development best practices
- Enforce compliance & security practices

CookStyle

Validate your Ruby

- Validate your Chef code against Ruby best practices
- Identify potential Ruby errors
  Unclosed strings, etc.
- Identify style/convention that helps write better code
  Single quotes vs. double quotes

ChefSpec

Simulate Chef

- Validate your Chef code will run
- Testing for more Chef advanced use cases
- Useful for regression testing

DEEP INTEGRATION TESTING

Test Kitchen

Let's do this (almost) for real

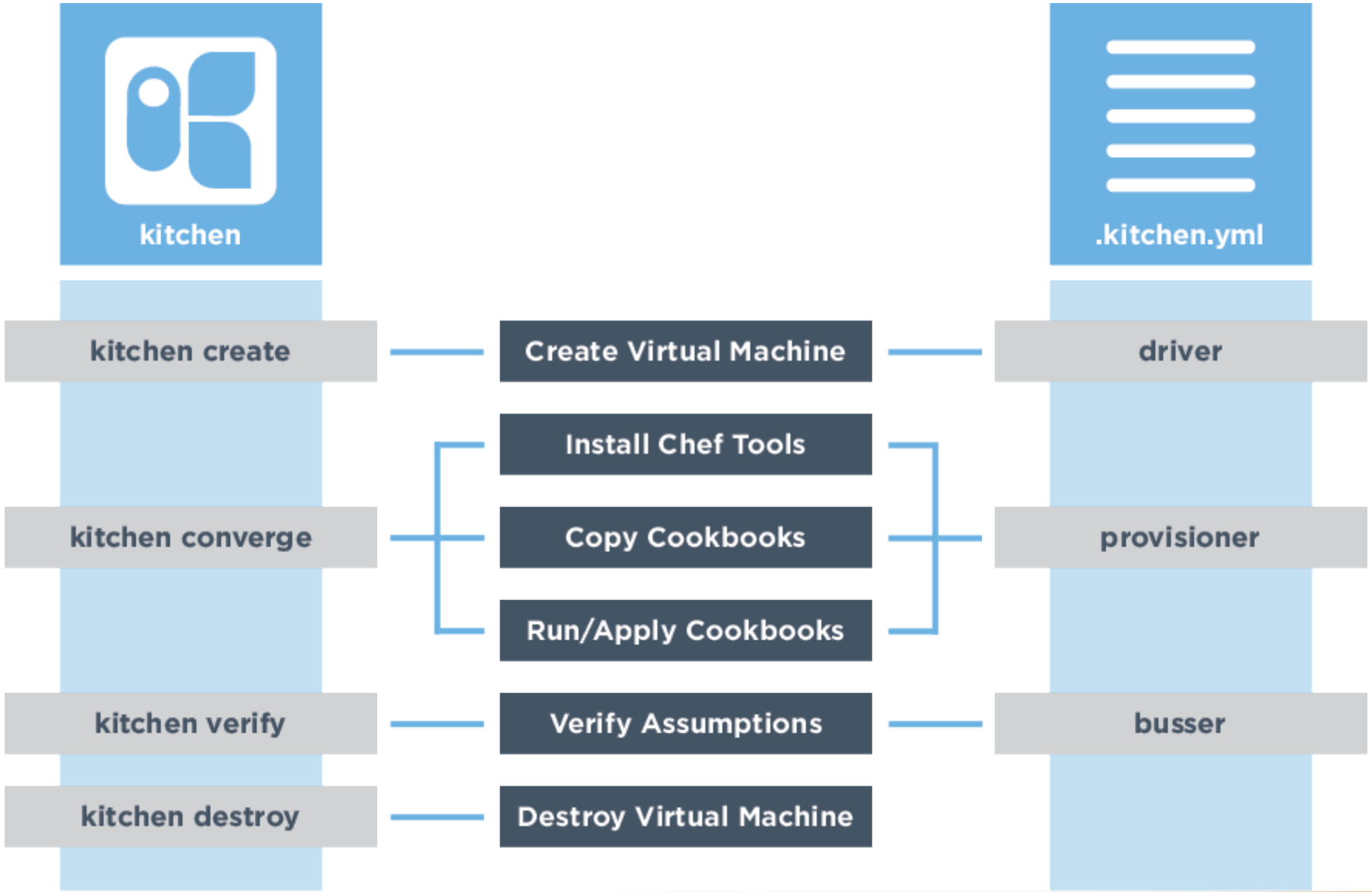
- Executes your Chef code on an instance or container
- Integrates with Cloud and Virtualization providers
- Validate your Chef code locally before sharing
- Speed development of Chef Cookbooks

InSpec

Verify automation results & ensure compliance

- Assert the intention of your Chef code
- Verify on live systems that your Chef code produced the correct result
- Confirm your Chef code didn't not produce compliance drift

Test Kitchen Commands and Configuration



CHEF

What Can 'kitchen' Do?

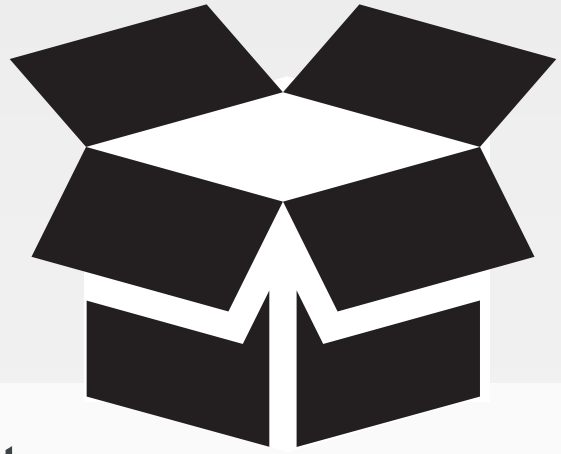
```
$ kitchen --help
```

```
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
```


CHEF

CONCEPT

.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

Each cookbook has a .kitchen.yml

```
$ tree cookbooks/workstation -a
```

```
...
├── .kitchen.yml
├── metadata.rb
├── README.md
├── recipes
│   ├── default.rb
│   └── setup.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
│           └── default_spec.rb
└── test
```


CHEF

What is Inside .kitchen.yml?

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

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

verifier:
  name: inspec

platforms:
  - name: ubuntu-16.04
  - name: centos-7.2
```


CHEF

GL: Edit the Workstation Kitchen Configuration File

~/cookbooks/workstation/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

verifier:
  name: inspec

platforms:
  - name: centos-6.7
```

tech/kitchen-docker

CHEF

Lab: Configuring Test Kitchen for Apache

~/cookbooks/apache/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

verifier:
  name: inspec

platforms:
  - name: centos-6.7
```

CHEF

GL: Converge the Cookbook

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.11.0)
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  193 kB
(skipping)
----->  Finished creating <default-centos-67> (1m18.32s).
-----> Converging <default-centos-67>...
$$$$$$ Running legacy converge for 'Docker' Driver
(skipping)
Synchronizing Cookbooks:
         - workstation (0.1.0)
       Compiling Cookbooks...
       Converging 2 resources
       Running handlers:
```


CHEF

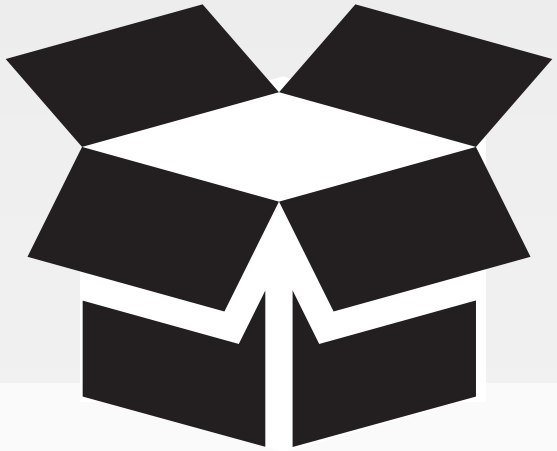
Lab: Converge the Cookbook

```
$ kitchen converge
```

```
-----> Starting Kitchen (v1.11.0)
-----> Creating <default-centos-67>...
       Sending build context to Docker daemon  194 kB
       Sending build context to Docker daemon
(skipping)
 Installing Chef
       installing with rpm...
       warning: /tmp/install.sh.23/chef-12.13.37-1.el6.x86_64.rpm: Header V4
DSA/SHA1 Signature, key ID 83ef826a: NOKEY
(skipping)
       Synchronizing Cookbooks:
         - apache (0.1.0)
       Compiling Cookbooks...
```


CHEF

ALL OF THE KITCHEN COMMANDS



Kitchen Test


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

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

CONCEPT

Kitchen Create


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

Create one or more instances.
```

CONCEPT

Kitchen Converge


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

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

CHEF

CONCEPT

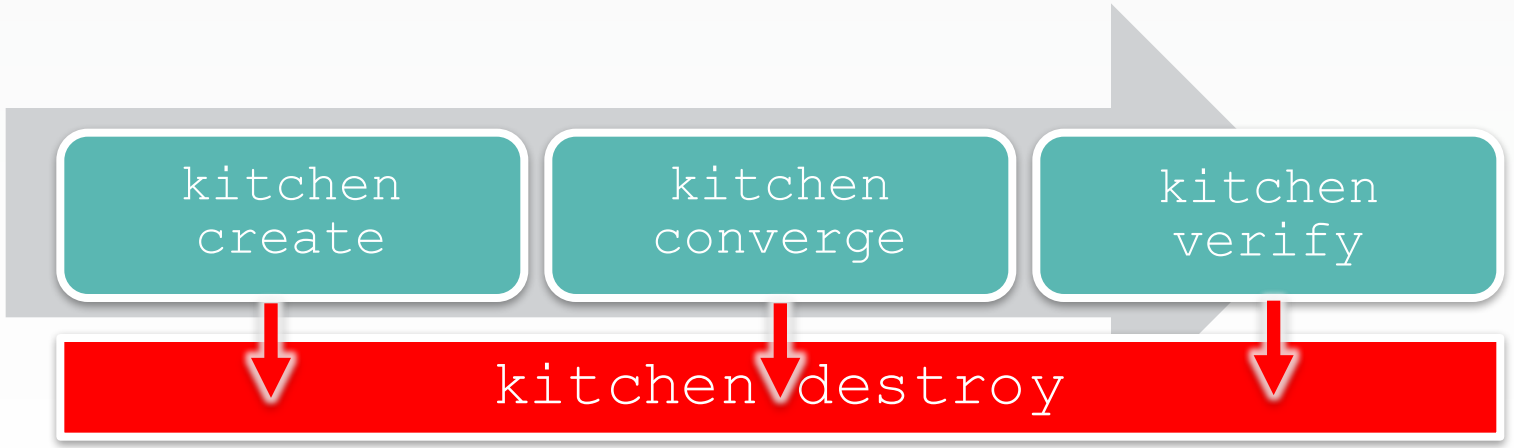
Kitchen Verify


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

Create, converge, and verify one or more instances.

CONCEPT

Kitchen Destroy

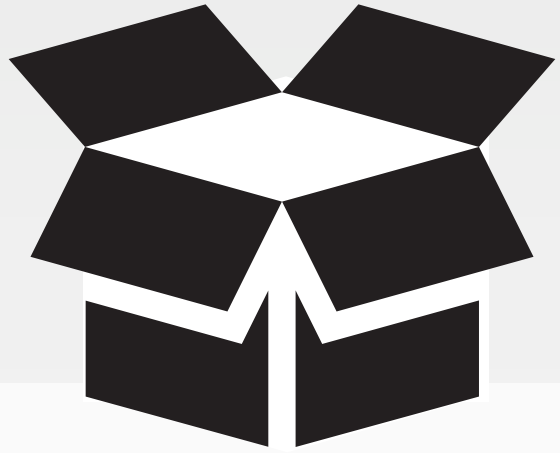



```
$ kitchen destroy [INSTANCE|REGEXP|all]
```

Destroys one or more instances.

CONCEPT

Where do Tests Live?



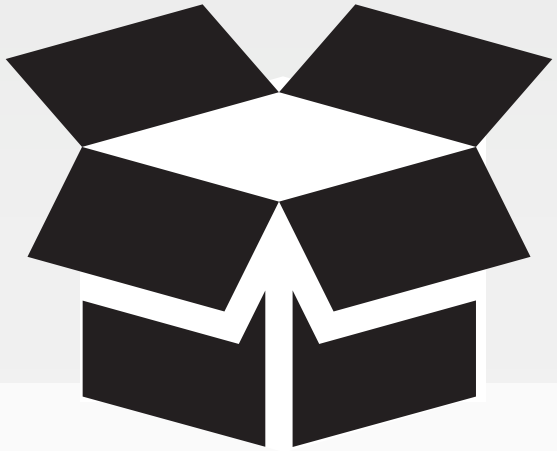
`workstation/test/recipes/default_test.rb`

Test Kitchen will look for tests to run under this directory.

This is configurable in the kitchen configuration file (.kitchen.yml) in the suites section.

CONCEPT

InSpec



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

http://inspec.io/

Example: Is the 'tree' Package Installed?

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

I expect the package tree should be installed.

http://inspec.io/docs/reference/resources/#id118

CHEF

GL: Example Tests

~/cookbooks/workstation/test/recipes/default_test.rb

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end
```

When not on windows a user named `root` should exist.

The port 80 should not be listening for incoming connections.

GL: Describing the Resources

~/cookbooks/workstation/test/recipes/default_test.rb

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end
```

A user named 'root'

The port 80

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

CHEF

GL: Describing the State of the Resources

~/cookbooks/workstation/test/recipes/default_test.rb

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end
```

The user named 'root' should exist.

The port 80 should not be listening.

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

CHEF

GL: The `skip` Reminds Us to Remove These Tests

~/cookbooks/workstation/test/recipes/default_test.rb

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end
```

skip will show a message in the test results.

These skips will remind you that the following expectations are only examples.

GL: Adding a New Test

~/cookbooks/workstation/test/recipes/default_test.rb

```
unless os.windows?
  describe user('root') do
    it { should exist }
    skip 'This is an example...test.'
  end
end

describe port(80) do
  it { should_not be_listening }
  skip 'This is an example... test.'
end

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

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/recipes/default_test.rb

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

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

describe file('/etc/motd') do
  its('content') { should match(/Property of/) }
end
```

Lab: Our Assertion in a spec File

~/cookbooks/workstation/test/recipes/default_test.rb

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

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

describe file('/etc/motd') do
  its('content') { should match(/Property of/) }
  it { should be_owned_by 'root' }
end
```

Lab: What Does the Webserver Say?

~/cookbooks/apache/test/recipes/default_test.rb

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

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

Lab: What Does the Webserver Say?

~/cookbooks/apache/test/recipes/default_test.rb

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

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

Lab: Remove the server test file

```
$ rm test/recipes/server.rb
```


CHEF

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?

Q&A

What questions can we help you answer?

- Test Kitchen
- kitchen commands
- kitchen configuration
- InSpec

CHEF


MAY 22-24 | AUSTIN
CHEFCONF
2017
chefconf.chef.io
REGISTER NOW
$995 EARLY
BIRD PRICE
ENDS MARCH 31st