



# Introduction to Chef

Cookbook development workflow



Chef Fundamentals by [Chef Software, Inc.](#) is licensed under a  
[Creative Commons Attribution-ShareAlike 4.0 International License](#).

v0.1.3



# Prerequisites

- ssh client
  - good text editor (Atom, Sublime, vim, emacs)
  - Vagrant & Virtual Box
  - git
  - Chef Development Kit
- 
- Vagrant .box file



# Introductions

v0.1.3



# Instructor

- Nathon Harvey
- Community Director, Chef
- Co-host of the Food Fight Show
- Co-organizer of DevOpsDC Meetup
- Chair of DevOps Days DC



# Course Objectives & Style



v0.1.3

# Course Objectives

- After completing this course you will be able to:
  - Automate common infrastructure tasks with Chef
  - Verify your automation code BEFORE it runs in production
  - Describe some of Chef's tools
  - Apply Chef's primitives to solve your problems

# Learning Chef

- You bring the domain expertise about your business and problems
- Chef provides a framework for solving those problems
- Our job is to work together to help you express solutions to your problems with Chef

# Chef is a Language

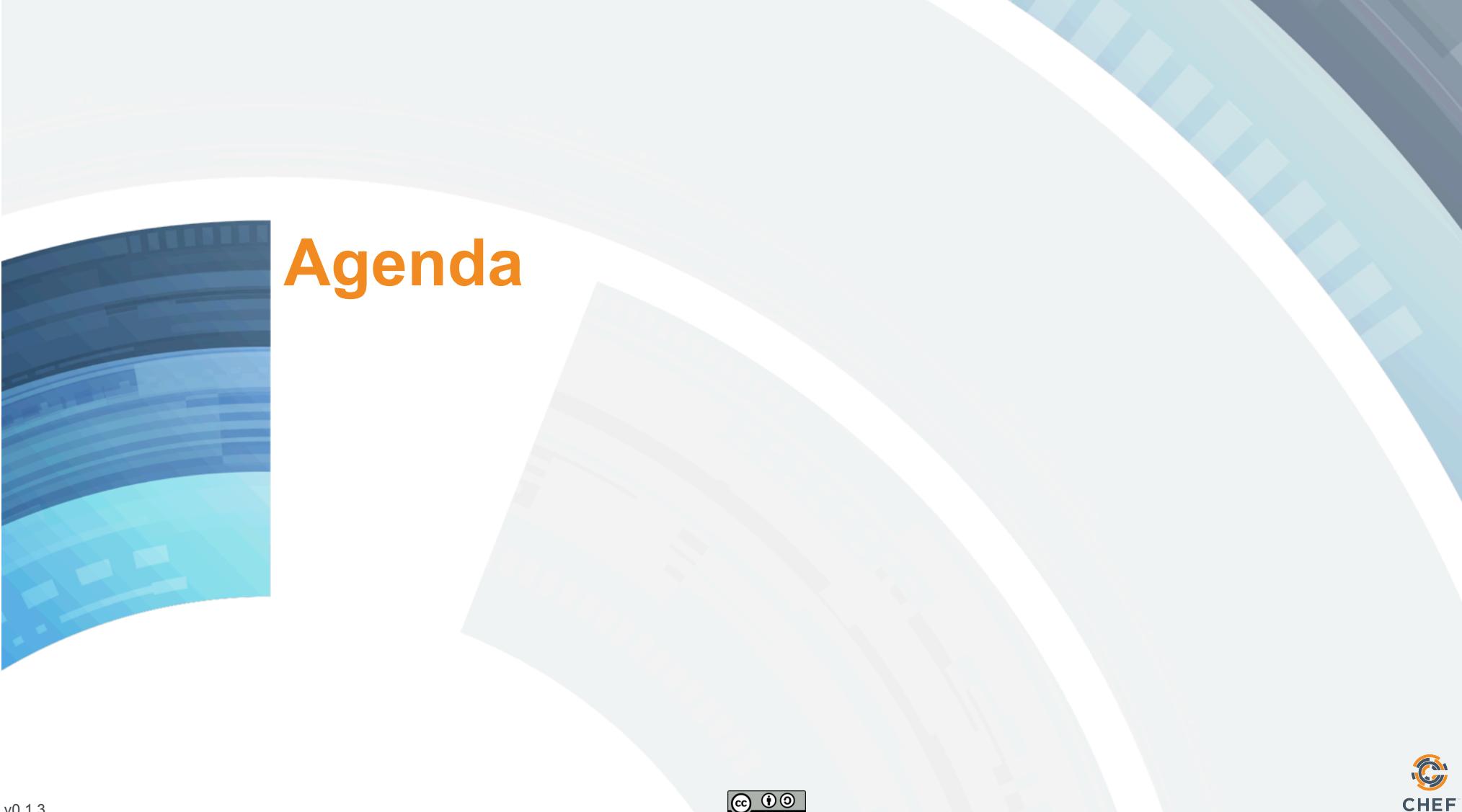
- Learning Chef is like learning the basics of a language
  - 80% fluency reached quickly
  - 20% just takes practice
- The best way to **LEARN** Chef is to **USE** Chef

# Training is a discussion

- Lots of hands on labs
- Lots of typing
- Ask questions when they come to you
- Ask for help when you need it
- Help each other
- We will troubleshoot and fix bugs on the spot

# Just an Introduction

- Today is just an Introduction to testing your automation code with Chef and it's tools
- We'll cover lots of topics but won't go too deep on any of them
- Any discussion that takes us too far off the path will be captured
- We will return to these topics as time permits



# Agenda

v0.1.3



# Agenda

- Overview of Chef
- Resources
- Describing Policies
- A Sandbox for testing
- Verifying node state
- Even faster feedback
- Clean code
- Wrap Up

# Breaks!

- We will take breaks as often as we need them
- We will break for lunch

# Prerequisites

- ssh client
  - good text editor (Atom, Sublime, vim, emacs)
  - Vagrant & Virtual Box
  - git
  - Chef Development Kit
- 
- Vagrant .box file



# Overview of Chef

## Policy-based Infrastructure as Code

v0.1.3



# Benefits of Automation



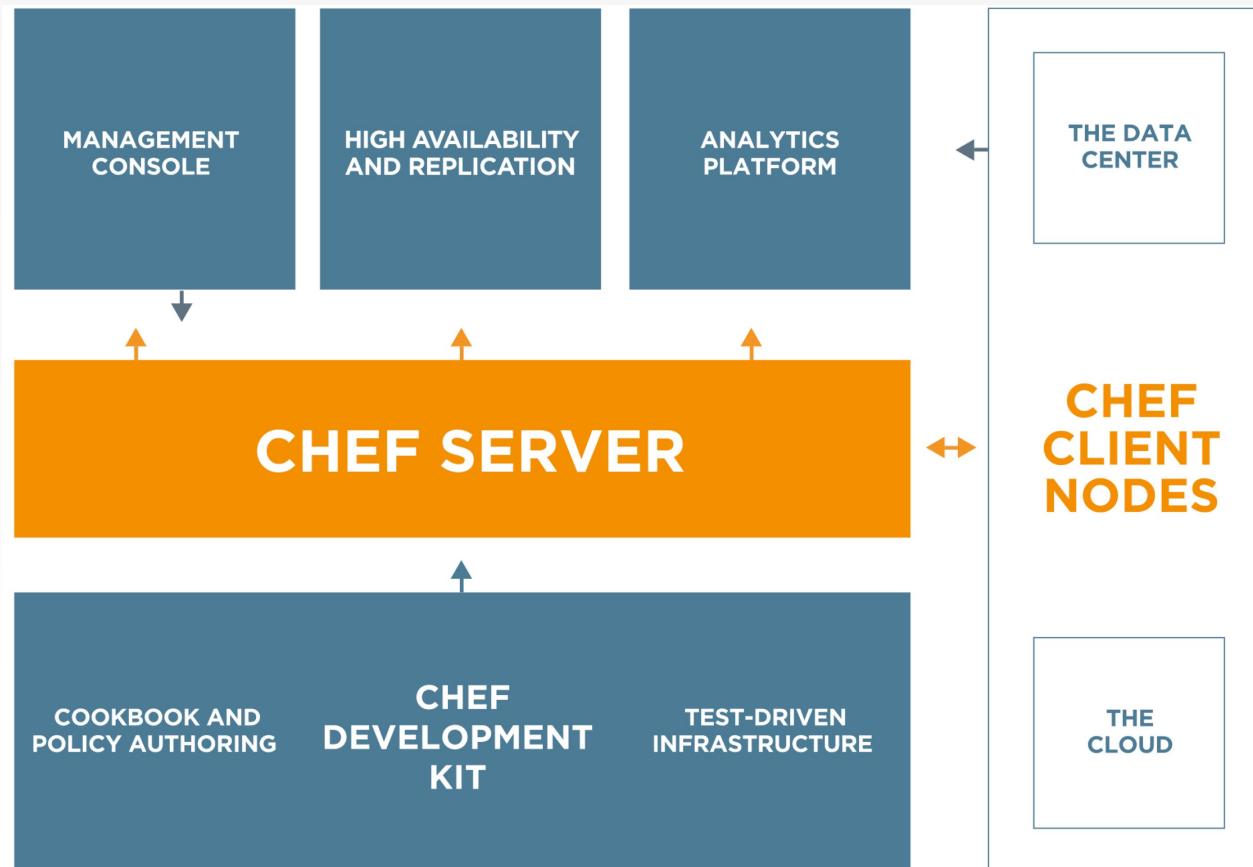
# Dimensions of Scale



# What is Chef

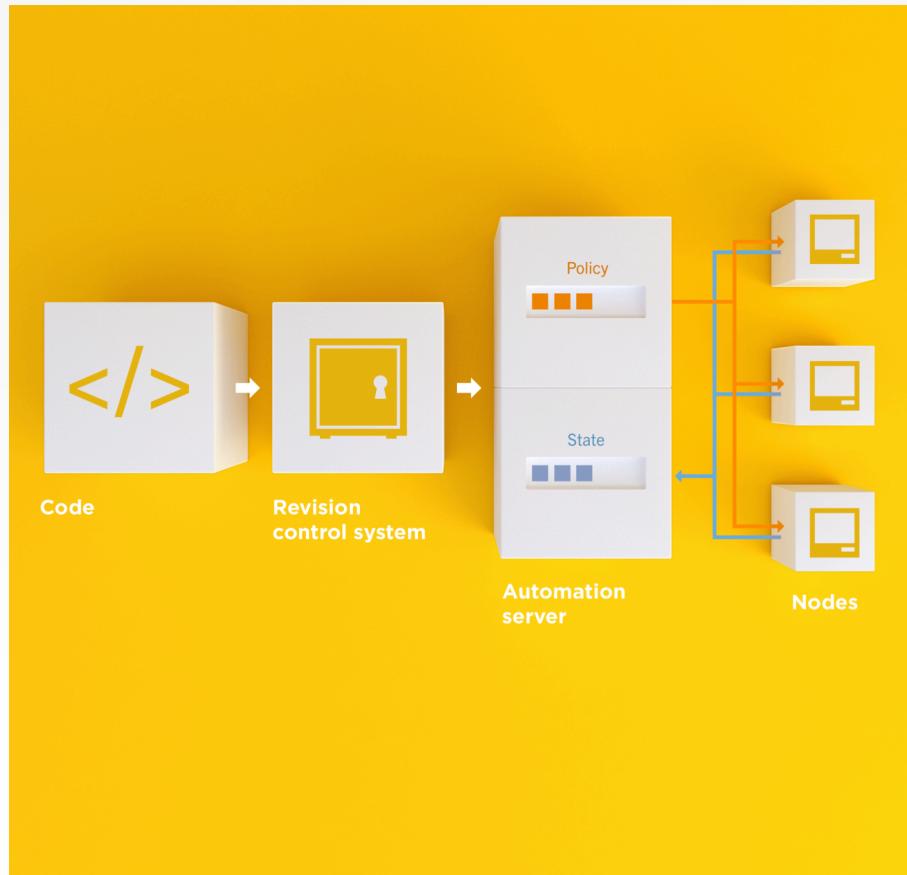
- Framework for managing complexity in your infrastructure through policy-driven automation code
- A community of professionals
- A company

# Chef

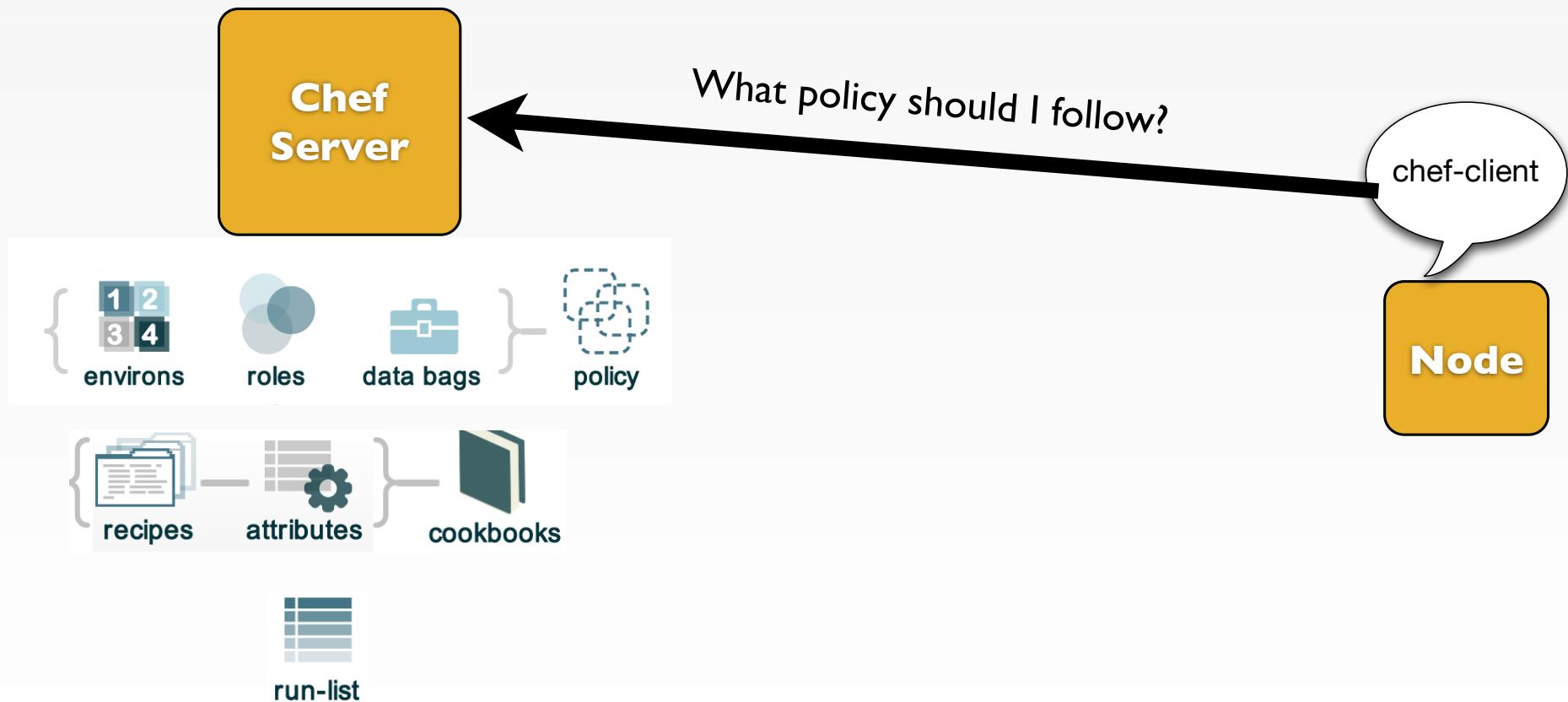


<https://www.chef.io/chef/>

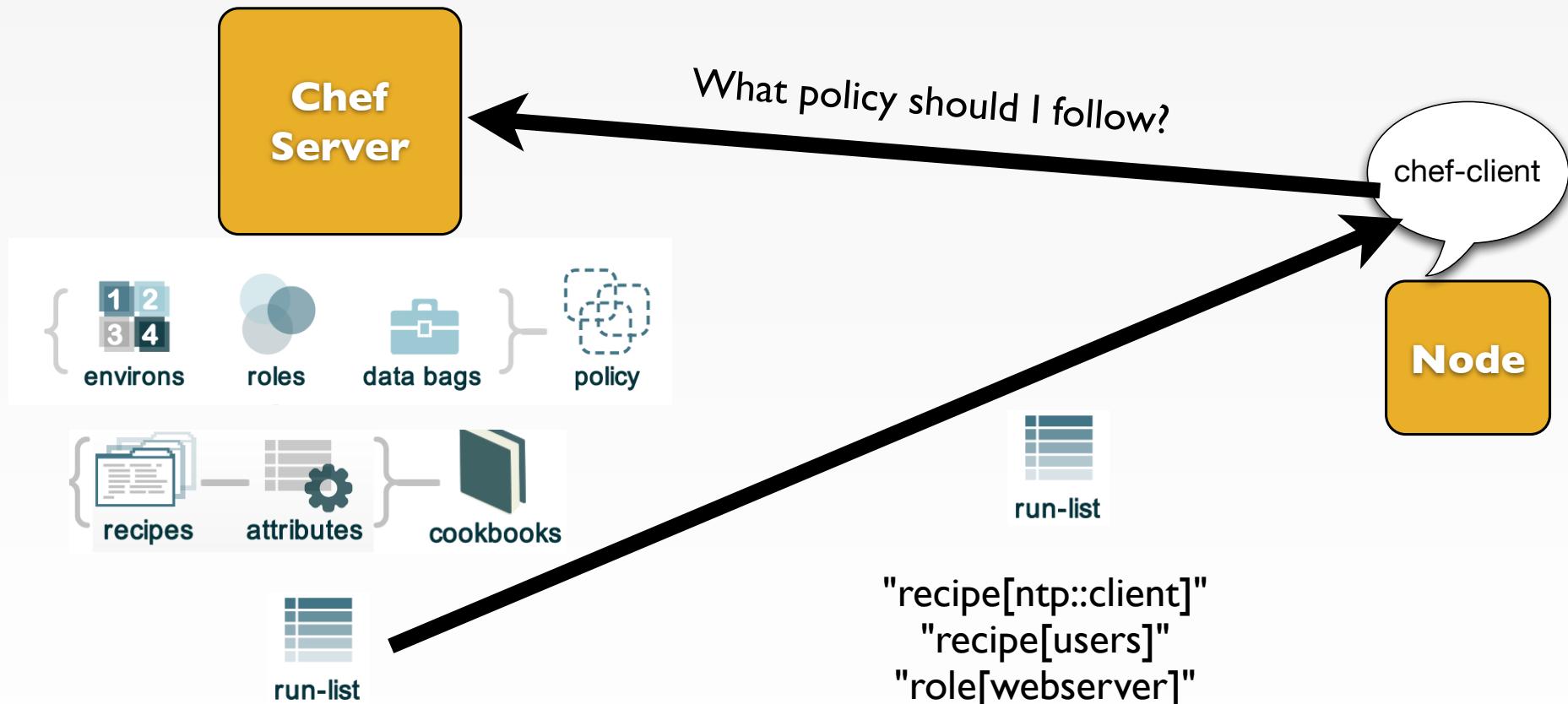
# Chef Server – Current & Desired State



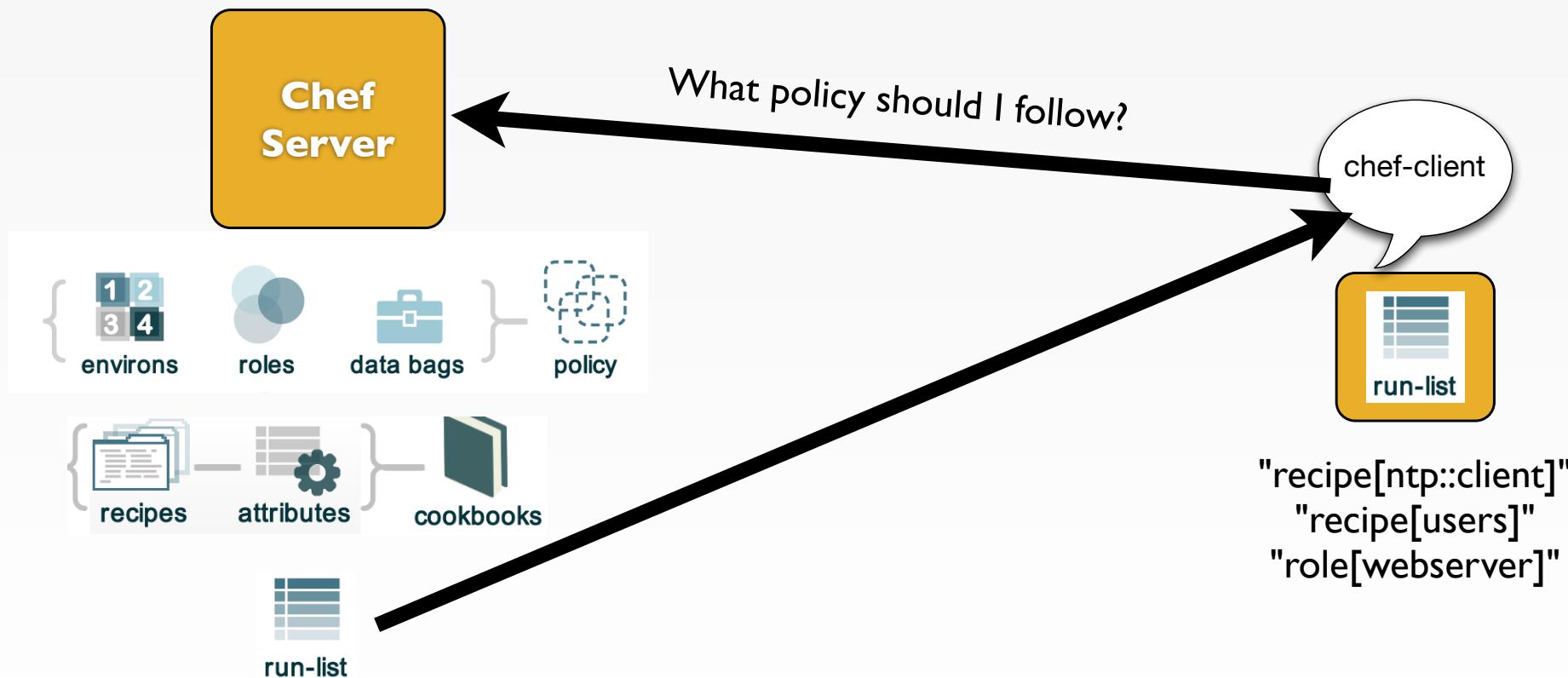
# Desired Configuration



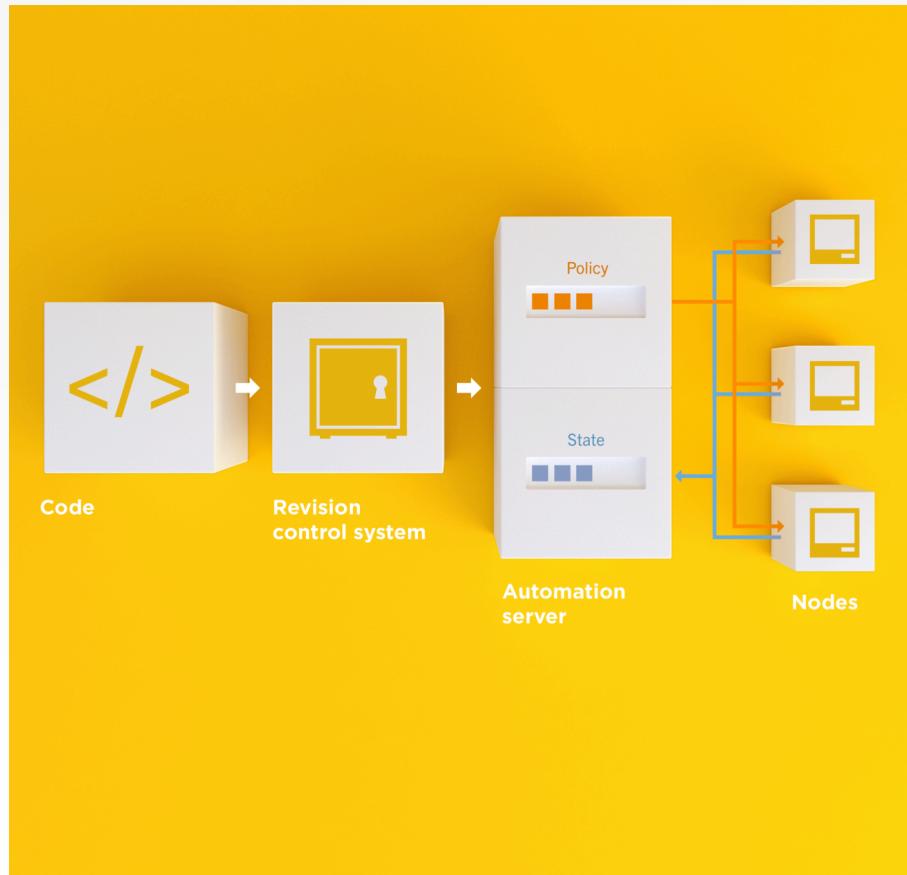
# Desired Configuration



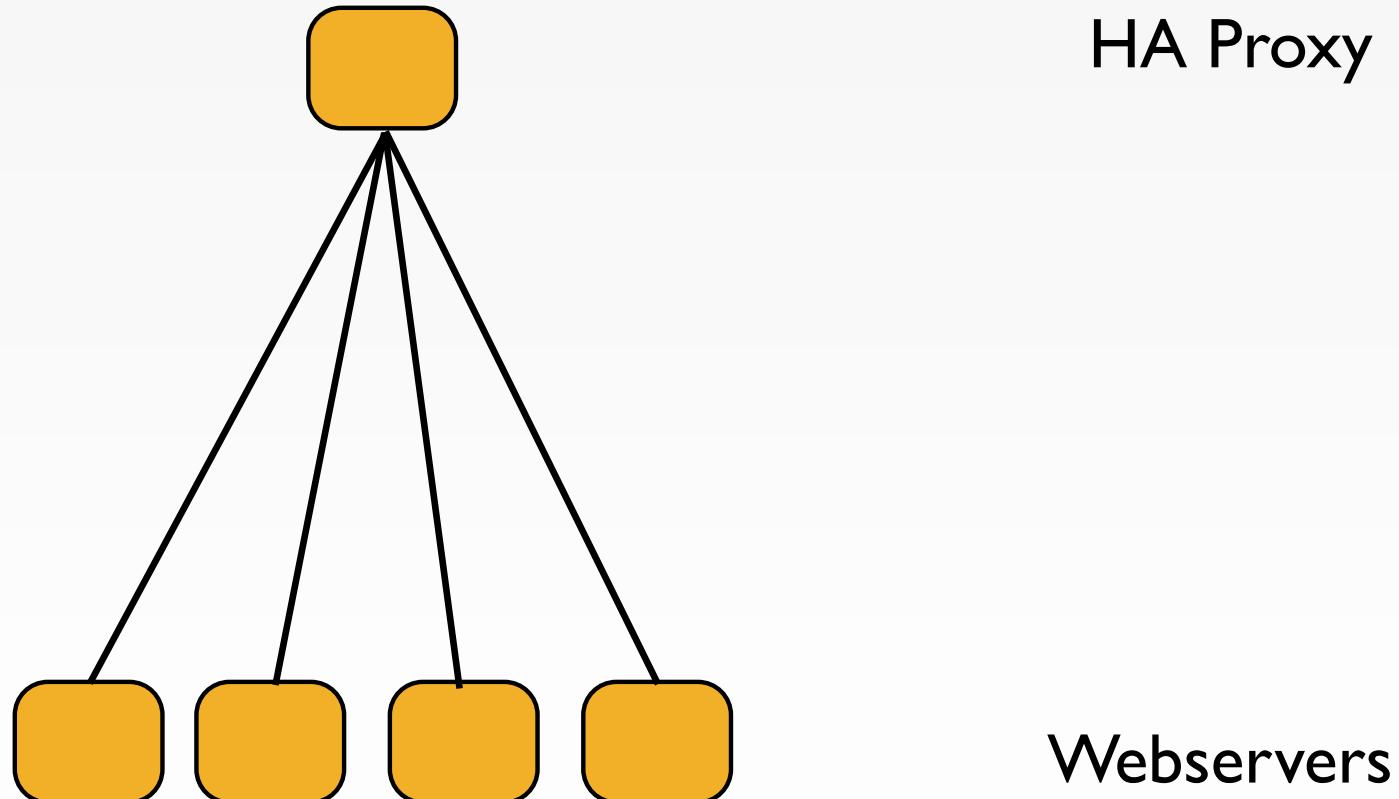
# Desired Configuration



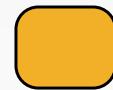
# Chef Server – Current & Desired State



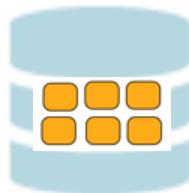
# HA Proxy Configuration



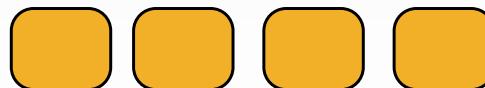
# HA Proxy Configuration



HA Proxy

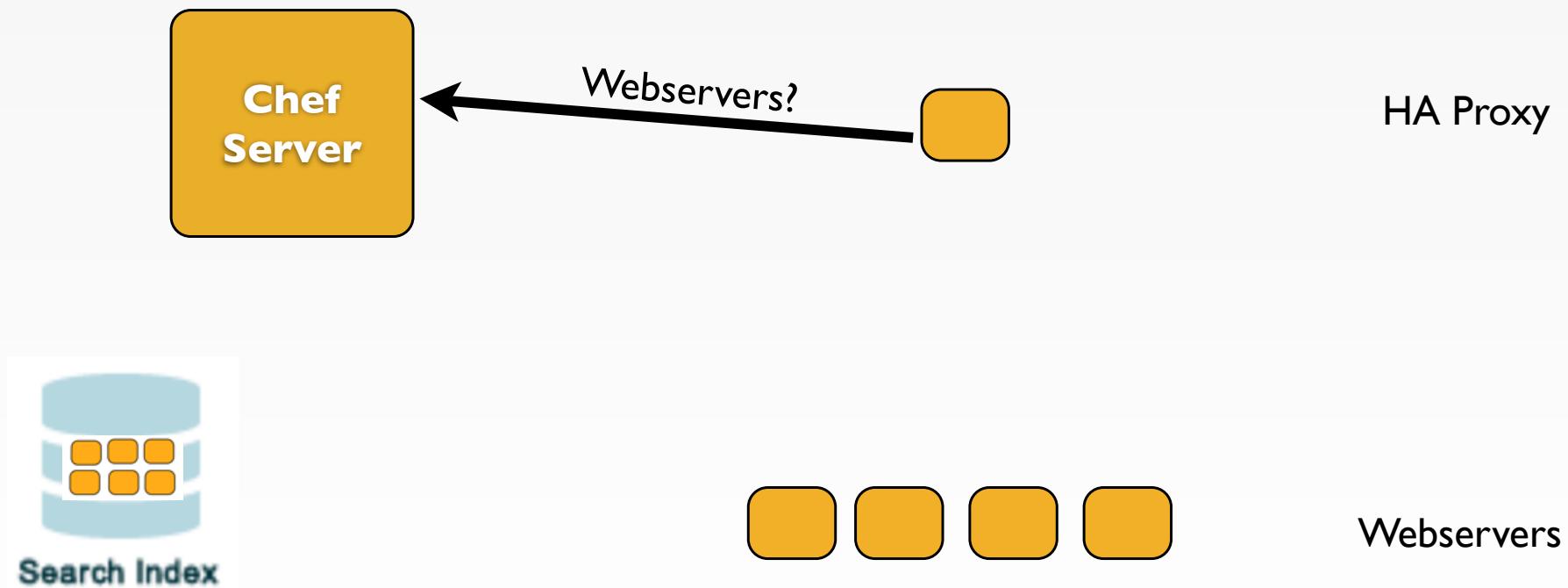


Search Index

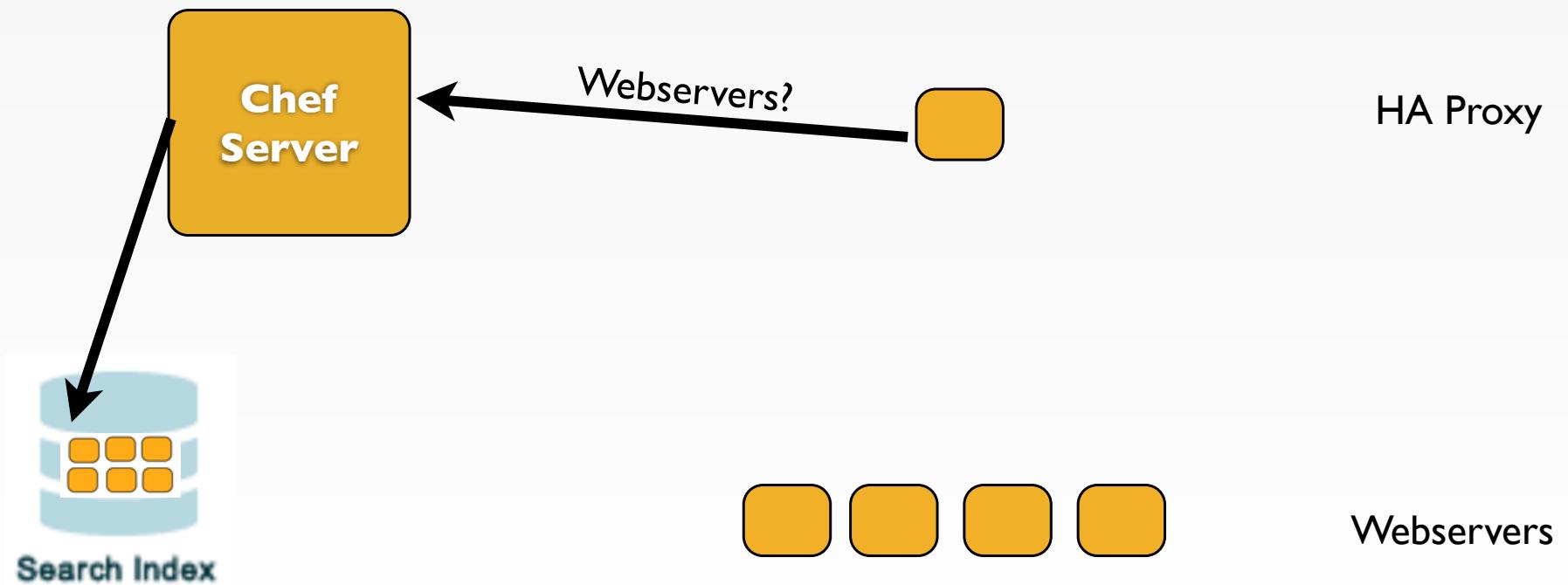


Webservers

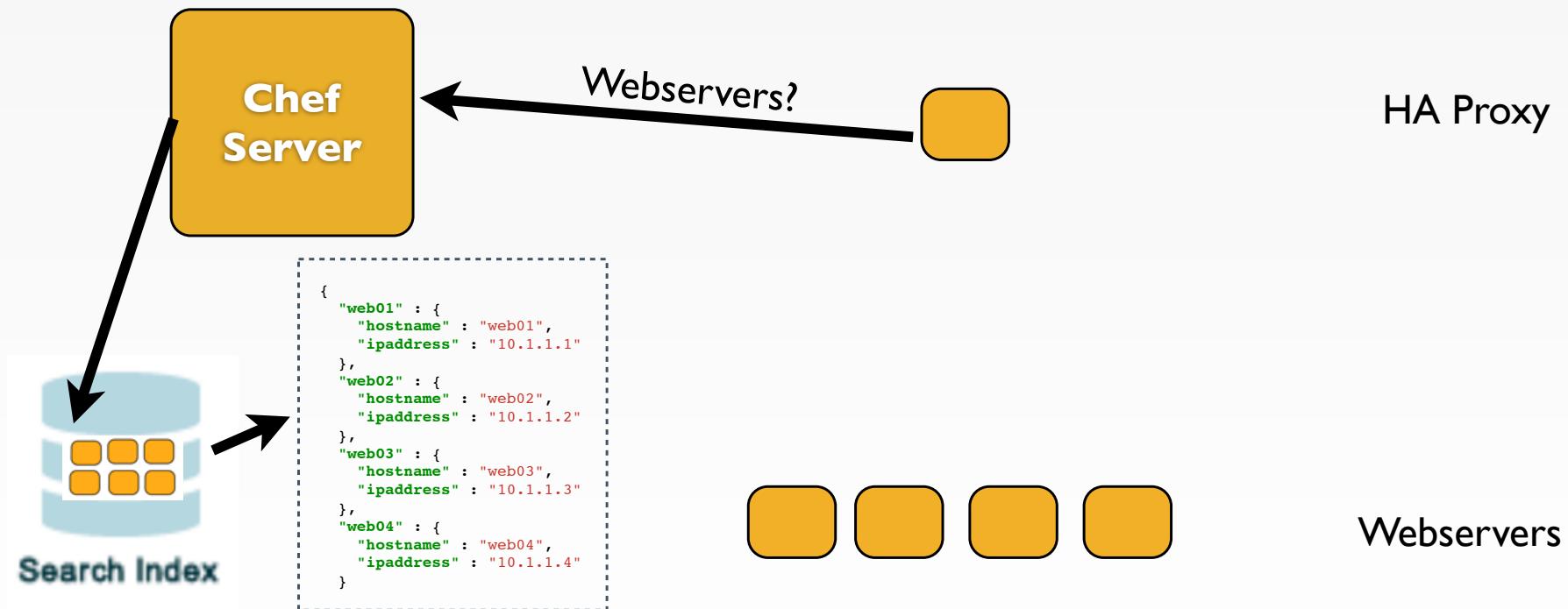
# HA Proxy Configuration



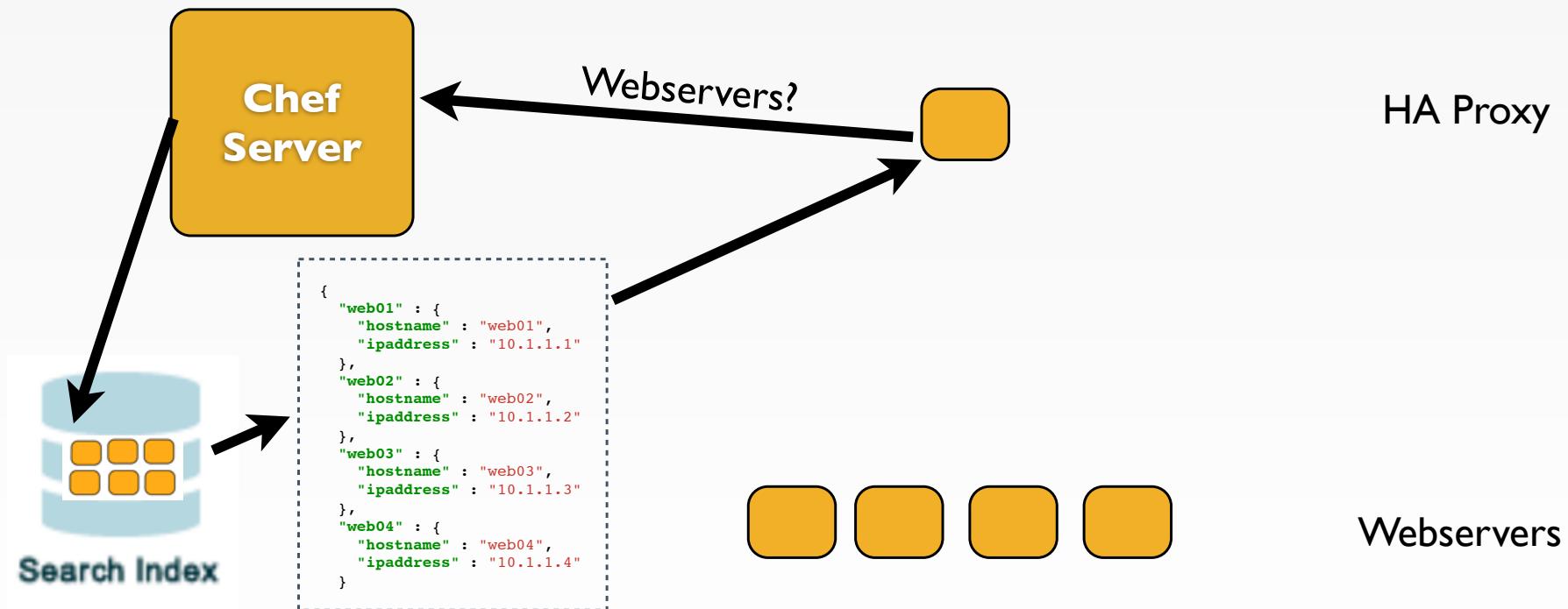
# HA Proxy Configuration



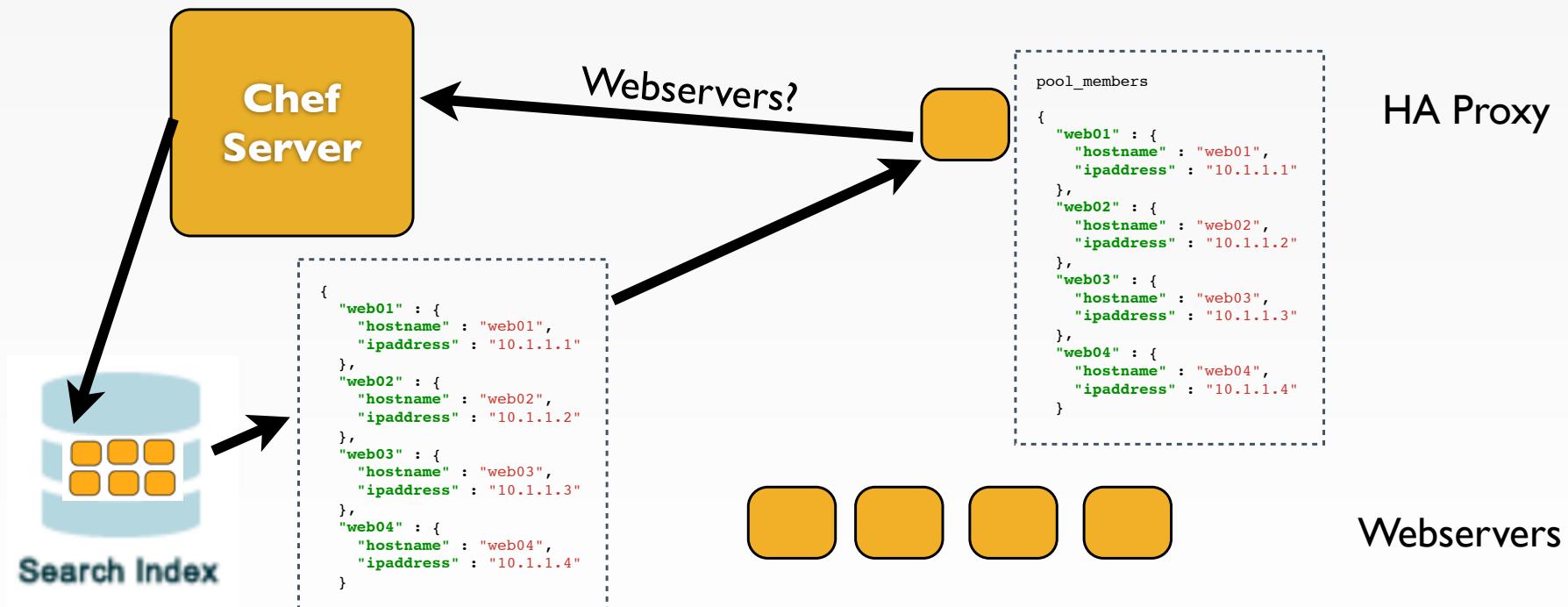
# HA Proxy Configuration



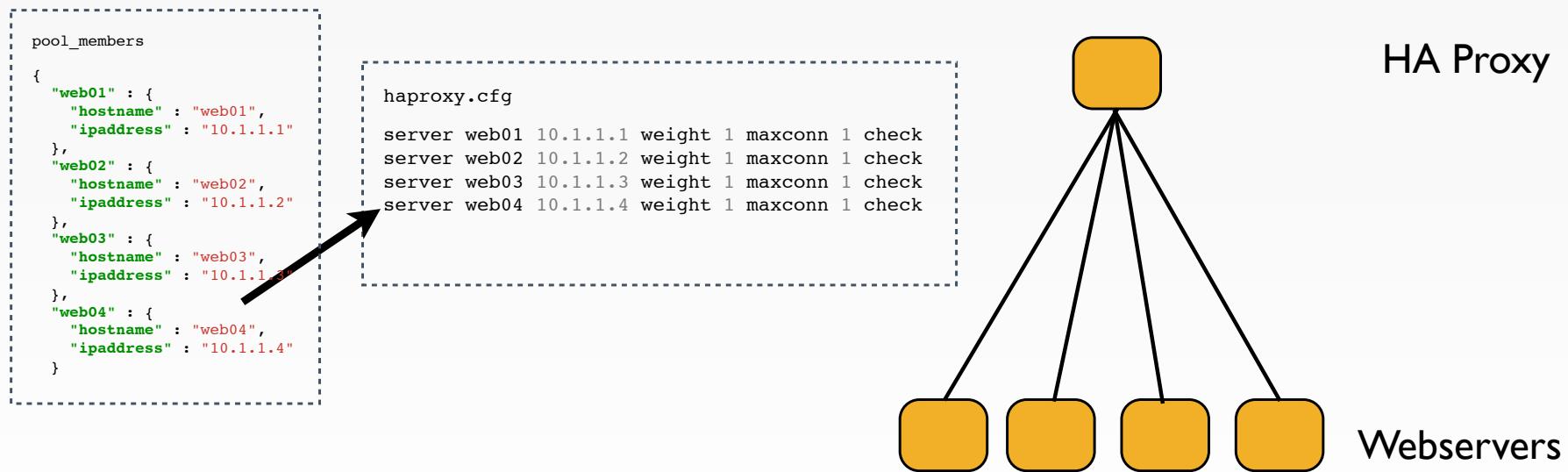
# HA Proxy Configuration



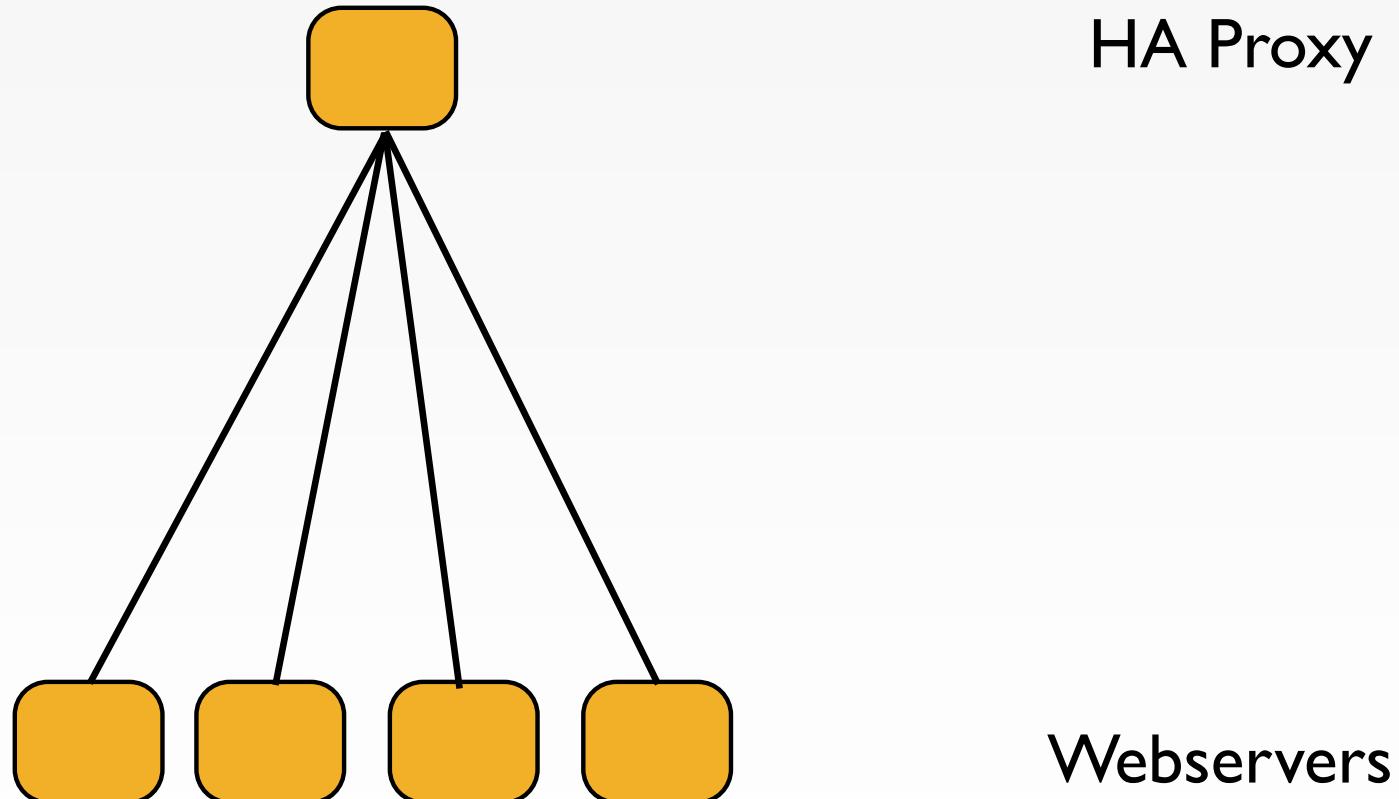
# HA Proxy Configuration



# HA Proxy Configuration



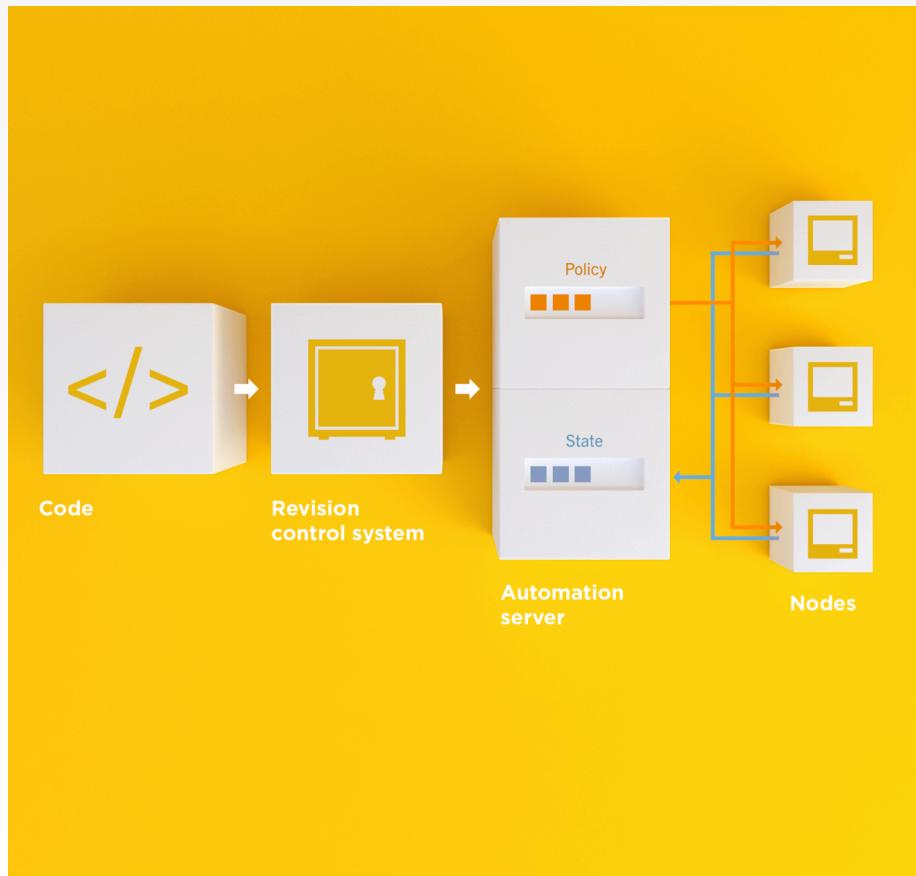
# HA Proxy Configuration



# Automation Platform

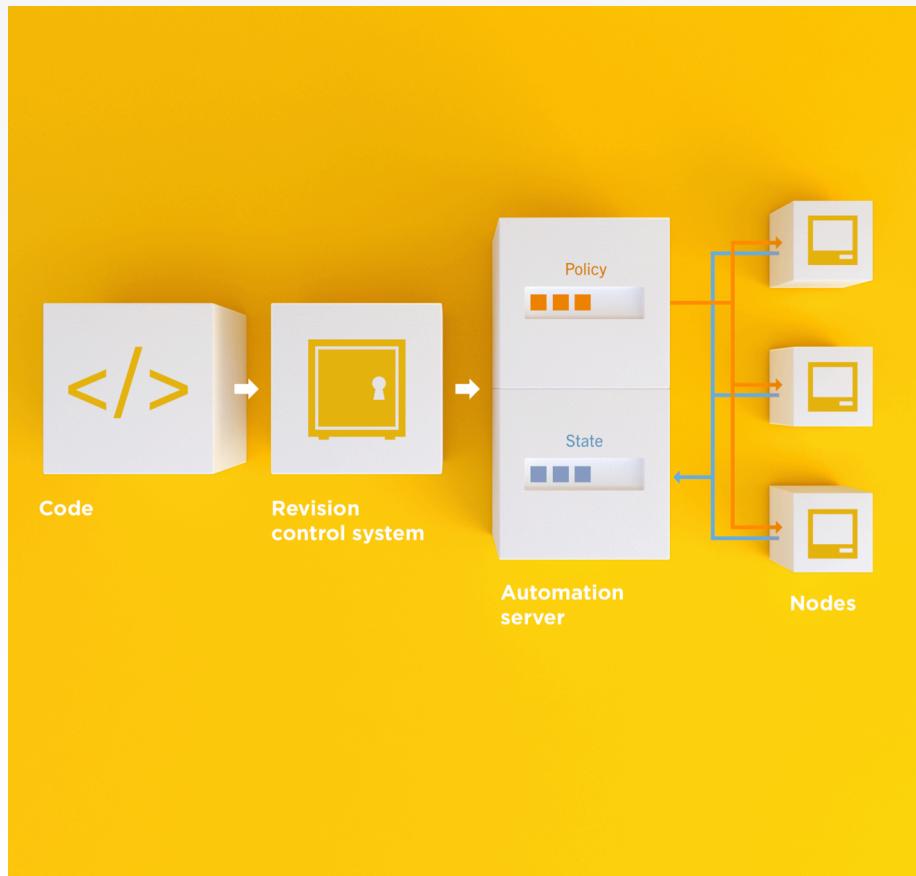
- Creates a dependable view of your entire network's state.
- Can handle complex dependencies among the nodes of your network.
- Is fault tolerant.
- Is secure.
- Can handle multiple platforms
- Can manage cloud resources
- Provides a foundation for innovation

# Infrastructure as Code



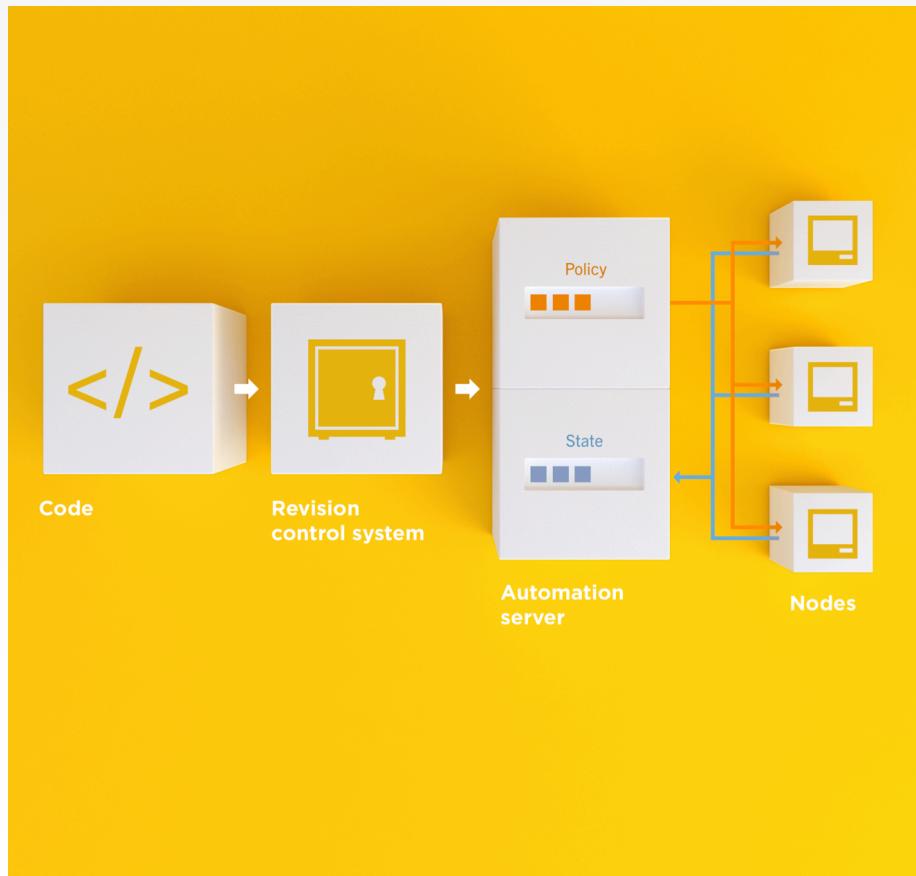
- Programmatically provision and configure components

# Infrastructure as Code



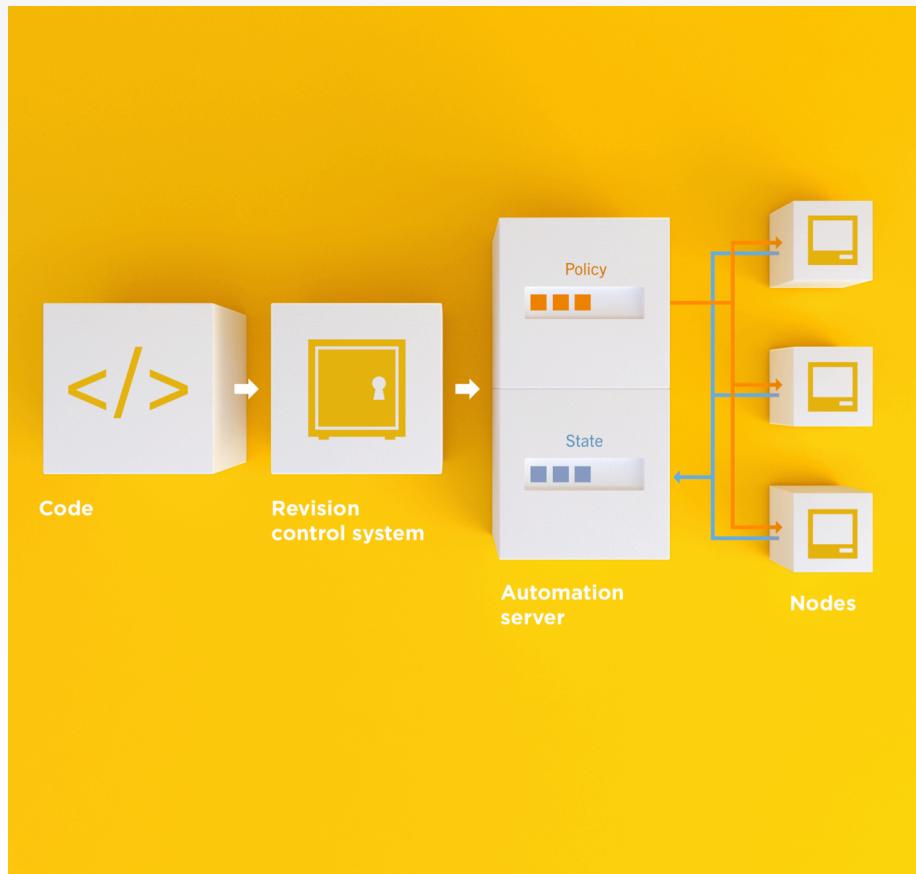
- Treat like any other code base

# Infrastructure as Code



- Reconstruct business from code repository, data backup, and compute resources

# Infrastructure as Code



- Programmatically provision and configure components
- Treat like any other code base
- Reconstruct business from code repository, data backup, and compute resources

# Policy-based

- You capture the policy for your infrastructure in code
- Chef ensures each node in your infrastructure complies with the policy

# Policy-based

- Chef provides a domain-specific language (DSL) that allows you to specify policy for your infrastructure
- Policy describes the desired state
- Policies can be statically or dynamically defined



# Resources

Fundamental building blocks

# Resources

- Piece of the system and its desired state

# Resources - Package

Package that should be installed

```
package "mysql-server" do
  action :install
end
```

# Resources - Service

Service that should be running and restarted on reboot

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

# Resources - Service

File that should be generated

```
file "/etc/motd" do
  content "Property of Chef Software"
end
```

# Resources - Cron

Cron job that should be configured

```
cron "restart webserver" do
  hour '2'
  minute '0'
  command 'service httpd restart'
end
```

# Resources - User

User that should be managed

```
user "nginx" do
  comment "Nginx user <nginx@example.com>"
  uid 500
  gid 500
  supports :manage_home => true
end
```

# Resources - DSC

DSC resource that  
should be run

```
dsc_script 'emacs' do
  code <<-EOH
    Environment 'texteditor'
  {
    Name = 'EDITOR'
    Value = 'c:\\emacs\\bin\\emacs.exe'
  }
EOH
end
```

# Resources – Registry Key

Registry key that should be created

```
registry_key "HKEY_LOCAL_MACHINE\  
  \SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System"  
do  
  values [ {  
    :name => "EnableLUA",  
    :type => :dword,  
    :data => 0  
  } ]  
  action :create  
end
```

# Resources

- Piece of the system and its desired state
- <http://docs.chef.io/chef/resources.html>

# Lab 1 – Install a text editor

- **Problem:** Our workstation does not have \$EDITOR installed
- **Success Criteria:** You can edit files with \$EDITOR
- \$EDITOR is your favorite command line text editor: vim, emacs, or nano

# Your Lab

- Vagrant using the given .box file
- Initialize a Vagrantfile
- Create the instance
- SSH into the instance

# Initialize the instance

```
$ vagrant init local-chef-development
```

A `Vagrantfile` has been placed in this directory. You are now ready to `vagrant up` your first virtual environment! Please read the comments in the Vagrantfile as well as documentation on `vagrantup.com` for more information on using Vagrant.

# Create the instance

```
$ vagrant up
```

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'local-chef-development'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: tmp_default_1432618530058_1324
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
...
```

# Login to the instance

```
$ vagrant ssh
```

```
Last login: Sat May 23 01:30:30 2015 from 10.0.2.2
```

# Vagrant ssh configuration

```
$ vagrant ssh-config
```

```
Host default
  HostName 127.0.0.1
  User vagrant
  Port 2222
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile /Users/nathenharvey/.vagrant.d/boxes/
local-chef-development/0/virtualbox/vagrant_private_key
  IdentitiesOnly yes
  LogLevel FATAL
```

# SSH directly

```
$ ssh -p 2222 vagrant@localhost
```

```
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be  
established.  
RSA key fingerprint is 7c:cd:63:05:bc:91:d4:3c:7b:80:93:ac:22:9d:60:63.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known  
hosts.  
vagrant@localhost's password:  
Last login: Tue May 26 05:36:34 2015 from 10.0.2.2
```

# SSH directly

```
$ ssh -p 2222 chef@localhost
```

```
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be  
established.  
RSA key fingerprint is 7c:cd:63:05:bc:91:d4:3c:7b:80:93:ac:22:9d:60:63.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known  
hosts.  
vagrant@localhost's password:  
Last login: Tue May 26 05:36:34 2015 from 10.0.2.2
```

# Welcome to your workstation

- ChefDK version 0.6.0 is installed
  - chef --version
- Chef user has passwordless sudo access
  - sudo cat /etc/shadow

# Is \$EDITOR installed?

```
$ which vim
```

```
/usr/bin/which: no vim in (/opt/
chefdk/bin:/home/chef/.chefdk/gem/
ruby/2.1.0/bin:/opt/chefdk/embedded/
bin:/usr/local/bin:/bin:/usr/bin:/
usr/local/sbin:/usr/sbin:/sbin:/
home/chef/bin)
```

# chef-apply

- chef-apply is an executable program that allows you to work with resources
- Is included as part of the ChefDK
- A great way to explore resources
- NOT how you'll eventually use Chef in production

# What does chef-apply do?

```
$ chef-apply --help
```

Usage: chef-apply [RECIPE_FILE] [-e RECIPE_TEXT] [-s]	Use colored output,
--[no-]color	
defaults to enabled	
-e, --execute RECIPE_TEXT	Execute resources
supplied in a string	
-l, --log_level LEVEL	Set the log level
(debug, info, warn, error, fatal)	
-s, --stdin	Execute resources
read from STDIN	
-v, --version	Show chef version
-W, --why-run	Enable whyrun mode
-h, --help	Show this message



# Install vim

```
$ sudo chef-apply -e "package 'vim'"
```

```
Recipe: (chef-apply cookbook) ::(chef-apply recipe)
* package[vim] action install
  - install version 7.2.411-1.8.el6 of package vim-enhanced
```

# Install emacs

```
$ sudo chef-apply -e "package 'emacs'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply recipe)
* package[emacs] action install
  - install version 23.1-25.el6 of package emacs
```

# Install nano

```
$ sudo chef-apply -e "package 'nano'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply  
recipe)
```

```
* package[nano] action install  
- install version 2.0.9-7.el6 of package nano
```

# Resources

- Describe the desired state
- Do not need to tell Chef how to get there
- What happens if you re-run the chef-apply command?

# Install \$EDITOR again with chef-apply

```
$ sudo chef-apply -e "package 'vim'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply
recipe)
  * package[vim] action install (up to date)
```

# Test and Repair

Resources follow a test  
and repair model

```
package "vim"
```

# Test and Repair

Resources follow a **test** and repair model

package "vim"

**Test** Is vim installed?

# Test and Repair

Resources follow a **test** and repair model

package "vim"

Test Is vim installed?

Yes

# Test and Repair

Resources follow a **test** and repair model

package "vim"

Test Is vim installed?

Yes

Done

# Test and Repair

Resources follow a **test** and repair model

package "vim"

Test Is vim installed?

Yes

Done

No

# Test and Repair

Resources follow a **test** and repair model

package "vim"

Test Is vim installed?

Yes

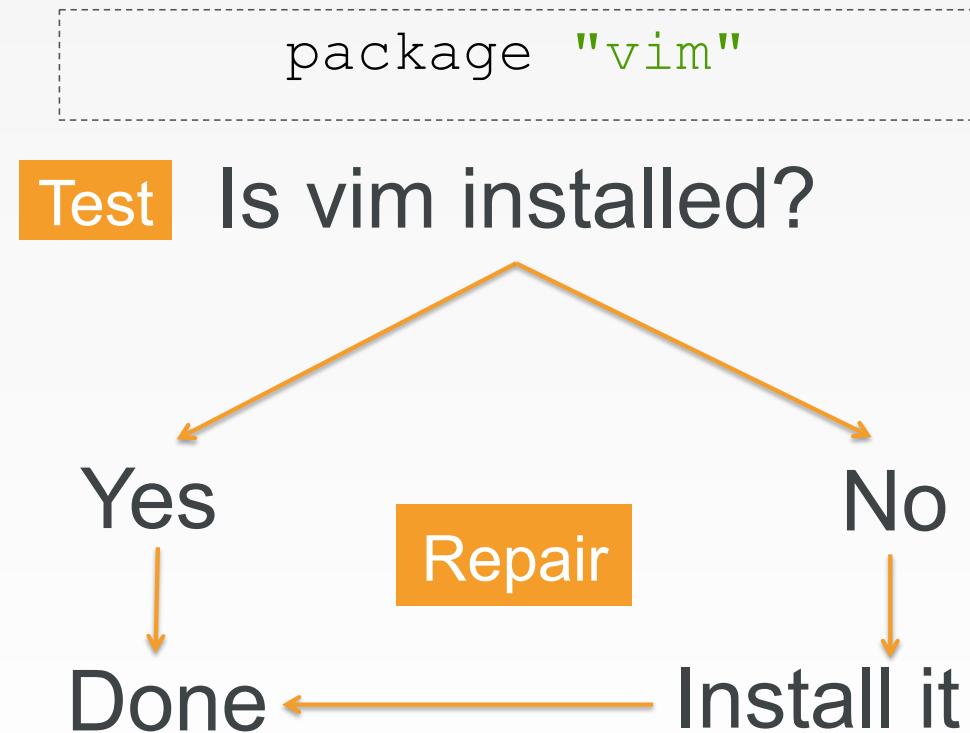
Done

No

Install it

# Test and Repair

Resources follow a **test** and **repair** model



## Resources – Test and Repair

- Resources follow a test and repair model
- Resource currently in the desired state? (test)
  - Yes – Do nothing
  - No – Bring the resource into the desired state (repair)

# Resources

- package
- template
- service
- directory
- user
- group
- dsc\_script
- registry\_key
- powershell\_script
- cron
- mount
- route
- ...and more!

## Lab 2 – Hello, world!

- **Problem:** Oops, we forgot to start with “hello, world”
- **Success Criteria:** A file with “Hello, world!” content is available in our home directory.

# Hello, world!



**OPEN IN EDITOR:** ~/hello.rb

```
file "hello.txt" do
  action :create
  content "Hello, world!"
  mode "0644"
  owner "chef"
  group "root"
end
```

**SAVE FILE!**

# Apply hello.rb

```
$ sudo chef-apply hello.rb
```

```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * file[hello.txt] action create
    - create new file hello.txt
    - update content in file hello.txt from none to 315f5b
      --- hello.txt          2014-12-02 14:00:22.967821184 +0000
      +++ /tmp/.hello.txt20141202-1036-af0vmi      2014-12-02 14:00:22.970821184 +0000
      @@ -1 +1,2 @@
      +Hello, world!
    - change mode from '' to '0644'
    - change owner from '' to 'chef'
    - change group from '' to 'root'
    - restore selinux security context
```

## Read hello.txt

```
$ cat hello.txt
```

```
Hello, world!
```

# Chef Resources

Have a type

```
file "hello.txt"
```

# Chef Resources

Have a name

- Have a type

```
file "hello.txt"
```

# Chef Resources

Include details between keywords **do** and **end**

- Have a name
- Have a type

```
file "hello.txt" do  
end
```

# Chef Resources

Describe the state of the thing using the keyword `action`

- Include details between keywords `do` and `end`
- Have a name
- Have a type

```
file "hello.txt" do
  action :create
end
```

# Chef Resources – In Plain English

The **TYPE** named  
**NAME** should be  
**ACTION**'d

```
file "hello.txt" do
  action :create
end
```

# Chef Resources – In Plain English

The **TYPE** named  
**NAME** should be  
**ACTION**'d

The **file** named  
“**hello.txt**” should be  
created

```
file "hello.txt" do
  action :create
end
```

# Chef Resources

- Include additional details about the state of the thing (attributes)
- Describe the state of the thing using the keyword `action`
- Include details between keywords `do` and `end`
- Have a name
- Have a type

```
file "hello.txt" do
  action :create
  content "Hello, world!"
  mode "0644"
  owner "chef"
  group "root"
end
```

# Chef Resources – In Plain English

The **TYPE** named  
**NAME** should be  
**ACTION'd** with  
**ATTRIBUTES**

```
file "hello.txt" do
  action :create
  content "Hello, world!"
  mode "0644"
  owner "chef"
  group "root"
end
```

# Chef Resources – In Plain English

The **file** named “**hello.txt**” should be created with content of “**Hello, world!**”, permissions of **0644**, owned by the **chef** user and **chef** group

```
file "hello.txt" do
  action :create
  content "Hello, world!"
  mode "0644"
  owner "chef"
  group "root"
end
```

# Hello, world!



**OPEN IN EDITOR:** ~/hello.rb

```
file "hello.txt" do
  content "Hello, world!"
  action :create
  mode "0644"
  owner "chef"
  group "root"
end
```

**SAVE FILE!**

## Re-apply hello.rb

```
$ sudo chef-apply hello.rb
```

```
Recipe: (chef-apply cookbook) :: (chef-apply  
recipe)
```

```
* file[hello.txt] action create (up to date)
```

## Resources – Test and Repair

- Resources follow a test and repair model
- Resource currently in the desired state? (test)
  - Yes – Do nothing
  - No – Bring the resource into the desired state (repair)

## What if...?

- Change the content of the file using your favorite text editor?
- Change the ownership of the file?
- Delete the file?

# Resources

- package
- template
- service
- directory
- user
- group
- dsc\_script
- registry\_key
- powershell\_script
- cron
- mount
- route

# Resources

- What states can a file be in?
- What state will a file be in if you don't declare an action?
- What state will a package be in if you don't declare an action?
- Do you have to indent the attributes of a resource?
- What Chef tool allows us to easily explore resources?

## Lab 3 – Manage a file

The file named /etc/motd should have the contents “Property of COMPANY NAME”, permissions of “0644”, and owned by the group and user named root

## Lab 3 – Manage a file

The **file** named `/etc/motd` should have the contents “**Property of COMPANY NAME**”, permissions of “**0644**”, and owned by the group and user named **root**

# Resources

- What questions can I answer for you?



# Describing Policies

Recipes and Cookbooks

v0.1.3



## Resources > Recipes > Cookbooks

- A resource is a piece of the system and it's desired state
- A recipe is a collection of resources
- A cookbook is a “package” of policy information

# Recipe - a collection of resources

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

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode "0644"
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => :true
  action [:enable, :start]
end
```

# Recipes – Order Matters

- Resources are applied in order

1st



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

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode "0644"
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => :true
  action [:enable, :start]
end
```

# Recipes – Order Matters

- Resources are applied in order

1st

2nd

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

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode "0644"
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => :true
  action [:enable, :start]
end
```

# Recipes – Order Matters

- Resources are applied in order

1st

2nd

3rd

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

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode "0644"
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => :true
  action [:enable, :start]
end
```

# Cookbook

- A “package” for Chef policies
- Typically map 1:1 to a piece of software or functionality

# Cookbooks – Packaged Policies

- Distribution unit
- Versioned
- Re-usable

# Abstracting Data from Policy

- Policy – The desired state of the system
- Data – The details that might change

# Abstracting Data from Policy

- Policy – Tomcat should be installed
- Data – Version 6

# Abstracting Data from Policy

- Policy – A file should exist
- Data – The content of that file

## Lab 4 – Manage Data & Policy Separately

- **Problem:** Policy for the state and content of /etc/motd are currently intermingled.
- **Success Criteria:** State and content of /etc/motd are managed separately.

# Message of the day

State – policy that describes the resource

```
file "/etc/motd" do
  content "Property of COMPANY NAME"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

# Message of the day

- Content – data that may change independent of policy changes

```
file "/etc/motd" do
  content "Property of COMPANY NAME"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

# Lab 4 - Manage Data & Policy Separately

- Create a chef-repo
- Create a cookbook

# chef-repo

- Chef cookbooks and other policy files should be stored in a version control system
- Create a directory named `chef-repo`
- Manage that directory as a git repository

# chef

- chef is an executable command line tool for
  - generating cookbooks, recipes, and other things that make up your Chef code
  - ensuring RubyGems are downloaded properly for your development environment
  - verifying that all the components are installed and configured correctly
- Included with ChefDK

# What can chef generate?

```
$ chef generate --help
```

```
Usage: chef generate GENERATOR [options]
```

```
Available generators:
```

app	Generate an application repo
cookbook	Generate a single cookbook
recipe	Generate a new recipe
attribute	Generate an attributes file
template	Generate a file template
file	Generate a cookbook file
lwrp	Generate a lightweight resource/provider
repo	Generate a Chef policy repository

# How do we generate a repo?

```
$ chef generate repo --help
```

```
Usage: chef generate repo NAME [options]
  -C, --copyright COPYRIGHT          Name of the copyright holder - defaults to 'The Authors'
  -m, --email EMAIL                 Email address of the author - defaults to 'you@example.com'
  -I, --license LICENSE            all_rights, apache2, mit, gplv2, gplv3 - defaults to all_rights
  -p, --policy-only                Create a repository for policy only, not cookbooks
  -g GENERATOR_COOKBOOK_PATH,
        --generator-cookbook       Use GENERATOR_COOKBOOK_PATH for the code_generator cookbook
```

# Go home!

```
$ cd ~
```

# Create a chef-repo

```
$ chef generate repo chef-repo
```

```
Compiling Cookbooks...
Recipe: code_generator::repo
* directory[/home/chef/chef-repo] action create
  - create new directory /home/chef/chef-repo
  - restore selinux security context
* template[/home/chef/chef-repo/LICENSE] action create
  - create new file /home/chef/chef-repo/LICENSE
  - update content in file /home/chef/chef-repo/LICENSE from none to dbclaf
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/home/chef/chef-repo/README.md] action create
  - create new file /home/chef/chef-repo/README.md
  - update content in file /home/chef/chef-repo/README.md from none to 767ead
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/home/chef/chef-repo/Rakefile] action create
```



# Commit this chef-repo to git

```
$ cd chef-repo
```

# Lab 4 – Manage Data & Policy Separately

- ✓ Create a chef-repo
- Create a cookbook

# Create an motd cookbook

```
$ chef generate cookbook --help
```

```
Usage: chef generate cookbook NAME [options]
      -C, --copyright COPYRIGHT           Name of the copyright holder - defaults to 'The Authors'
      -m, --email EMAIL                  Email address of the author - defaults to 'you@example.com'
      -I, --license LICENSE             all_rights, apache2, mit, gplv2, gplv3 - defaults to all_rights
      -g GENERATOR_COOKBOOK_PATH,       Use GENERATOR_COOKBOOK_PATH for the code_generator cookbook
      --generator-cookbook
```

# Create a motd cookbook

```
$ cd cookbooks
```

# Create a cookbook

```
$ chef generate cookbook motd
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
* directory[/home/chef/chef-repo/cookbooks/motd] action create
  - create new directory /home/chef/chef-repo/cookbooks/motd
* template[/home/chef/chef-repo/cookbooks/motd/metadata.rb] action create_if_missing
  - create new file /home/chef/chef-repo/cookbooks/motd/metadata.rb
  - update content in file /home/chef/chef-repo/cookbooks/motd/metadata.rb from none to 7852c2
    (diff output suppressed by config)
* template[/home/chef/chef-repo/cookbooks/motd/README.md] action create_if_missing
...
...
```

## Copy your motd.rb

```
$ cat ~/motd.rb >> motd/recipes/default.rb
```

# Update the recipe



**OPEN IN EDITOR:** `~/chef-repo/motd/recipes/default.rb`

```
#  
# Cookbook Name:: motd  
# Recipe:: default  
#  
# Copyright (c) 2014 The Authors, All Rights Reserved.  
file "/etc/motd" do  
  content "Property of COMPANY NAME"  
  action :create  
  mode "0644"  
  owner "root"  
  group "root"  
end
```

**SAVE FILE!**

# What resource should we use?

**Resources:** [About Resources](#) | [Common Functionality](#) — **Resources:** [apt\\_package](#) | [bash](#) | [batch](#) | [breakpoint](#) | [chef\\_gem](#) | [chef\\_handler](#) | [cookbook\\_file](#) | [cron](#) | [deploy](#) | [directory](#) | [dpkg\\_package](#) | [dsc\\_script](#) | [easy\\_install\\_package](#) | [env](#) | [erl\\_call](#) | [execute](#) | [file](#) | [gem\\_package](#) | [git](#) | [group](#) | [http\\_request](#) | [ifconfig](#) | [link](#) | [log](#) | [mdadm](#) | [mount](#) | [ohai](#) | [package](#) | [powershell\\_script](#) | [registry\\_key](#) | [remote\\_directory](#) | [remote\\_file](#) | [route](#) | [rpm\\_package](#) | [ruby\\_block](#) | [script](#) | [service](#) | [subversion](#) | [template](#) | [user](#) | [yum\\_package](#) | [windows\\_package](#) — **Single Page:** [Resources and Providers](#)

- `cookbook_file`
- `file`
- `remote_file`
- `template`

# **cookbook\_file**

A file stored in the cookbook contains the content of the file.

```
motd
├── Berksfile
├── README.md
├── chefignore
└── files
    └── default
        └── motd
├── metadata.rb
└── recipes
    └── default.rb
```

# file

The content is described inline in the recipe

```
file "/etc/motd" do
  content "Property of COMPANY NAME"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

## **remote\_file**

The file is stored in a remote location, such as on the web

```
file "/etc/motd" do
  url "http://some.where.com/motd"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

# template

A template file is stored as part of the cookbook

```
motd
├── Berksfile
├── README.md
├── chefignore
├── metadata.rb
├── recipes
│   └── default.rb
└── templates
    └── default
        └── motd.erb
```

# template

A template file is stored as part of the cookbook and rendered to create the file.

`motd/templates/default/motd.erb`

Property of <%= @company\_name %>

# Which resource should we use?

- `cookbook_file` – static file, within the cookbook
- `file` – content managed inline
- `remote_file` – static file, obtained from a URL
- `template` – dynamic content based on ERB template

# Template Resource

- An ERB template stored as part of our cookbook

# Update the recipe



**OPEN IN EDITOR:** `~/chef-repo/motd/recipes/default.rb`

```
#  
# Cookbook Name:: motd  
# Recipe:: default  
  
#  
# Copyright (c) 2014 The Authors, All Rights Reserved.  
template "/etc/motd" do  
  action :create  
  source "motd.erb"  
  mode "0644"  
  owner "root"  
  group "root"  
end
```

**SAVE FILE!**

# Create the ERB template

```
$ chef generate template --help
```

```
Usage: chef generate template [path/to/cookbook] NAME [options]
      -C, --copyright COPYRIGHT           Name of the copyright holder
      - defaults to 'The Authors'
      -m, --email EMAIL                  Email address of the author
      - defaults to 'you@example.com'
      -I, --license LICENSE             all_rights, apache2, mit,
gplv2, gplv3 - defaults to all_rights
      -s, --source SOURCE_FILE          Copy content from
SOURCE_FILE
      -g GENERATOR_COOKBOOK_PATH,      Use GENERATOR_COOKBOOK_PATH
for the code_generator cookbook
      --generator-cookbook
```

# Go to the motd cookbook directory

```
$ cd ~/chef-repo/cookbooks/motd
```

# Create the ERB template

```
$ chef generate template . motd -s /etc/motd
```

```
Compiling Cookbooks...
Recipe: code_generator::template
 * directory[././templates/default] action create
   - create new directory ././templates/default
 * file[././templates/default/motd.erb] action create
   - create new file ././templates/default/motd.erb
   - update content in file ././templates/default/
motd.erb from none to 315f5b
 (diff output suppressed by config)
```

# Check the template



**OPEN IN EDITOR:** `~/chef-repo/cookbooks/motd/templates/default/motd.erb`

Property of COMPANY NAME

**SAVE FILE!**

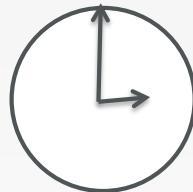
## chef-apply

- chef-apply does not understand cookbooks, only resources and recipes
- We cannot use chef-apply to apply the policy stored in our motd cookbook

# chef-client

- chef-client is an executable
  - performs all actions required to bring the node into the desired state
  - typically run on a regular basis
    - daemon
    - cron
    - Windows service
- Included with ChefDK

# chef-client applying policies

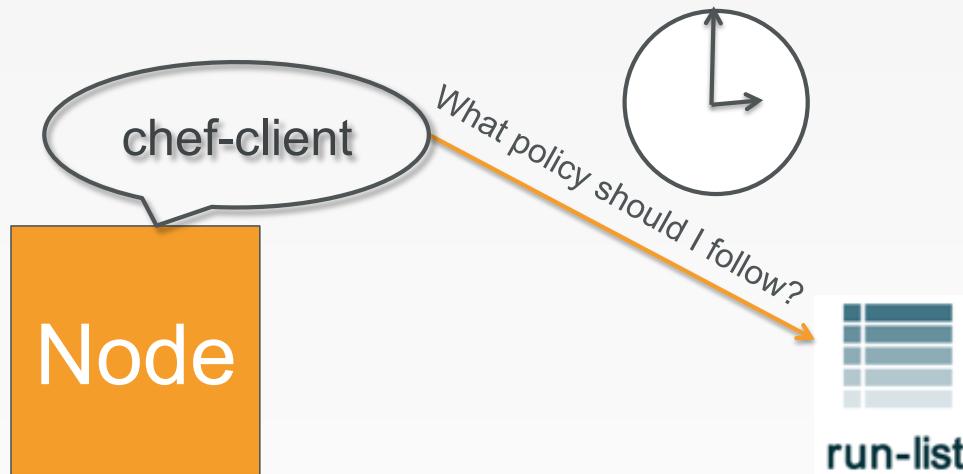


Node

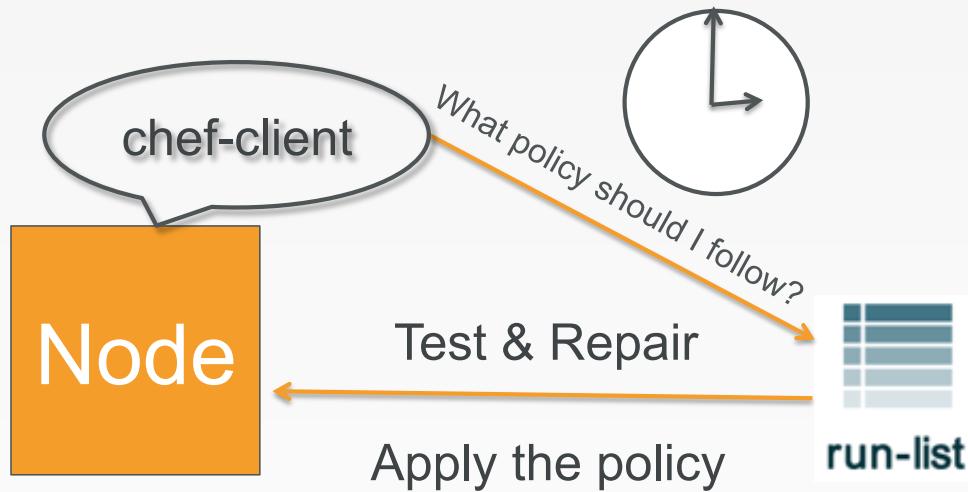
# chef-client applying policies



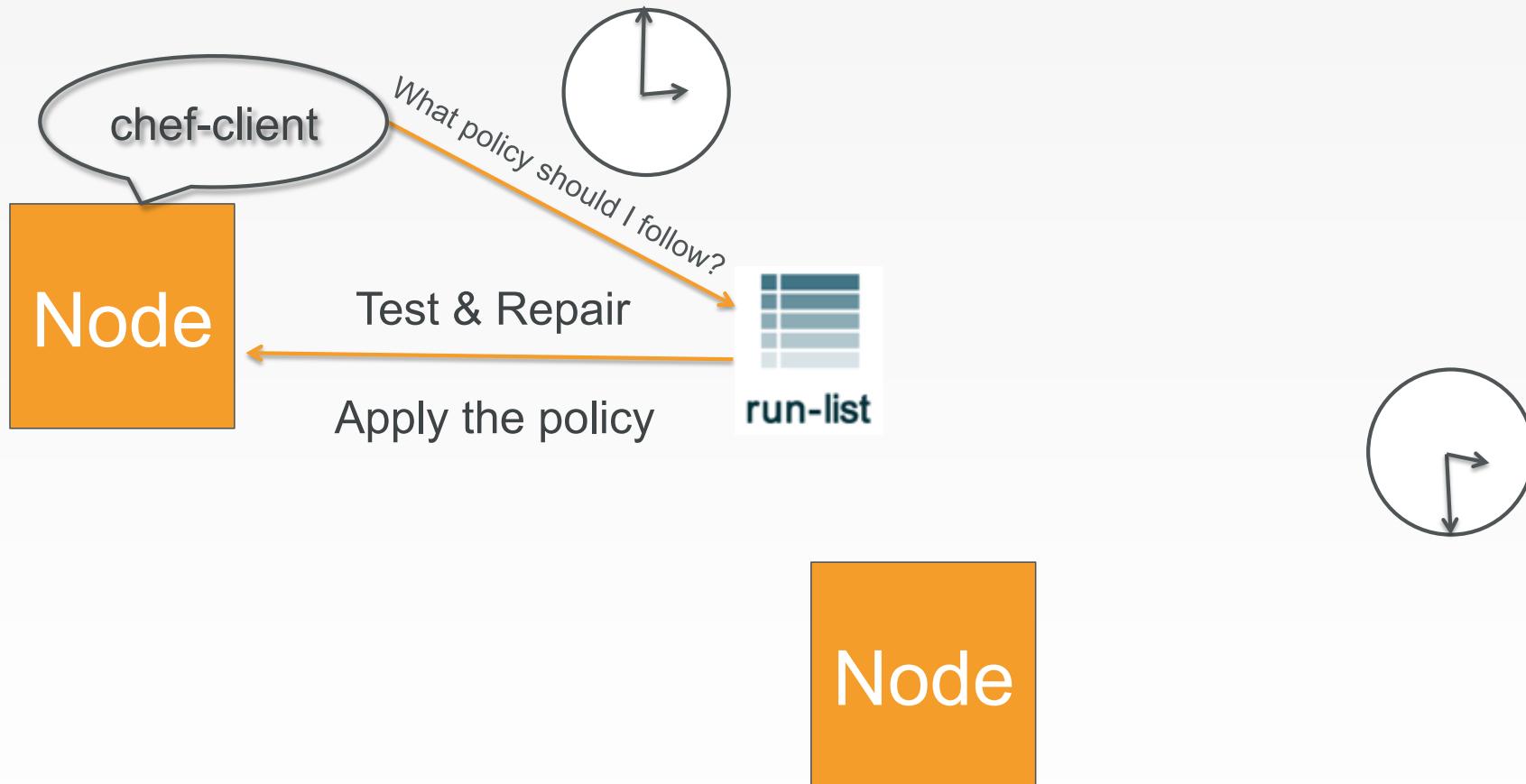
# chef-client applying policies



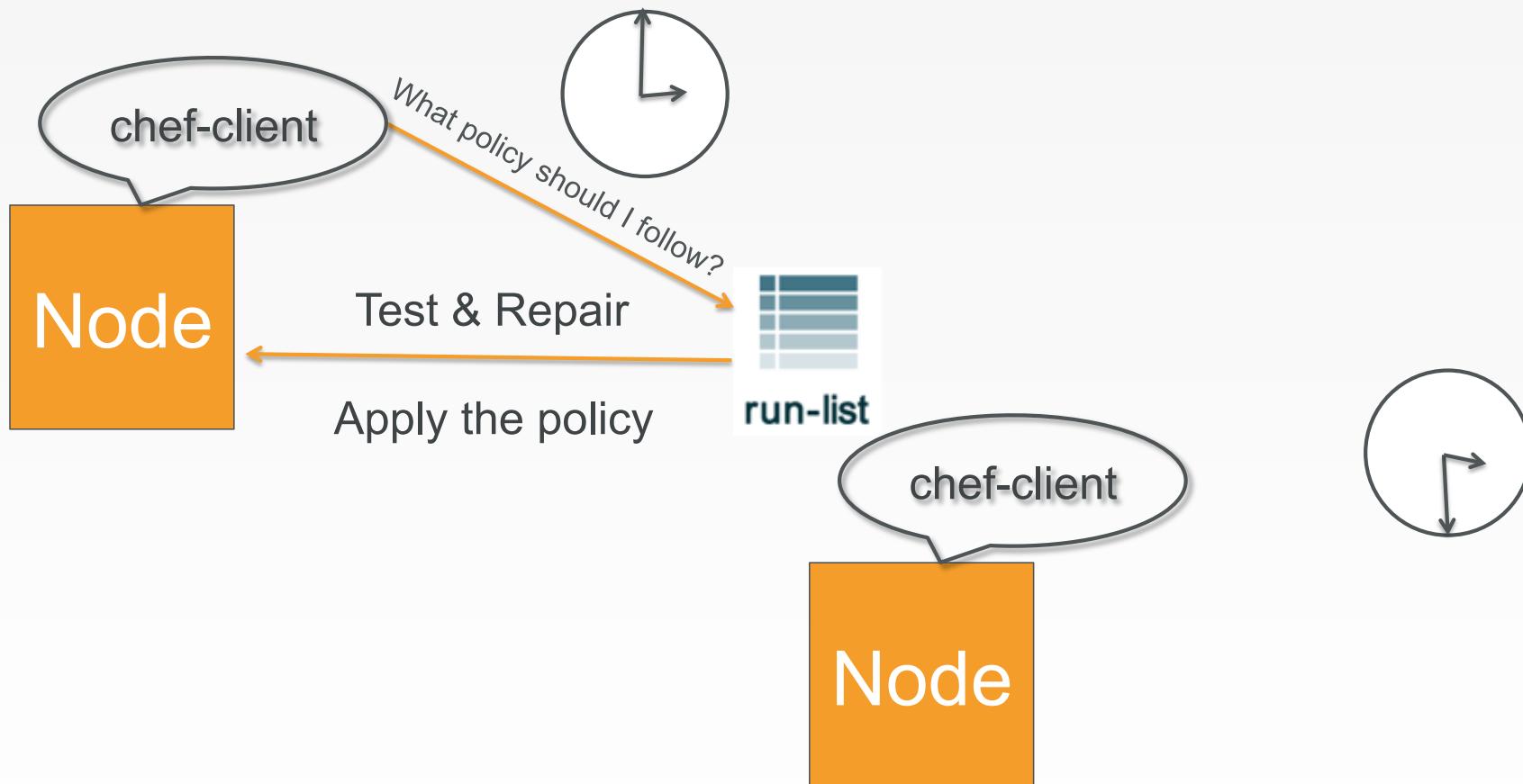
# chef-client applying policies



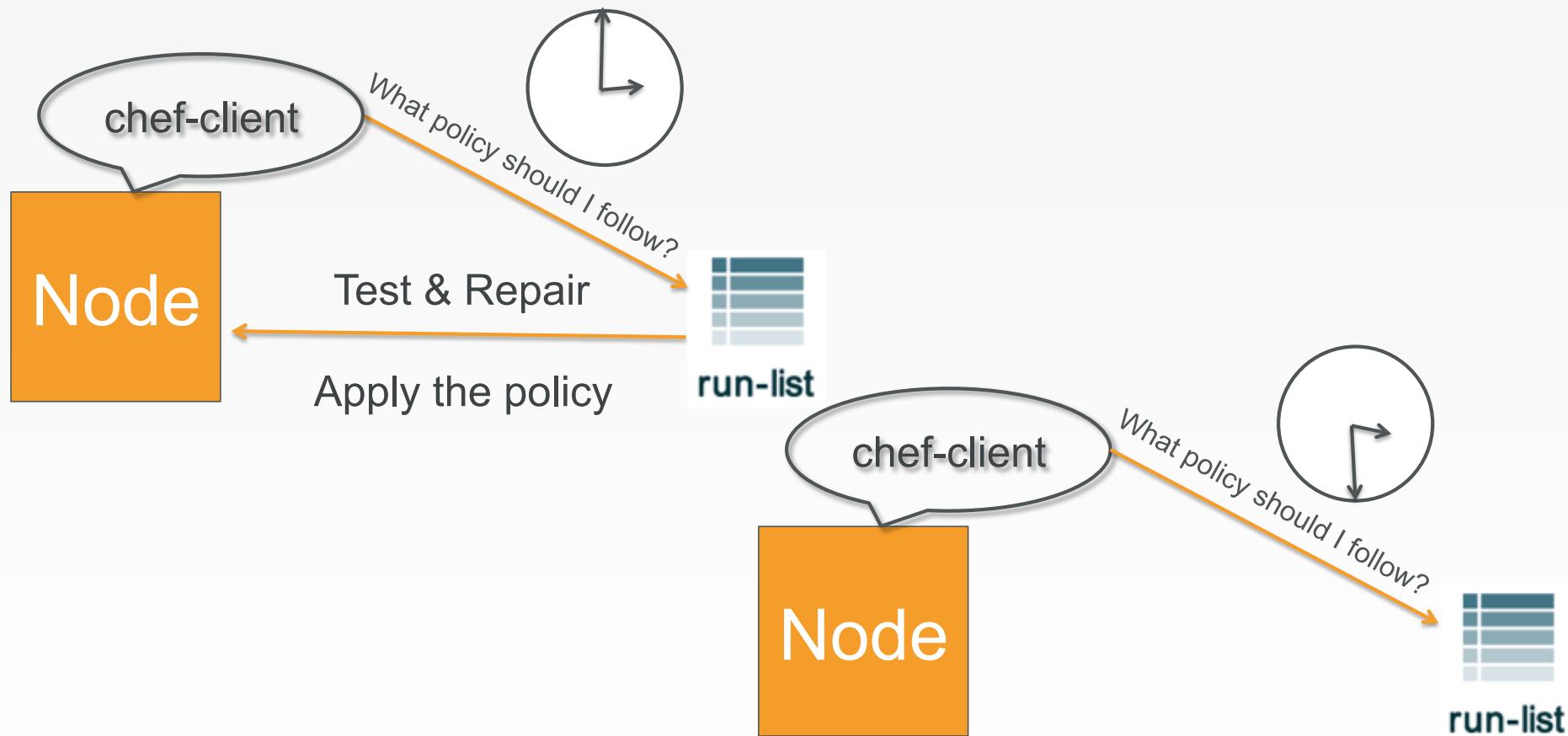
# chef-client applying policies repeatedly



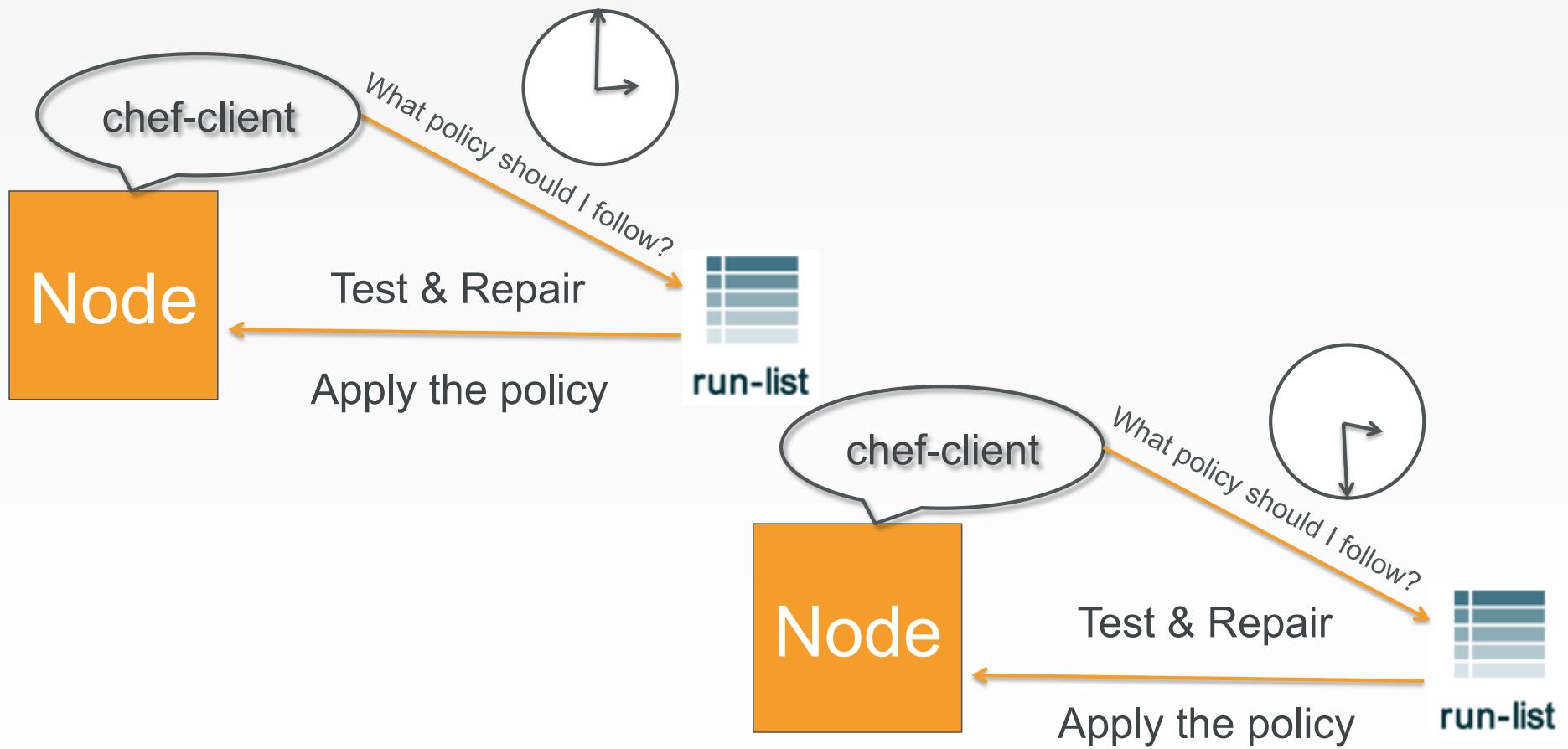
# chef-client applying policies repeatedly



# chef-client applying policies repeatedly



# chef-client applying policies repeatedly



## chef-client modes

- In conjunction with a Chef Server
- Local mode (no Chef Server)

# chef-client privileges

- Usually run with elevated privileges
  - root
  - sudo
  - Administrator
- Can run as a normal user

# Apply our recipe using chef-client

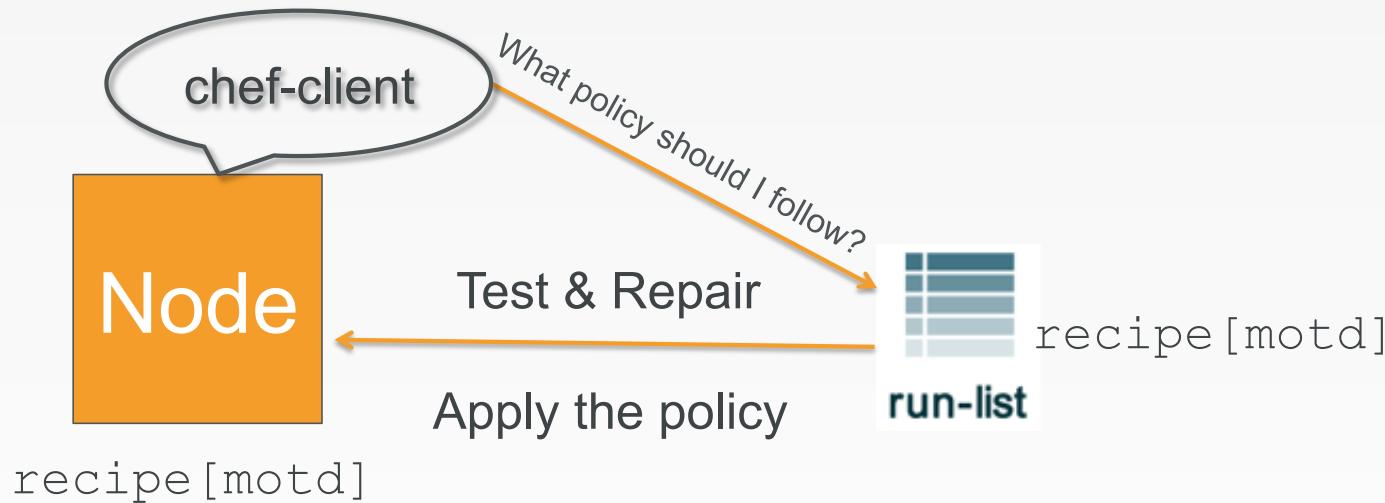
```
$ cd ~/chef-repo
```

# Apply our recipe using chef-client

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

```
[2014-12-02T15:13:21+00:00] WARN: No config file found or specified on command line, using command line options.  
Starting Chef Client, version 11.18.0.rc.1  
resolving cookbooks for run list: ["motd"]  
Synchronizing Cookbooks:  
  - motd  
Compiling Cookbooks...  
Converging 1 resources  
Recipe: motd::default  
* template[/etc/motd] action create  
  - update content in file /etc/motd from 4fe2f6 to e989a4  
    (no diff)  
  - restore selinux security context  
  
Running handlers:  
Running handlers complete  
Chef Client finished, 1/1 resources updated in 3.346092479 seconds
```

# chef-client applying policies



# Lab 4 – Manage Data & Policy Separately

- ✓ Create a chef-repo
- ✓ Create a cookbook

# Separating data from policy

- Storing the file's content directly in the recipe feels wrong
- We can manage that content separately using a different resource
  - cookbook\_file
  - remote\_file
  - template

# Template resource

- An ERB template that is used to generate files based on the variables and logic contained within the template.

## What if...?

- The contents of motd should be pulled from a file in an s3 bucket?
- The motd file should have variable content?

# Describing Policies

- Describe the relationship between resource, recipes, and cookbooks?
- What types of files might you find in a cookbook?
- Where is the version of a cookbook specified?

# Describing Policies

- What questions can I answer for you?



# A Sandbox for Testing

Test Kitchen

v0.1.3



# Our process

- Write policy
  - Apply policy
  - Verify policy
- 
- Not bad for the simple case, will quickly get untenable

# Faster Feedback

- Speed-up the feedback loops with automated testing.
- Have confidence in your changes before you run them in production

# The pedantries of testing

- Unit testing
- Integration testing
- Acceptance testing
- Functional testing
- Regression testing
- Smoke testing
- Load testing

# Chef Testing

- Did chef-client complete successfully?
- Did the recipe put the node in the desired state?
- Are the resources properly defined?
- Does the code follow our style guide?

# Test-driving infrastructure

- We are going to use a relatively simple scenario
- We are going to explore many facets of testing
- We are going to follow a test-first, test-driven model

# Our Scenario

- We want a custom home page available on the web.

# Lab 6 – Create a Sandbox Environment

- **Problem:** Applying recipes directly to our workstation is akin to making changes directly in production. We should NOT do that!
- **Success Criteria:** We have an isolated environment to verify the success status of a chef-client run

# Create an apache cookbook

```
$ cd ~/chef-repo/cookbooks
```

# Create an apache cookbook

```
$ chef generate cookbook apache
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
* directory[/home/chef/chef-repo/cookbooks/apache] action create
  - create new directory /home/chef/chef-repo/cookbooks/apache
  - restore selinux security context
* template[/home/chef/chef-repo/cookbooks/apache/metadata.rb] action create_if_missing
  - create new file /home/chef/chef-repo/cookbooks/apache/metadata.rb
  - update content in file /home/chef/chef-repo/cookbooks/apache/metadata.rb from none to 4c0e2d
    (diff output suppressed by config)
  - restore selinux security context
* template[/home/chef/chef-repo/cookbooks/apache/README.md] action create_if_missing
  - create new file /home/chef/chef-repo/cookbooks/apache/README.md
  - update content in file /home/chef/chef-repo/cookbooks/apache/README.md from none to 5c3d3a
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/home/chef/chef-repo/cookbooks/apache/chefignore] action create
...
...
```



# Create an apache cookbook

```
$ cd apache
```

# Create an apache cookbook

```
$ git add .
```

# Create an apache cookbook

```
$ git commit -m "initial apache cookbook"
```

# Chef client success status

- Requirements to verify chef-client success:
  - A target server running the same OS as production

# Chef client success status

- Requirements to verify chef-client success:
  - A target server running the same OS as production
  - A chef-client with access to the cookbook

# Test Kitchen

- Test harness to execute code on one or more platforms
- Driver plugins to allow your code to run on various cloud and virtualization providers
- Includes support for many testing frameworks
- Included with ChefDK



# Test Matrix

- Two operating systems

ubuntu-12.04
centos-6.4

# Test Matrix

- Two operating systems
- One recipe

	<b>default</b>
ubuntu-12.04	apache::default
centos-6.4	apache::default

# Test Matrix

- Two operating systems
- Two recipes

	default	ssl
ubuntu-12.04	apache::default	apache::ssl
centos-6.4	apache::default	apache::ssl

# Test Matrix

- Three operating systems
- Two recipes

	<b>default</b>	<b>ssl</b>
ubuntu-12.04	apache::default	apache::ssl
centos-6.4	apache::default	apache::ssl
ubuntu-14.04	apache::default	apache::ssl

# Configuring the Kitchen



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

```
---
```

```
driver:
  name: vagrant
```

```
provisioner:
  name: chef_zero
```

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

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

```
  attributes:
```

**SAVE FILE!**

# .kitchen.yml

- driver - virtualization or cloud provider

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

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

# .kitchen.yml

- **provisioner** - application to configure the node

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

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

# .kitchen.yml

- platforms - target operating systems

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

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

# .kitchen.yml

- suites - target configurations

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

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

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



# .kitchen.yml

	default
ubuntu-12.04	apache::default
centos-6.4	apache::default

```
---
```

```
driver:
```

```
  name: vagrant
```

```
provisioner:
```

```
  name: chef_zero
```

```
platforms:
```

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

```
  - name: centos-6.4
```

```
suites:
```

```
  - name: default
```

```
    run_list:
```

```
      - recipe[apache::default]
```



# .kitchen.yml

	<b>default</b>	<b>ssl</b>
ubuntu-12.04	apache::default	apache::ssl
centos-6.4	apache::default	apache::ssl

```
---
```

```
driver:
```

```
  name: vagrant
```

```
provisioner:
```

```
  name: chef_zero
```

```
platforms:
```

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

```
  - name: centos-6.4
```

```
suites:
```

```
  - name: default
```

```
    run_list:
```

```
      - recipe[apache::default]
```

```
  - name: ssl
```

```
    run_list:
```

```
      - recipe[apache::ssl]
```



# .kitchen.yml

	default	ssl
ubuntu-12.04	apache::default	apache::ssl
centos-6.4	apache::default	apache::ssl
ubuntu-14.04	apache::default	apache::ssl

```
--  
driver:  
  name: vagrant  
  
provisioner:  
  name: chef_zero  
  
platforms:  
  - name: ubuntu-12.04  
  - name: centos-6.4  
  - name: ubuntu-14.04  
  
suites:  
  - name: default  
    run_list:  
      - recipe[apache::default]  
  - name: ssl  
    run_list:  
      - recipe[apache::ssl]
```

## .kitchen.yml

- The configuration file for your Test Kitchen
- driver – virtualization or cloud provider
- provisioner – application to configure the node
- platforms – target operating systems
- suites – target configurations

# Update .kitchen.yml



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

```
---
```

```
driver:
  name: vagrant
```

```
provisioner:
  name: chef_zero
```

```
platforms:
  - name: centos-6.6
    box: local-chef-development
    box_url: https://s3.amazonaws.com/nathen-chef-lab/uscis.box
```

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

```
    attributes:
```

**SAVE FILE!**

# Move to the apache cookbook directory

```
$ cd ~/chef-repo/cookbooks/apache
```

# List the Test Kitchens

```
$ kitchen list
```

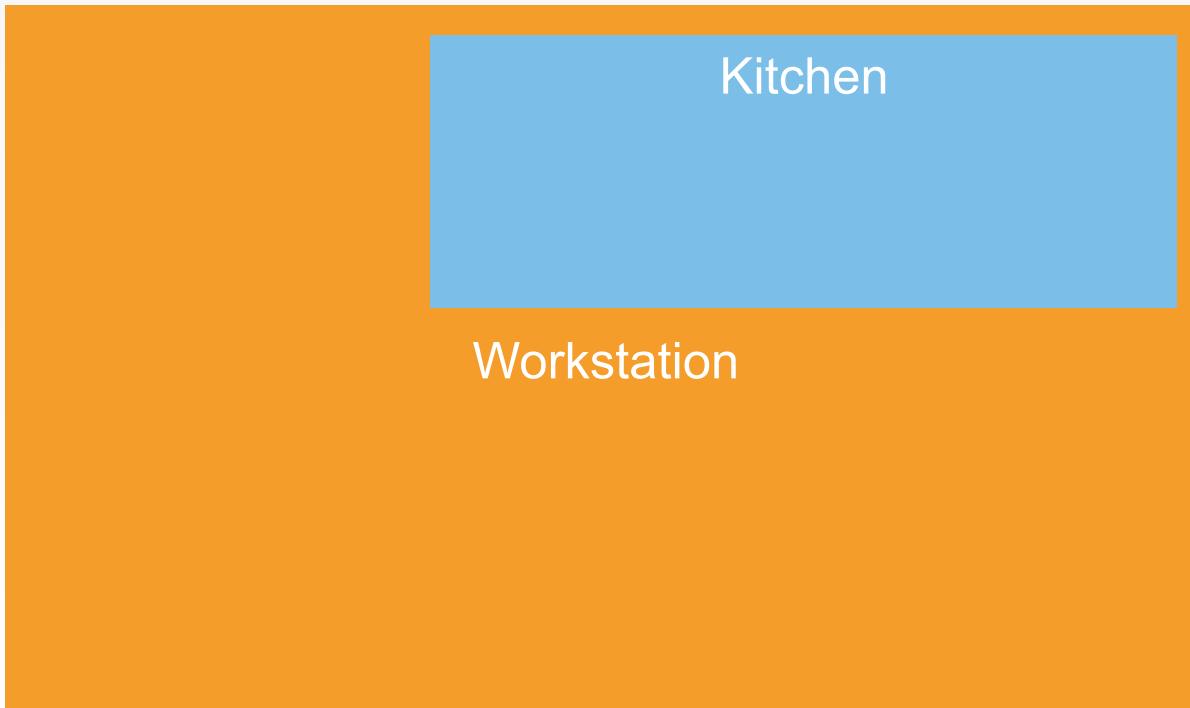
Instance	Driver	Provisioner	Verifier	Transport	Last Action
default-centos-66	Vagrant	ChefZero	Busser	Ssh	<Not Created>

# Create the kitchen

```
$ kitchen create
```

```
----> Starting Kitchen (v1.4.0)
----> Creating <default-centos-66>...
      Bringing machine 'default' up with 'virtualbox' provider...
      ==> default: Importing base box 'local-chef-development'...
==> default: Matching MAC address for NAT networking...
      ==> default: Setting the name of the VM: kitchen-apache-default-centos-66_default_1432619712020_38741
      ==> default: Fixed port collision for 22 => 2222. Now on port 2200.
      ==> default: Clearing any previously set network interfaces...
      ==> default: Preparing network interfaces based on configuration...
          default: Adapter 1: nat
      ==> default: Forwarding ports...
          default: 22 => 2200 (adapter 1)
      ==> default: Booting VM...
      ==> default: Waiting for machine to boot. This may take a few minutes...      default: SSH address:
127.0.0.1:2200
          default: SSH username: vagrant
          default: SSH auth method: private key
```

# Kitchen created



# Login to the kitchen

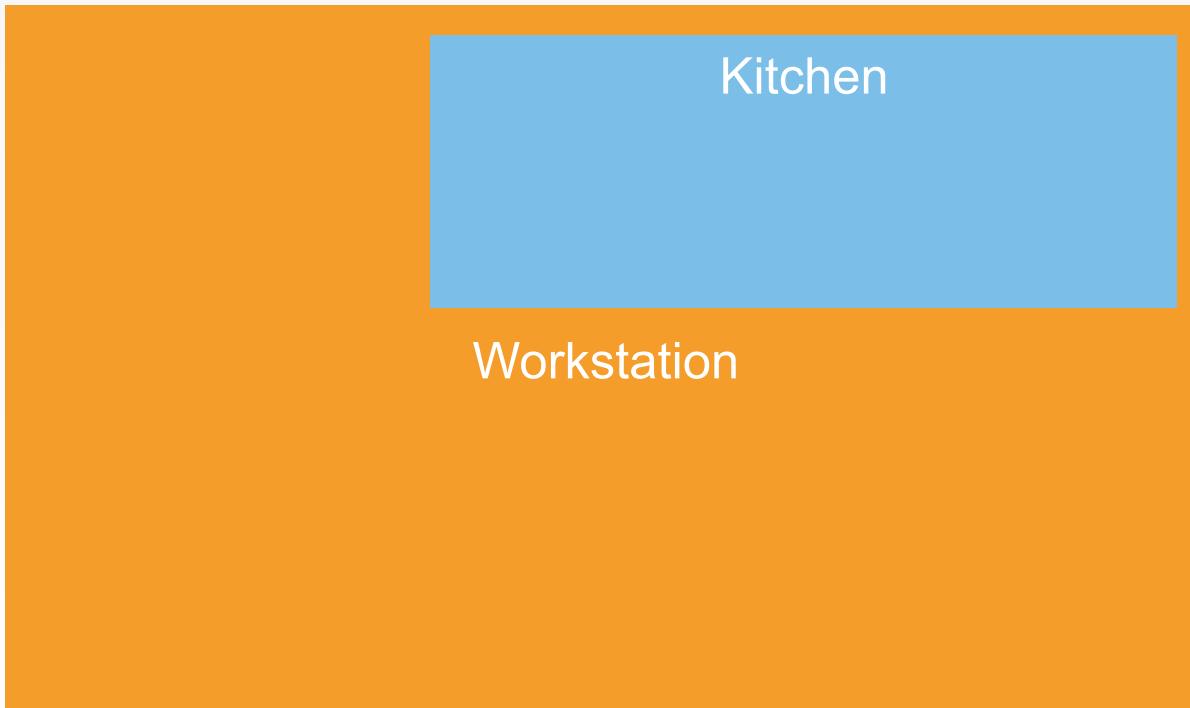
```
$ kitchen login
```

# Login to the kitchen

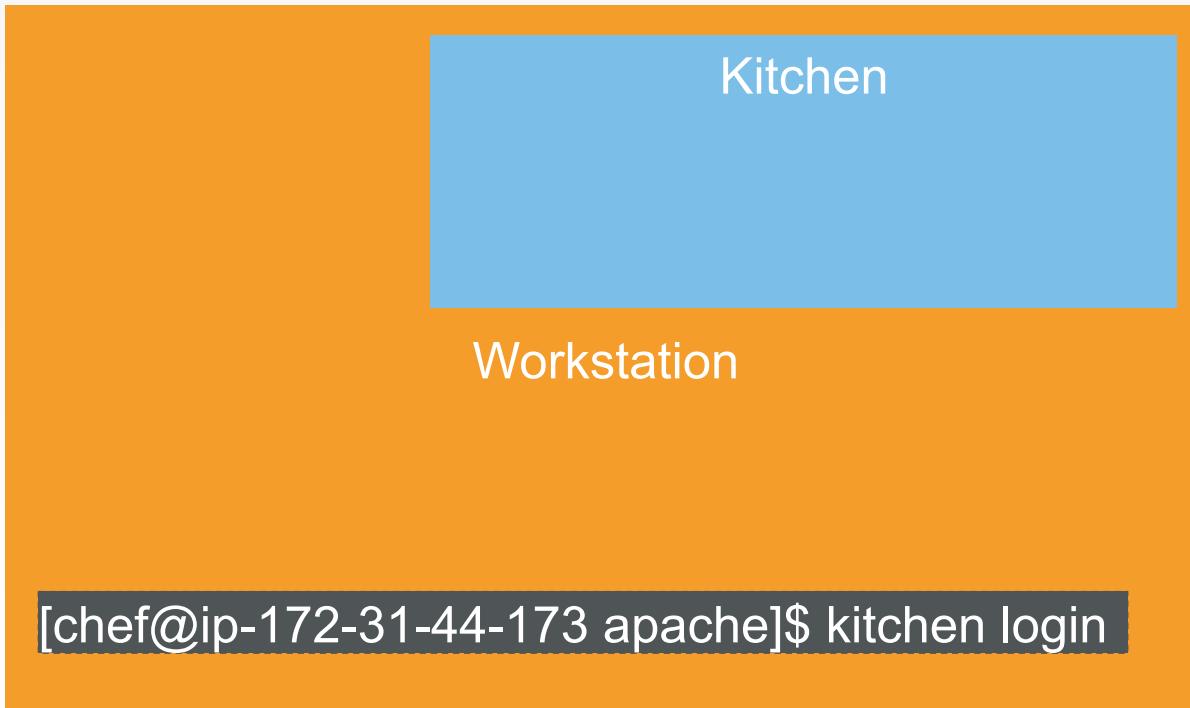
```
$ kitchen login
```

```
Last login: Tue May 26 05:55:46 2015 from 10.0.2.2
```

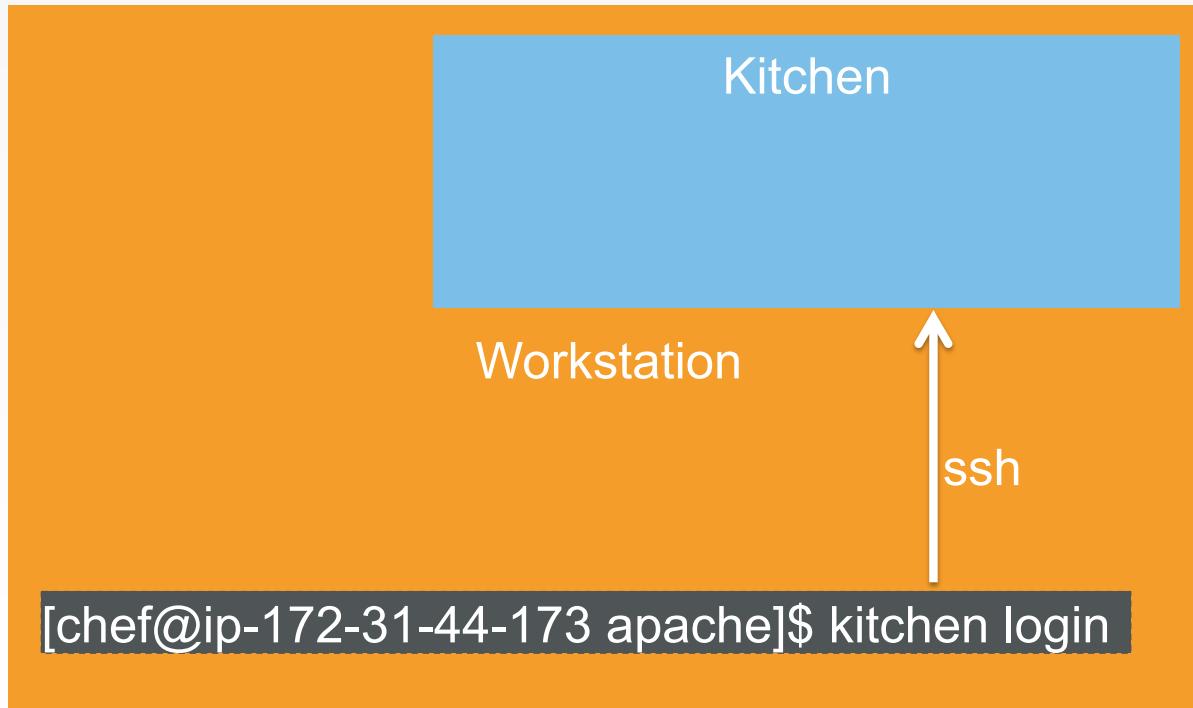
# Kitchen login



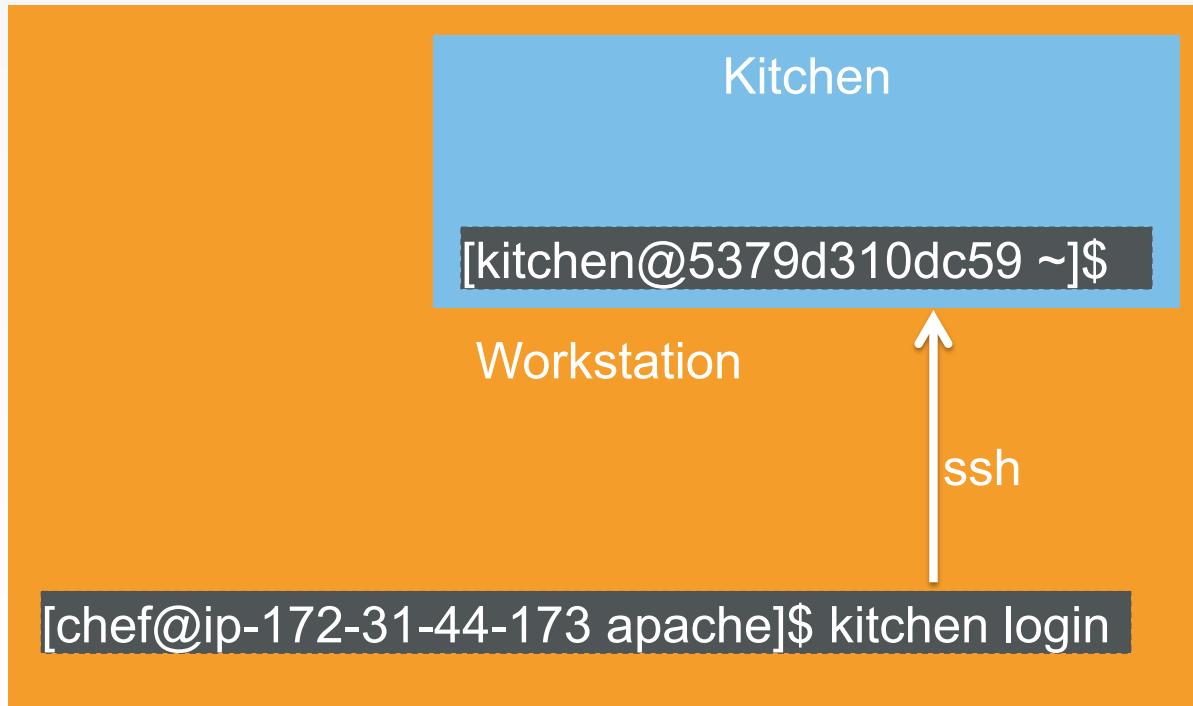
# Kitchen login



# Kitchen login



# Kitchen login



# Chef client success status

- Requirements to verify chef-client success:
  - A target server running the same OS as production
  - A chef-client with access to the cookbook

## Lab 7 – Apply our policy

- **Problem:** We have not applied our policy to the test environment.
- **Success Criteria:** The default apache recipe will be applied in the test environment

# Leave the kitchen

```
$ exit
```

```
logout
```

```
Connection to localhost closed.
```

# Go to the right place

```
$ cd ~/chef-repo/cookbooks/apache
```

# Apply our policy

```
$ kitchen converge
```

```
----> Starting Kitchen (v1.4.0)
----> Converging <default-centos-66>...
      Preparing files for transfer
      Preparing dna.json
      Resolving cookbook dependencies with Berkshelf
3.2.4...
      Removing non-cookbook files before transfer
      Preparing validation.pem
      Preparing client.rb
----> Chef Omnibus installation detected (install only
if missing)
```

# Kitchen converge



# Status Check

- **Success Criteria:** We have an isolated environment to verify the success status of a chef-client run
- **Success Criteria:** The default apache recipe will be applied in the test environment

# Chef Testing

- Did chef-client complete successfully?
- Did the recipe put the node in the desired state?
- Are the resources properly defined?
- Does the code following our style guide?

# Chef Testing

- ✓ Did chef-client complete successfully?
- Did the recipe put the node in the desired state?
- Are the resources properly defined?
- Does the code following our style guide?

# Test Kitchen

- What is a driver?
- What is a provisioner?
- What are platforms?
- What are suites?

# Kitchen Commands

- kitchen list
- kitchen create
- kitchen converge
- kitchen login

## What if...?

- You wanted to test our recipe on Ubuntu as well as CentOS?
- You wanted to remove the kitchen sandbox?

# Test Kitchen

- What questions can I answer for you?



# Verifying node state

Serverspec

v0.1.3



# Chef Testing

- ✓ Did chef-client complete successfully?
- Did the recipe put the node in the desired state?
- Are the resources properly defined?
- Does the code following our style guide?

# Login to the kitchen

```
$ kitchen login
```

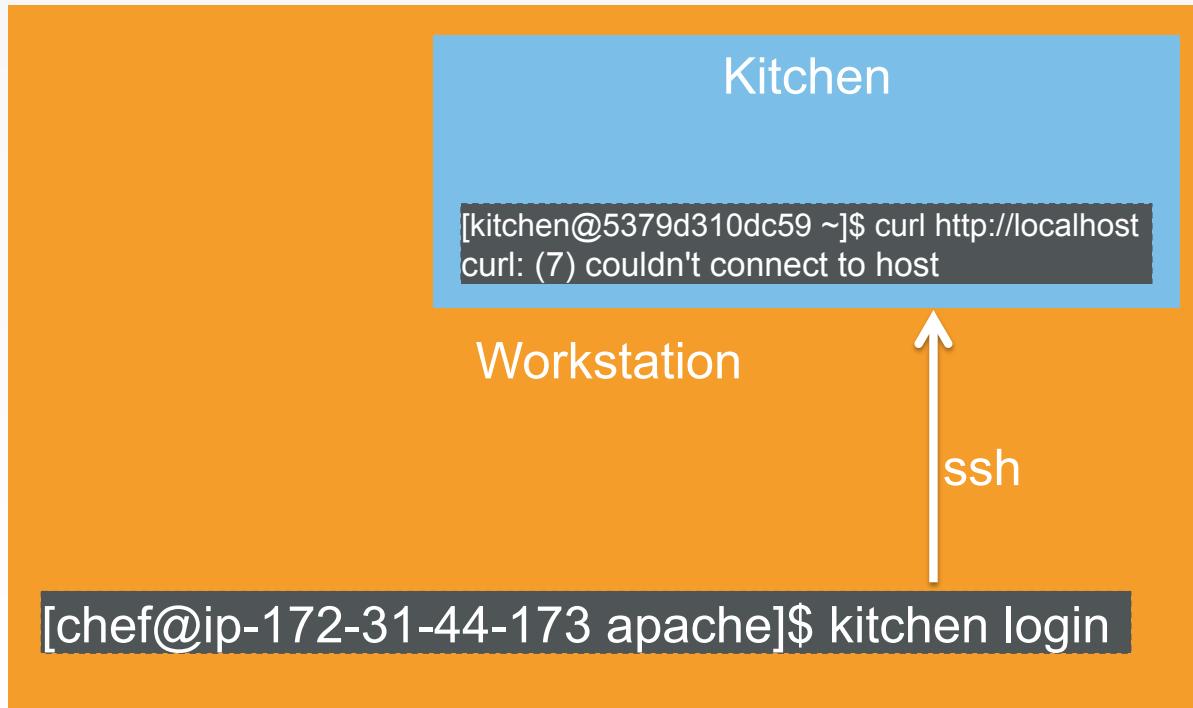
```
Last login: Tue May 26 05:55:46 2015 from 10.0.2.2
```

# Manually inspect the test node

```
$ curl http://localhost
```

```
curl: (7) couldn't connect to host
```

# Kitchen login



## Lab 9 – Verify node state

- **Problem:** Manually verifying the state of the test node is tedious and error-prone.
- **Success Criteria:** The end state of the node is automatically tested.

# Serverspec

- Write tests to verify your servers
- Not dependent on Chef
- Defines many resource types
  - package, service, user, etc.
- Works well with Test Kitchen
- <http://serverspec.org/>



# Leave the Kitchen

```
$ exit
```

```
logout
```

```
Connection to localhost closed.
```

# Move to the proper directory

```
$ cd ~/chef-repo/cookbooks/apache
```

# Default location for tests

- Test Kitchen will look in the test / integration directory for test-related files

# Suite subdirectory

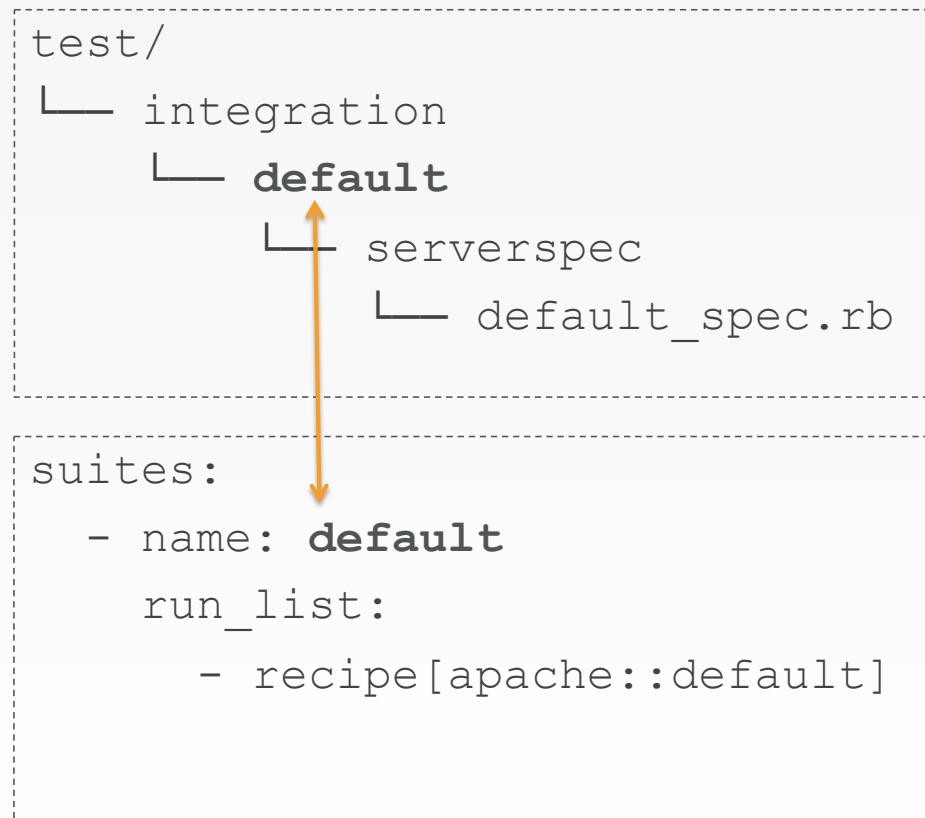
- The next level subdirectory will match the suite name.

```
test/
└ integration
    └ default
        └ serverspec
            └ default_spec.rb
```

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

# Suite subdirectory

- The next level subdirectory will match the suite name.



# Busser subdirectory

- Test Kitchen utilizes **bussers** to manage test plugins.
- We'll be using the **serverspec** plugin

```
test/
└ integration
    └ default
        └ serverspec
            └ default_spec.rb
```

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

# Write a Serverspec test



**OPEN IN EDITOR:** `test/integration/default/serverspec/default_spec.rb`

```
require 'spec_helper'

describe 'apache::default' do
  # Serverspec examples can be found at
  # http://serverspec.org/resource_types.html
  it 'does something' do
    skip 'Replace this with meaningful tests'
  end
end
```

**SAVE FILE!**

# Generic Expectation Form

```
describe "<subject>" do
  it "<description>" do
    expect(thing).to eq result
  end
end
```

# Awesome Expectations



**OPEN IN EDITOR:** test/integration/default/serverspec/default\_spec.rb

```
require 'spec_helper'

describe 'apache::default' do
  # Serverspec examples can be found at
  # http://serverspec.org/resource_types.html
  it "is awesome" do
    expect(true).to eq true
  end
end
```

**SAVE FILE!**

# Run the serverspec test

```
$ kitchen verify
```

```
----> Running serverspec test suite
      /opt/chef/embedded/bin/ruby -I/tmp/busser/suites/serverspec -I/tmp/
busser/gems/gems/rspec-support-3.1.2/lib:/tmp/busser/gems/gems/rspec-
core-3.1.7/lib /opt/chef/embedded/bin/rspec --pattern /tmp/busser/suites/
serverspec/\*\*/\* spec.rb --color --format documentation --default-path /
tmp/busser/suites/serverspec

apache
    is awesome

Finished in 0.02823 seconds (files took 0.99875 seconds to load)
1 example, 0 failures
Finished verifying <default-centos-64> (0m5.03s).
```

## How would you test our criteria?

- We want a custom home page available on the web.

# What is success?

- Package is installed?
- Page is displayed?
- What else?

# Verify package is installed



**OPEN IN EDITOR:** test/integration/default/serverspec/default\_spec.rb

```
describe "apache" do
  it "is awesome" do
    expect(true).to eq true
  end

  it "is installed" do
    expect(package("httpd")).to be_installed
  end
end
```

**SAVE FILE!**

# Exercise the test

```
$ kitchen verify
```

```
apache
    is awesome
    is installed (FAILED - 1)
```

```
Failures:
```

```
1) apache is installed
   Failure/Error: expect(package("httpd")).to
be_installed
   expected Package "httpd" to be installed
   /bin/sh -c rpm\ -q\ httpd
   package httpd is not installed
```

# Test is failing, make it pass

- Test-driven development involves
  - Write a test to verify something is working
  - Watch the test fail
  - Write just enough code to make the test pass
  - Repeat

# Update our cookbook



**OPEN IN EDITOR:** `~/chef-reop/cookbooks/apache/recipes/default.rb`

```
package "httpd"
```

**SAVE FILE!**

# Converge the node again

```
$ kitchen converge
```

```
----> Converging <default-centos-64>...
      Preparing files for transfer
      Resolving cookbook dependencies with Berkshelf 3.1.5...
      Removing non-cookbook files before transfer
      Transferring files to <default-centos-64>
      [2014-11-10T09:20:26+00:00] INFO: Starting chef-zero on host localhost, port 8889
with repository at repository at /tmp/kitchen
      One version per cookbook

      [2014-11-10T09:20:26+00:00] INFO: Forking chef instance to converge...
      Starting Chef Client, version 11.16.4
      [2014-11-10T09:20:27+00:00] INFO: *** Chef 11.16.4 ***
      [2014-11-10T09:20:27+00:00] INFO: Chef-client pid: 571
      ...
```



# Exercise the test

```
$ kitchen verify
```

```
apache
    is awesome
    is installed
```

```
    Finished in 0.48165 seconds (files took 1.05
seconds to load)
```

```
    2 examples, 0 failures
```

```
    Finished verifying <default-centos-64>
(0m5.64s).
```

```
----> Kitchen is finished. (0m11.84s)
```

## What else will you test?

- Is the service running?
  - Is the port accessible?
  - Is the expected content being served?
- 
- Make sure everything works from a fresh kitchen, too!

# Time to hack!



<https://www.flickr.com/photos/peterpearson/424047087>

# Extend the Serverspec test



**OPEN IN EDITOR:** [test/integration/default/serverspec/default\\_spec.rb](#)

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

  it "is running" do
    expect(service 'httpd').to be_running
  end

  it "is listening on port 80" do
    expect(port 80).to be_listening
  end

  it "displays a custom home page" do
    expect(command("curl localhost").stdout).to match /hello/
  end
end
```

**SAVE FILE!**

# Verify the kitchen

```
$ kitchen verify
```

```
apache
  is installed
  is running
  is listening on port 80
  displays a custom home page
```

```
Finished in 0.3968 seconds
4 examples, 0 failures
Finished verifying <default-centos-64> (0m4.25s).
```

# Kitchen Workflow

- kitchen create
- kitchen converge
- kitchen verify
- kitchen destroy
- All at once with kitchen test

# Chef Testing

- ✓ Did chef-client complete successfully?
- ✓ Did the recipe put the node in the desired state?
  - Are the resources properly defined?
  - Does the code following our style guide?

## Lab 10 – Verify node state in motd

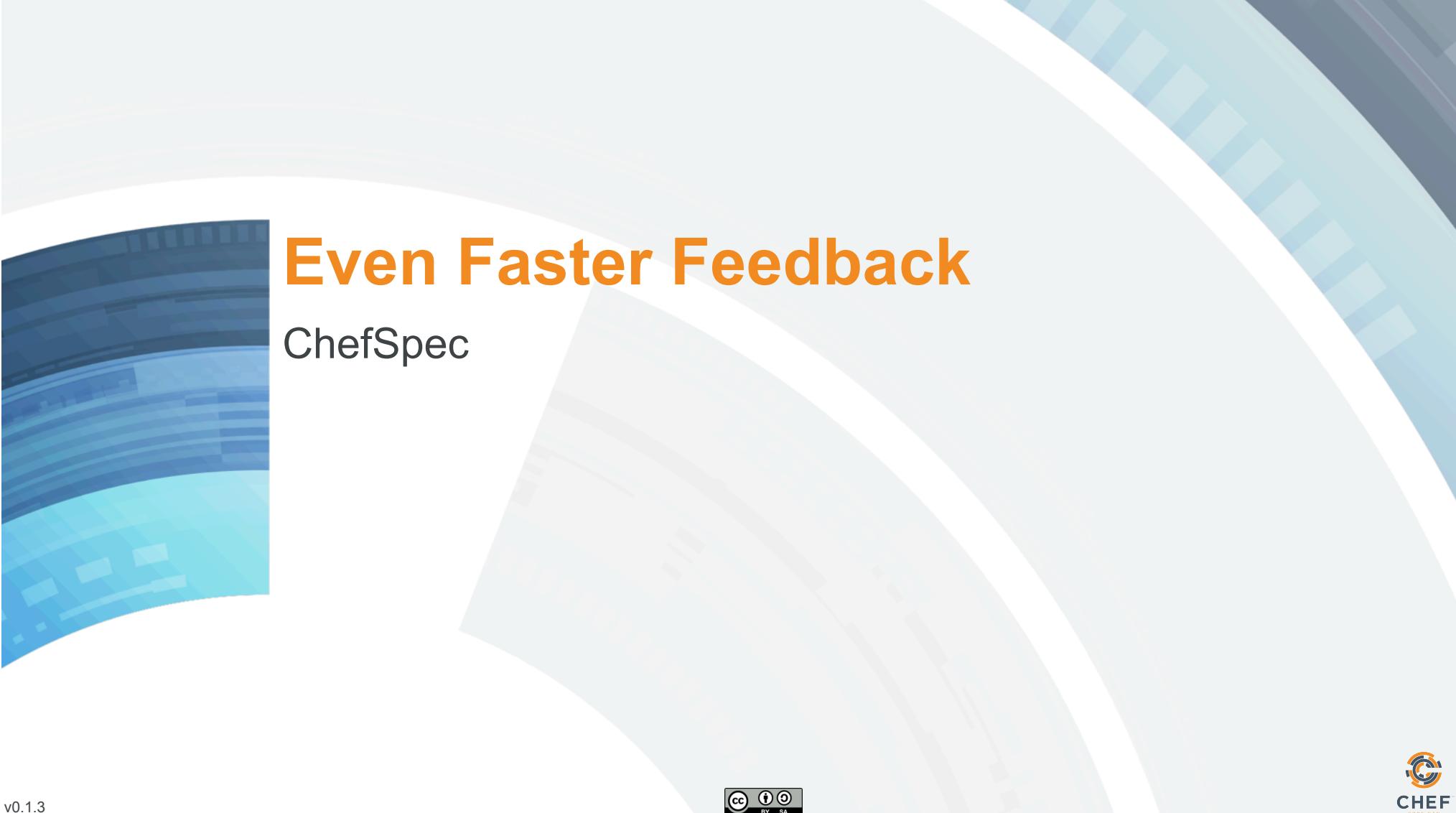
- Add serverspec tests to your motd cookbook

## Verifying the node

- What command will show you the current state of your test kitchen suites?
- Can you view your kitchen's custom home page from your laptop's browser? How? Why?
- Is it important to start with a fresh kitchen?

# Verifying Node State

- What questions can I answer for you?



# Even Faster Feedback

ChefSpec

v0.1.3



# Chef Testing

- ✓ Did chef-client complete successfully?
- ✓ Did the recipe put the node in the desired state?
  - Are the resources properly defined?
  - Does the code following our style guide?

# This is too slow!

- To test our code, we need to spin up a test kitchen, converge a node, execute some tests.
- Our simple test case takes about 2 minutes to fully execute.

# Properly configured resources

- We need a way to verify that the resources in our recipes are properly configured
- We want to get faster feedback

# Lab 9 – Verify the resources

- **Problem:** We should be able to catch errors before we need to converge a node
- **Success Criteria:** Catch a typo prior to converge

# ChefSpec

- Test before you converge
- Get feedback on cookbook changes without the need for target servers

The screenshot shows the GitHub Pages version of the ChefSpec project. It has a dark background with white text. At the top is the title "ChefSpec". Below it is a subtitle: "Write RSpec examples and generate coverage reports for Chef recipes!". There are three red buttons with white text: "Download ZIP", "Download TAR", and "View On GitHub". At the bottom, it says "This project is maintained by [sethvargo](#)".

Hosted on [GitHub Pages](#)

<http://sethvargo.github.io/chefspec/>

## ChefSpec

gem v4.0.2 build passing dependency

ChefSpec is a unit testing framework for examples and get fast feedback on cool servers.

ChefSpec runs your cookbook locally us primary benefits:

- It's really fast!
- Your tests can vary node attribute under varying conditions.

## What people are saying

*I just wanted to drop you a line to say*

*OK chefspec is my new best friend. L*

Chat with us - #chefspec on Freenode



# Make a directory for our ChefSpec tests

```
$ cd ~/chef-repo/cookbooks/apache
```

# Make a directory for our ChefSpec tests

```
$ mkdir -p spec/unit
```

# Write a ChefSpec test



**OPEN IN EDITOR:** spec/unit/default.rb

```
require 'chefspec'

describe 'apache::default' do
  let(:chef_run) do
    ChefSpec::Runner.new.converge(described_recipe)
  end

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

**SAVE FILE!**

# Run the ChefSpec tests

```
$ rspec spec/unit/*.rb
```

```
.
```

```
Finished in 0.00865 seconds (files took 5.5 seconds to load)
1 example, 0 failures
```

# Break the cookbook



**OPEN IN EDITOR:** recipes/default.rb

```
package "http"
```

```
service "httpd" do
  action :start
end
```

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

**SAVE FILE!**

# Run the ChefSpec tests

```
$ rspec spec/unit/*.rb
```

```
F
```

```
Failures:
```

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

  package[http]
```

```
# ./spec/unit/default_spec.rb:9:in `block (2 levels) in <top (required)>'
```

```
Finished in 0.00847 seconds (files took 4.85 seconds to load)
```

```
1 example, 1 failure
```

```
Failed examples:
```

```
rspec ./spec/unit/default_spec.rb:8 # apache::default installs apache
```



# Fix the cookbook



**OPEN IN EDITOR:** recipes/default.rb

```
package "httpd"
```

```
service "httpd" do
  action :start
end
```

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

**SAVE FILE!**

# Time to hack!



<https://www.flickr.com/photos/peterpearson/424047087>

# Chef Testing

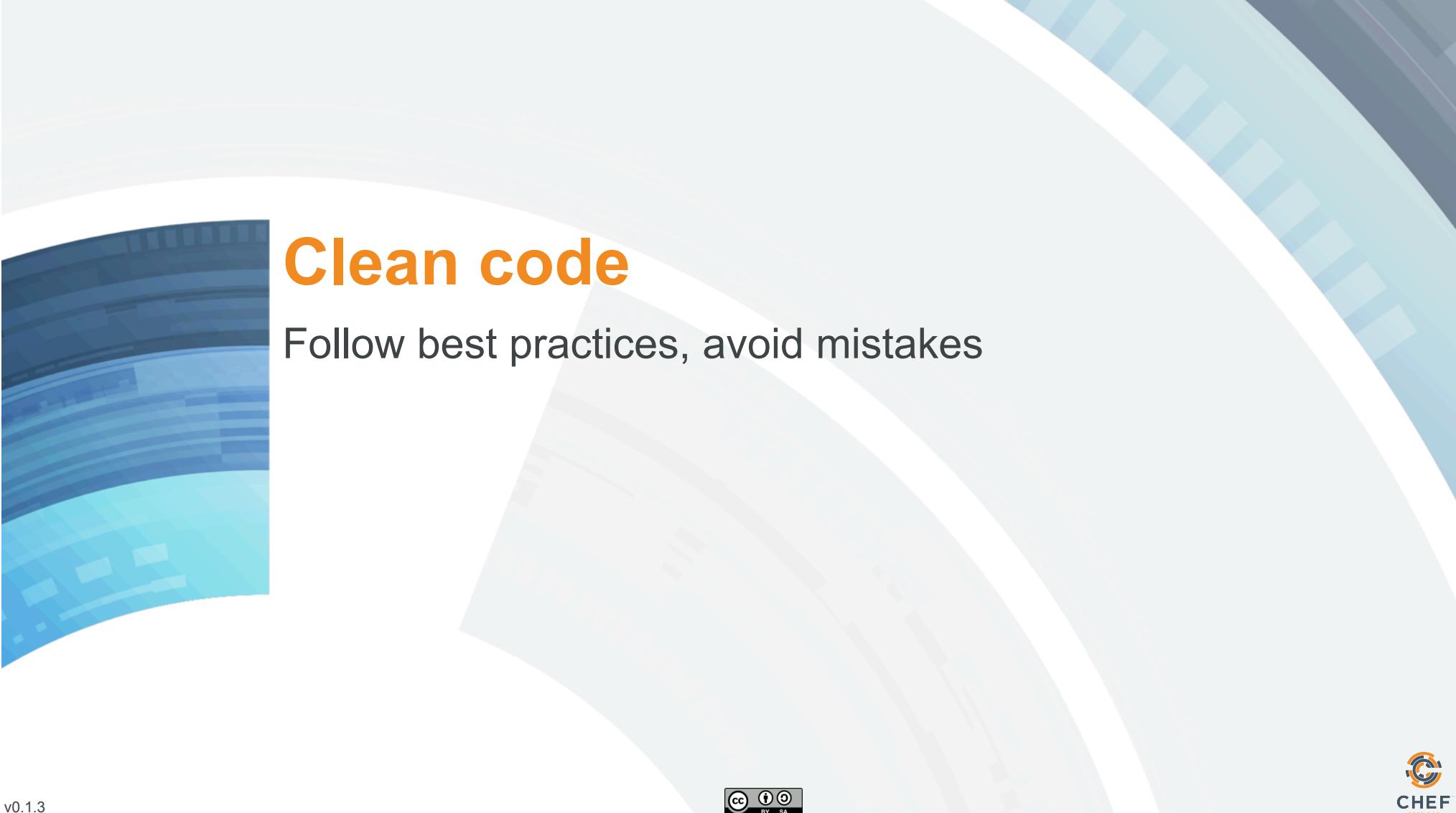
- ✓ Did chef-client complete successfully?
- ✓ Did the recipe put the node in the desired state?
- ✓ Are the resources properly defined?
- Does the code following our style guide?

# ChefSpec

- What is the primary difference between ChefSpec and ServerSpec?
- Why use ChefSpec if you already have ServerSpec tests?
- Do passing ChefSpec tests ensure your recipe will work?
- How would you feel about removing some of your ServerSpec tests now that you have ChefSpec in place?

# ChefSpec

- What questions can I answer for you?



# Clean code

Follow best practices, avoid mistakes

# Foodcritic

- Check cookbooks for common problems
- Style, correctness, deprecations, etc.
- Included with ChefDK



<http://www.foodcritic.io/>

# Change our recipe



**OPEN IN EDITOR:** recipes/default.rb

```
package_name = "httpd"

package "#{package_name}"

service "httpd" do
  action :start
end

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

**SAVE FILE!**

# Run Foodcritic

```
$ foodcritic .
```

```
FC002: Avoid string interpolation  
where not required: ./recipes/  
default.rb:7
```

# Chef Testing

- ✓ Did chef-client complete successfully?
- ✓ Did the recipe put the node in the desired state?
- ✓ Are the resources properly defined?
- ✓ Does the code following our style guide?

# Foodcritic

- What rules have been deprecated?
- What does Foodcritic return on success?

# Foodcritic

- What questions can I answer for you?



# Wrap Up

v0.1.3



# Course Objectives

- After completing this course you will be able to:
  - Automate common infrastructure tasks with Chef
  - Verify your automation code BEFORE it runs in production
  - Describe Chef's various tools
  - Apply Chef's primitives to solve your problems

# Tool Survey

- chef-apply
- chef
- chef-client in local mode
- Test Kitchen
- Docker
- Serverspec
- ChefSpec
- Foodcritic

# Vocabulary

- Resources
- Recipes
- Cookbooks

# Resources

- Package
- Service
- File
- Template

## But wait...

- ...there's more, so much more!
- How much time do we have left? I could go on for days!

# Further Resources

- [learnchef.com](http://learnchef.com)
  - Guided tutorials
  - Chef Fundamental Series
- Upcoming Training
  - [chef.io/blog/events/category/training-events/](http://chef.io/blog/events/category/training-events/)

# Chef Fundamentals Q & A Forum

- Chef Fundamentals Google Group Q&A Forum
- <http://bit.ly/ChefFundamentalsForum>
- Join the group and post questions

# A list of URLs

- <http://chef.io>
- <http://docs.chef.io>
- <http://supermarket.chef.io>
- <http://youtube.com/getchef>
- <http://lists.opscode.com>
- [irc.freenode.net: #chef, #chef-hacking](#)
- Twitter: [@chef](#) [#getchef](#), [@learnchef](#) [#learnchef](#)

# Food Fight Show

- [foodfightshow.org](http://foodfightshow.org)
- Podcast where DevOps Chefs Do Battle
- Best practices for working with Chef



# What questions do you have?

- Chef Server
- Roles
- Environments
- Data Bags
- Bootstrapping new nodes
- Open source projects
- Working with IaaS providers
- chef-provisioner
- Search
- Suspenders?!
- Thank You!
- @nathenharvey

# What else would you like to work on?

- Make the cookbook work for ubuntu?
- Explore Chef Server
- Learn about other top-level Chef Objects
  - Node
  - Roles
  - Environments
  - Data Bags

# Time to hack!



<https://www.flickr.com/photos/peterpearson/424047087>