# Introduction to Chef

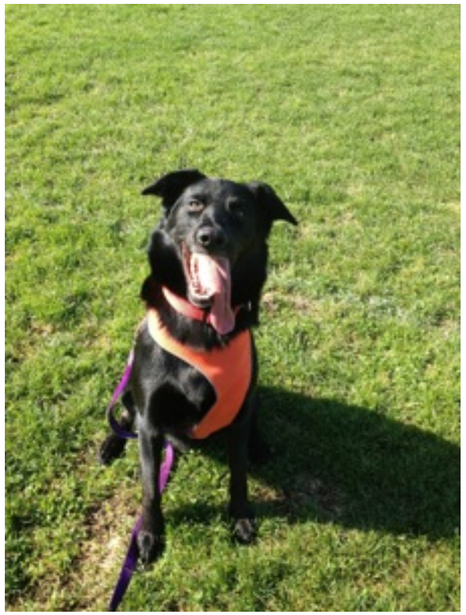
training@getchef.com

# Pre-Requisites

- Laptop with an SSH client
- Basic Unix command line knowledge
- Wireless: UCSFguest

# Introductions

# Course Instructor



Jennifer Davis

Solutions Engineer

Email:sigje@chef.io Twitter: @sigje

# Communicate and Collaborate

- Hashtags: **#TSWS15 #getchef**

# Instructor Introduction

- **Name:** Franklin Webber
- **Current job role:** Developer / Instructor
- **Previous job roles/background:** ARMY SGT, QA, SDET
- **Experience with Chef/Config Management:** Deployments with Ruby on Rails Applications
- **Favorite Text Editor:** Sublime Text

# Introduce Yourselves

- Name
- Current job role
- Previous job roles/background
- Experience with infrastructure/automation
- Experience with Chef and/or config management

# Course Objectives and Style

# Course Objectives

- Upon completion of this course you will be able to
  - Automate common infrastructure tasks with Chef
  - Describe some of Chef's tools
  - Apply Chef's primitives to solve your problems
  - Know where to go for more info

# Learning Chef

- You bring domain expertise about your business and problems to solve.
- Chef provides a framework to solve problems.
- Together, we 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
- The remaining 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 issues on the spot

# Just an Introduction

- This course is an introduction to using Chef to manage your infrastructure.
- We cover a lot of material!
- Any discussion items that are too far off topic will be captured
  - We'll return to these items as time permits

# Setting Expectations

- Experience Level: Beginner
- Flexible content based on class
- It’s ok to leave (and come back).

# Agenda

# Topics

- Introduction to Infrastructure Automation
- Overview of Chef
- Workstation Setup
- Resources
- Describing Policies
- Wrap Up

# Breaks!

- We'll take a break as often as we need them.
- We'll take a break for lunch.

# Legend

# Legend: Example Terminal Command and Output

```
$ ifconfig
```

```
lo0:
flags=8049<UP,LOOPBACK,RUNNING,MULTICA
ST> mtu 16384
options=3<RXCSUM,TXCSUM>
inet6 fe80::1%lo0 prefixlen 64
scopeid 0x1
inet 127.0.0.1 netmask 0xff000000
```

# Legend: Example of editing a file on your workstation

**OPEN IN EDITOR:** ~/hello_world

```
Hi!

I am a friendly file.
```

**SAVE FILE!**

# Infrastructure Automation

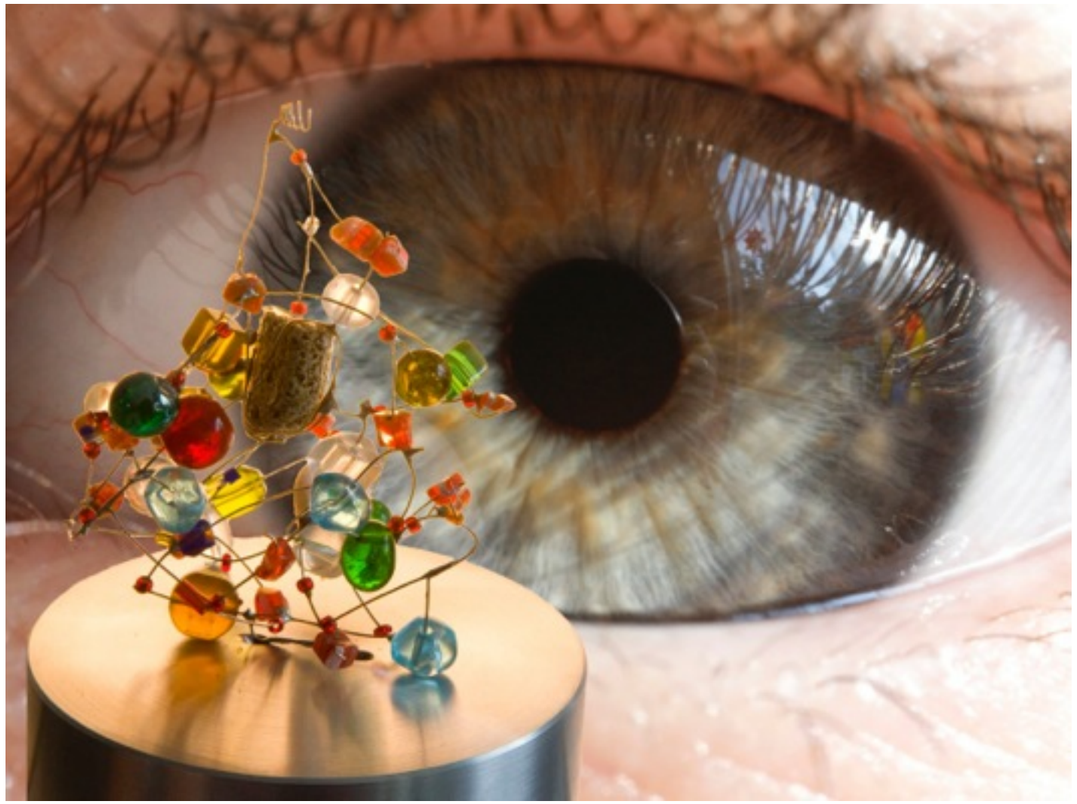
# What is Infrastructure Automation?

# What is infrastructure?

# Applications


- MongoDB
- Apache
- Java
- Drupal
- WordPress
- cakephp

# Applications - Complexity

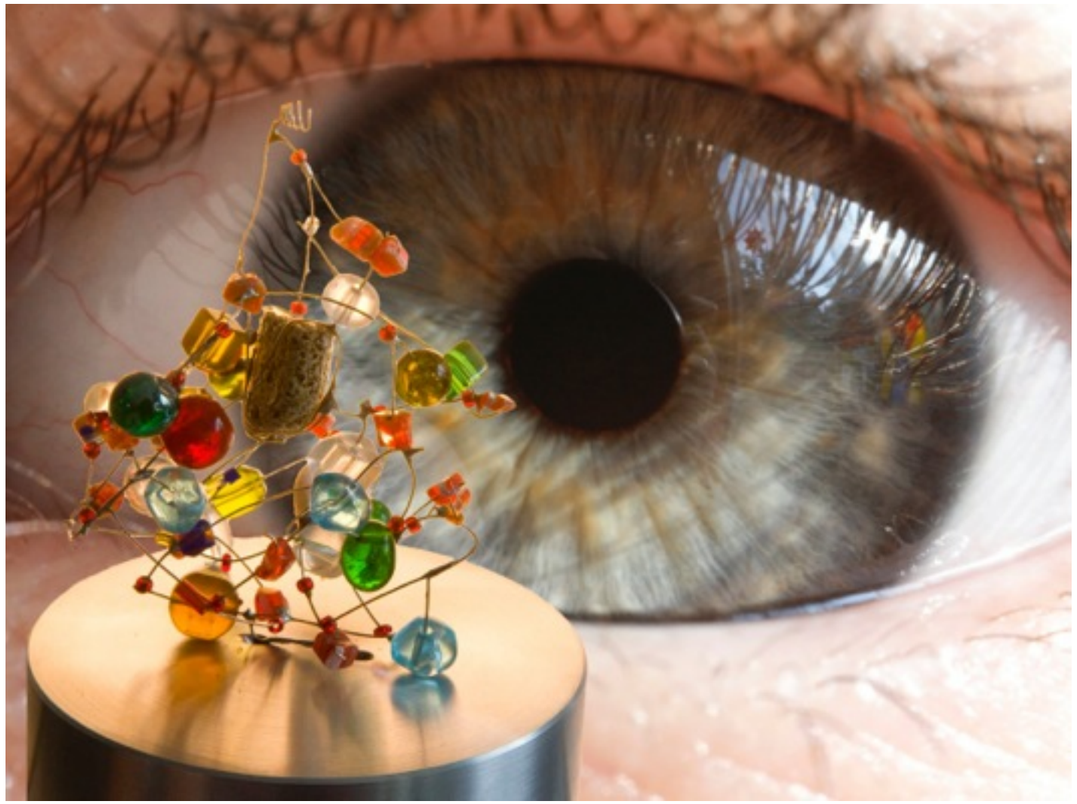


- Single service == multiple applications.
- Single application == a component of service.

# Configurations


- Operating Systems
- Software
- Network
- Storage

# Configurations - Complexity

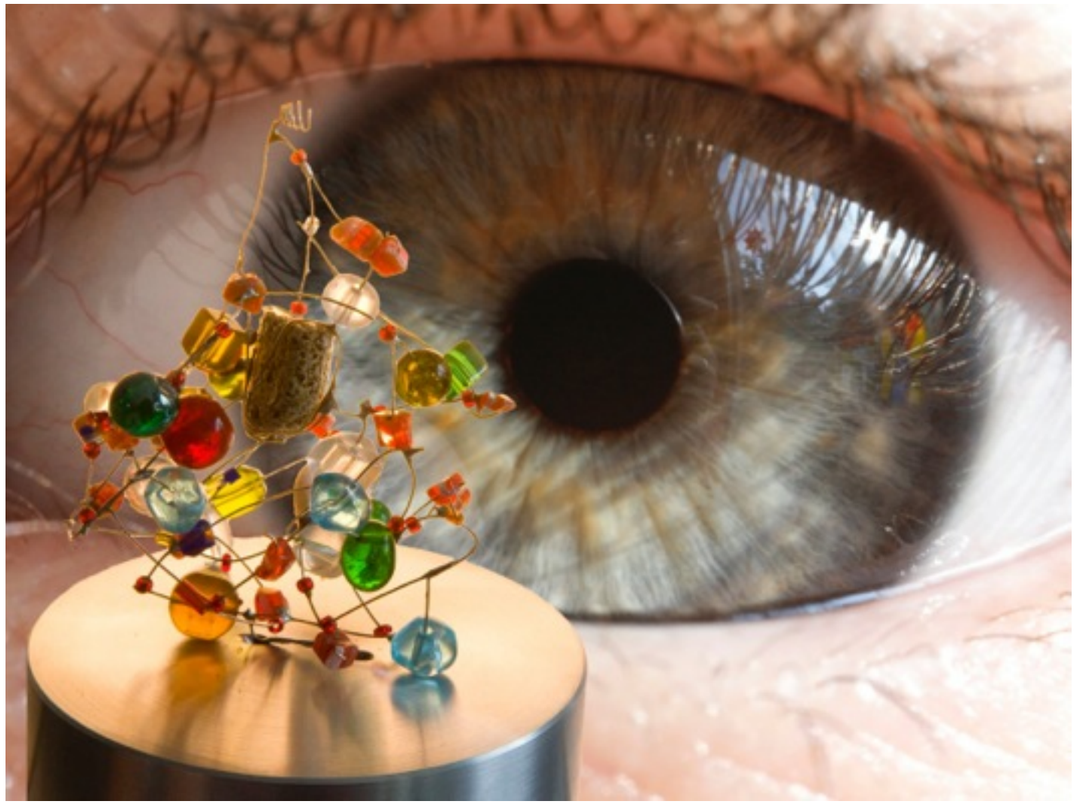


- Multiple services using same software with different configurations.
- Different Operating Systems requiring same software (and user experience).

# Access Control


- Users
- Groups
- Organizations

# Access Control - Complexity



- Different users access to different systems.
- External versus internal.
- Compliance.

# Data

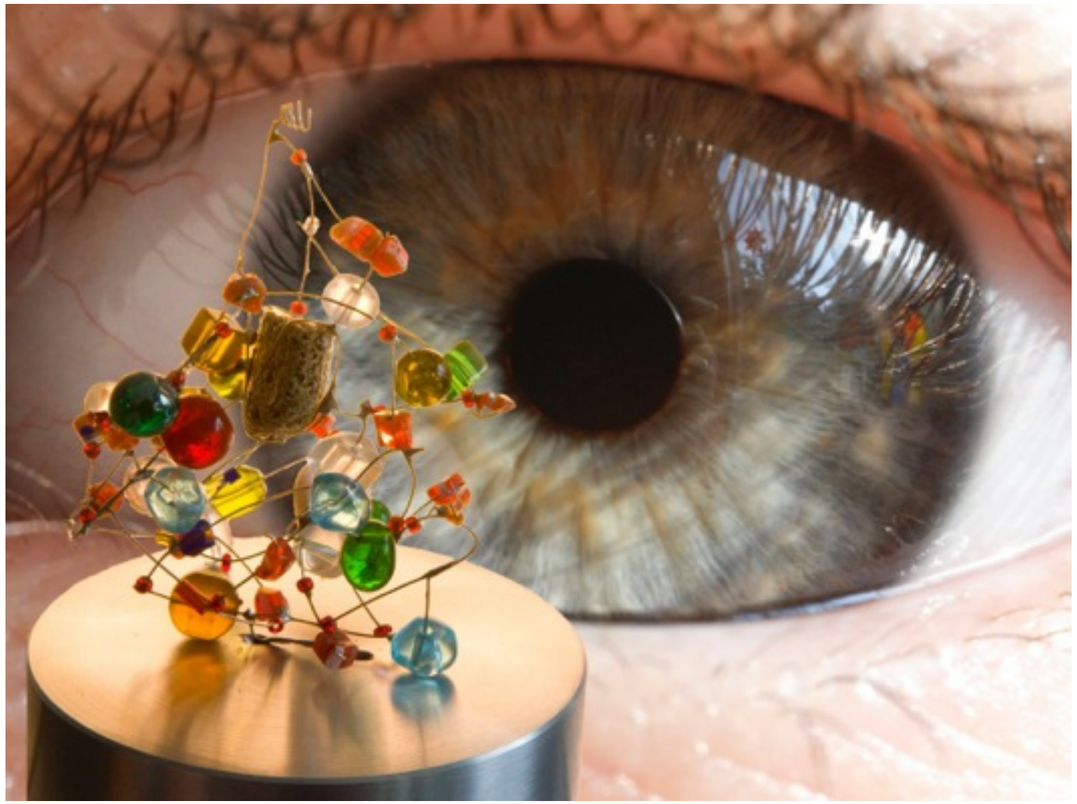
- **Tweets**
- **Profiles**
- **Emails**

**Specific to your organization.**

# Servers

- physical hardware
- cloud services
- virtual

# Servers - Complexity



- Specialization in hardware.
- Lack of standardization.

# Network & Storage

- routers
- switches
- firewalls
- tape libraries
- NAS/SAN attached storage

# People

# What is automation?

# Automaton

- **acting of one's own will.**

# Automation

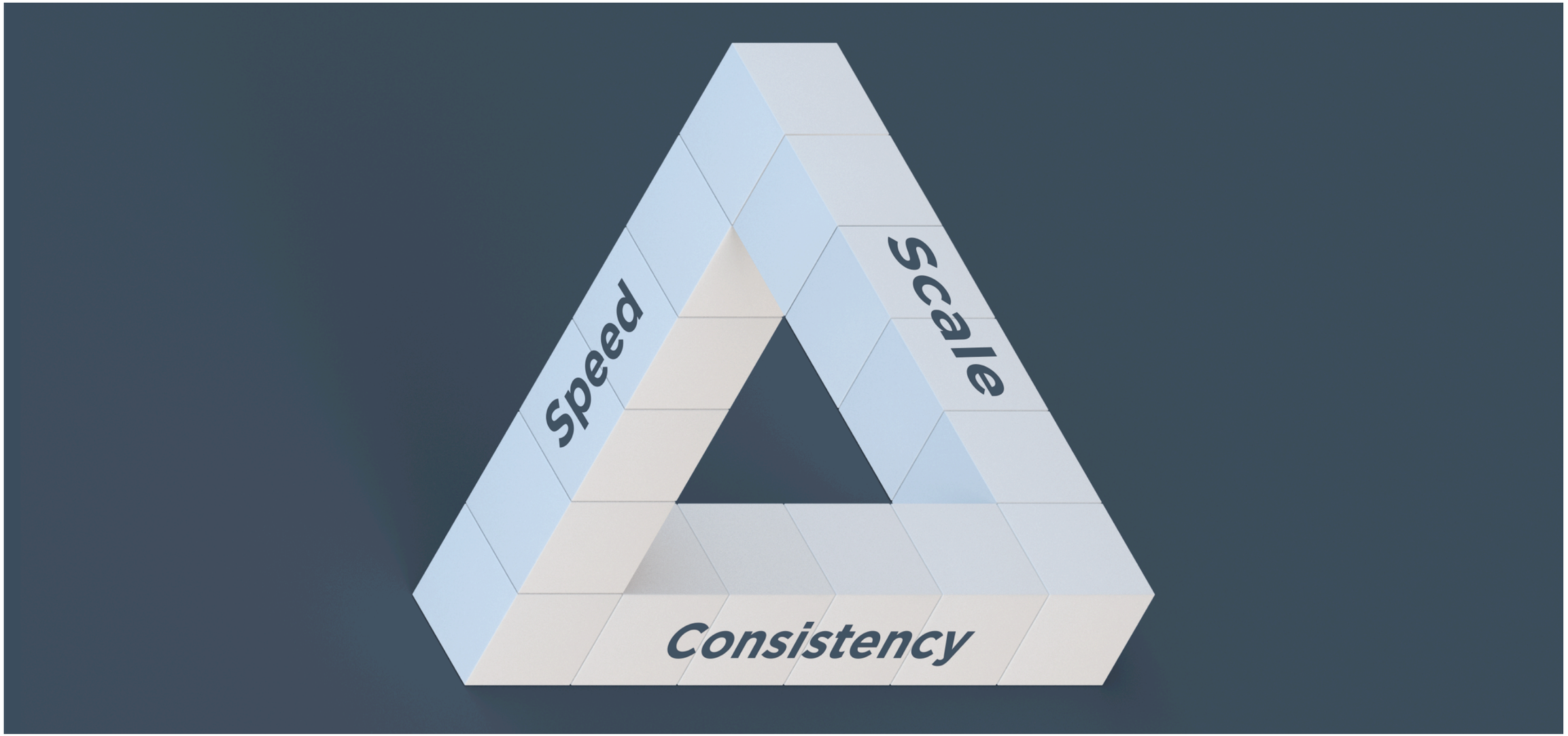
- reducing labor, energy, and materials
- improve quality, precision, accuracy

# What is infrastructure automation?

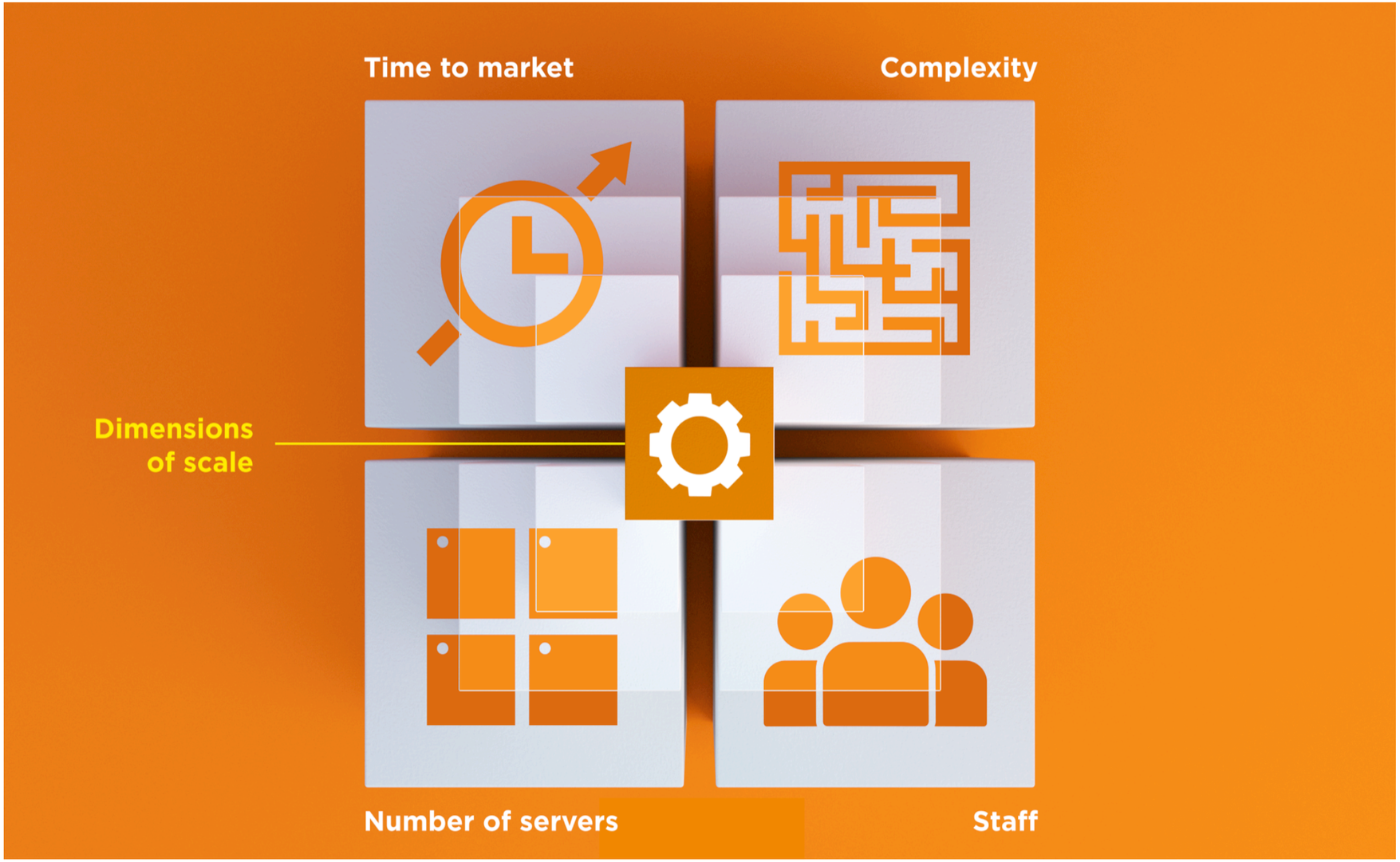
**Infrastructure Automation is creating control systems that reduce the burden on people to manage services and increase the quality, accuracy and precision of a service to the consumers of the service.**

# Overview of Chef

# Benefits of Automation



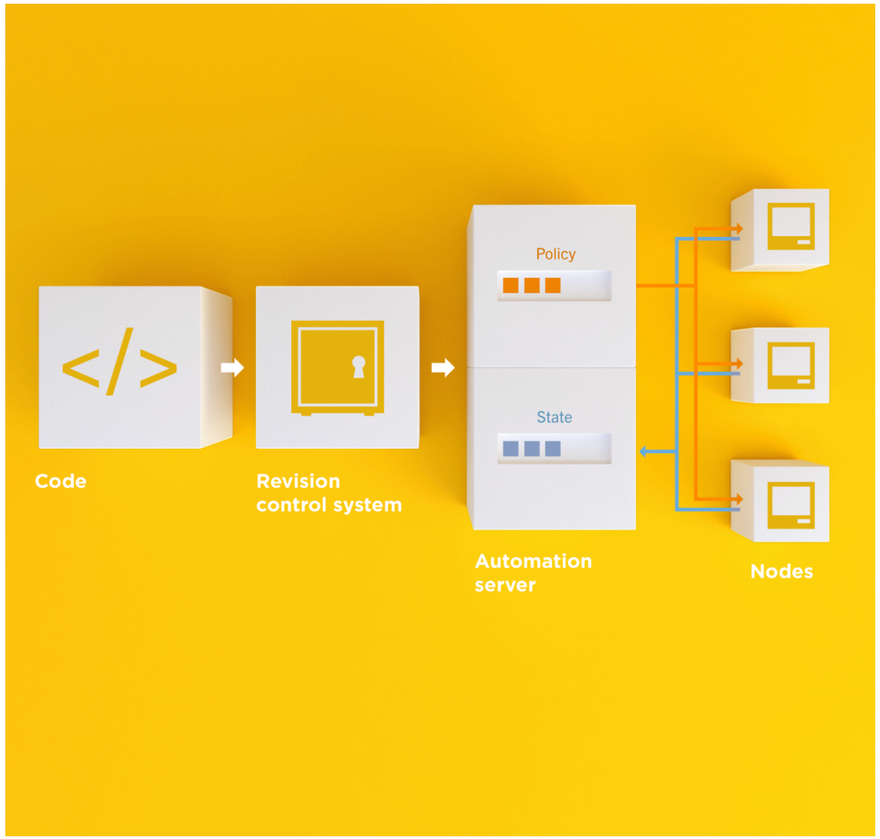
# Dimensions of Scale



# Automation Platform

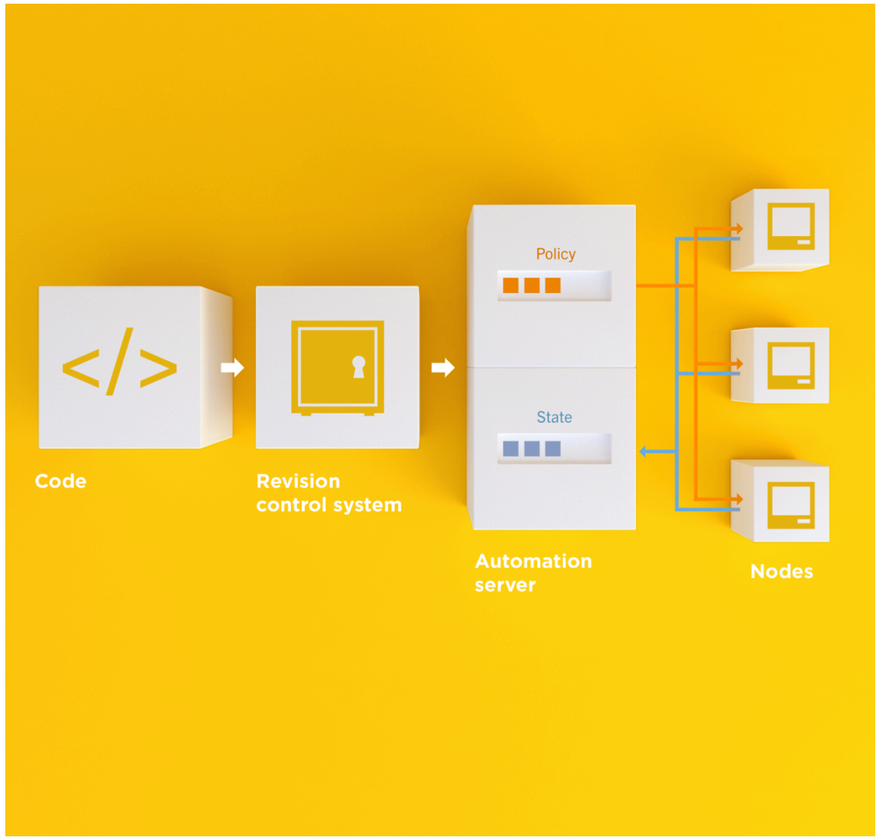
- Dependable view of your network's state.
- Handles complex dependencies among nodes.
- Fault tolerant.
- Secure.
- Multi-platform.
- Supports multiple compute platforms.
- Foundation for innovation.

# Infrastructure as Code



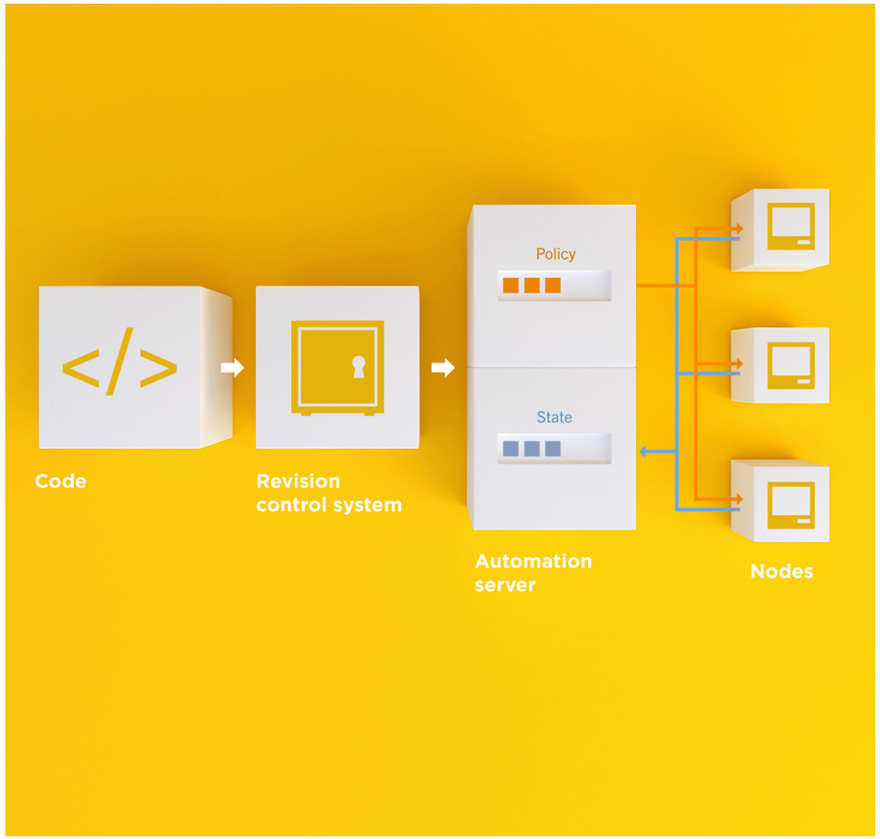

- Programmatically provision and configure components.

# Infrastructure as Code



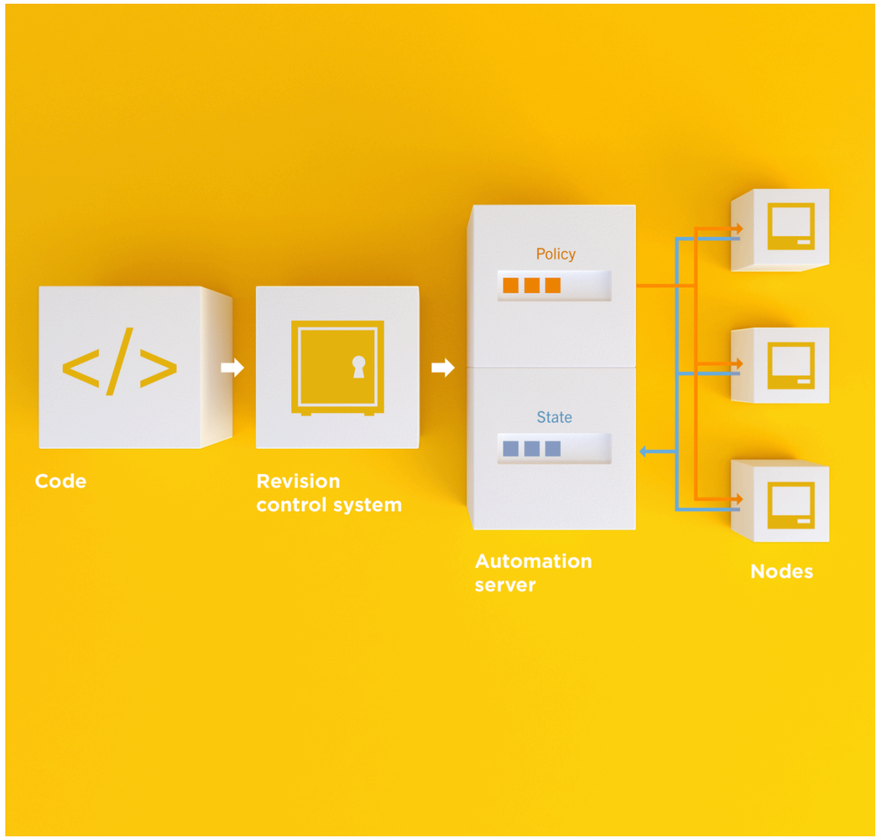

- Treat like any code base.

# 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.

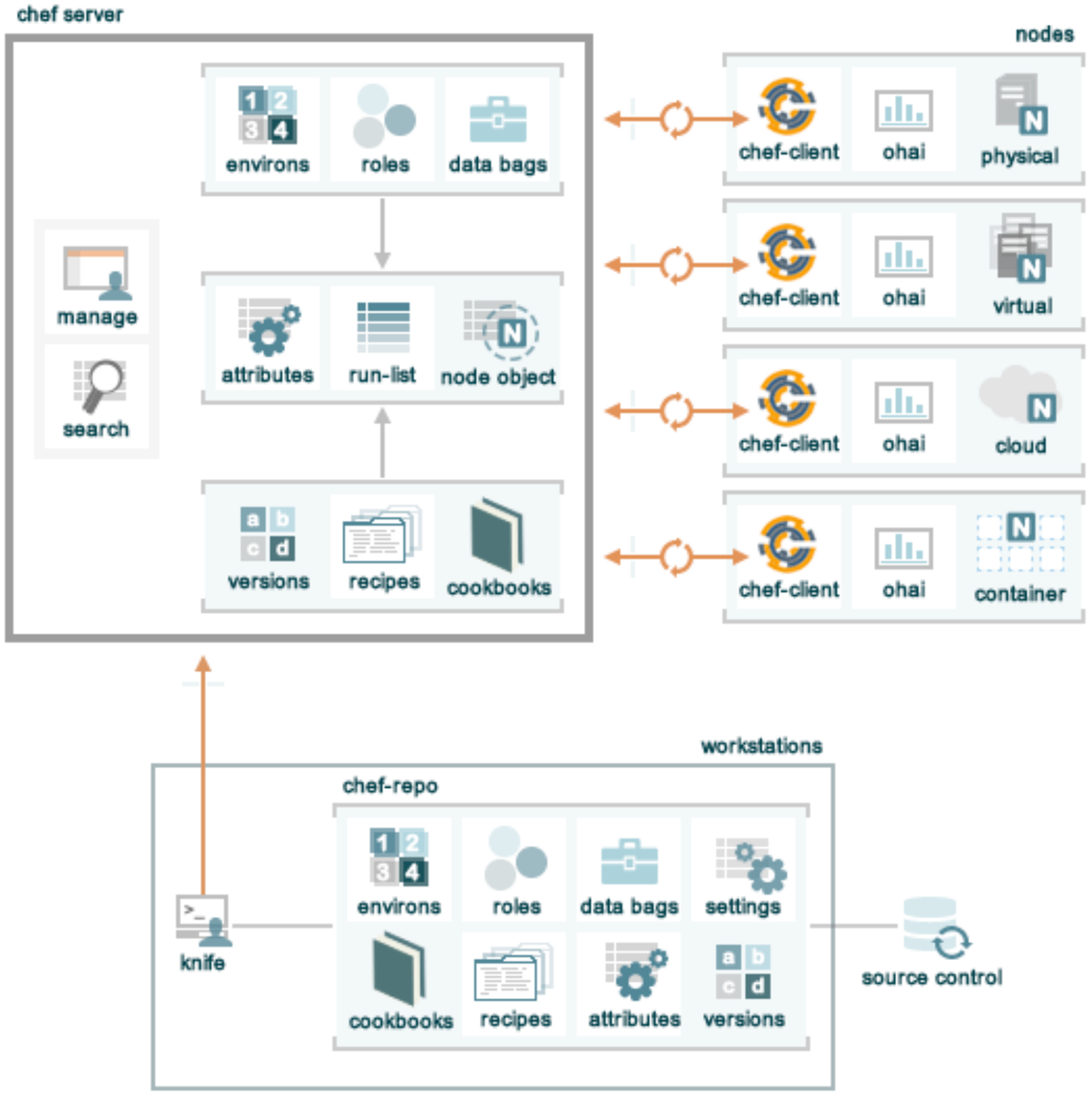
# Policy-based Management

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

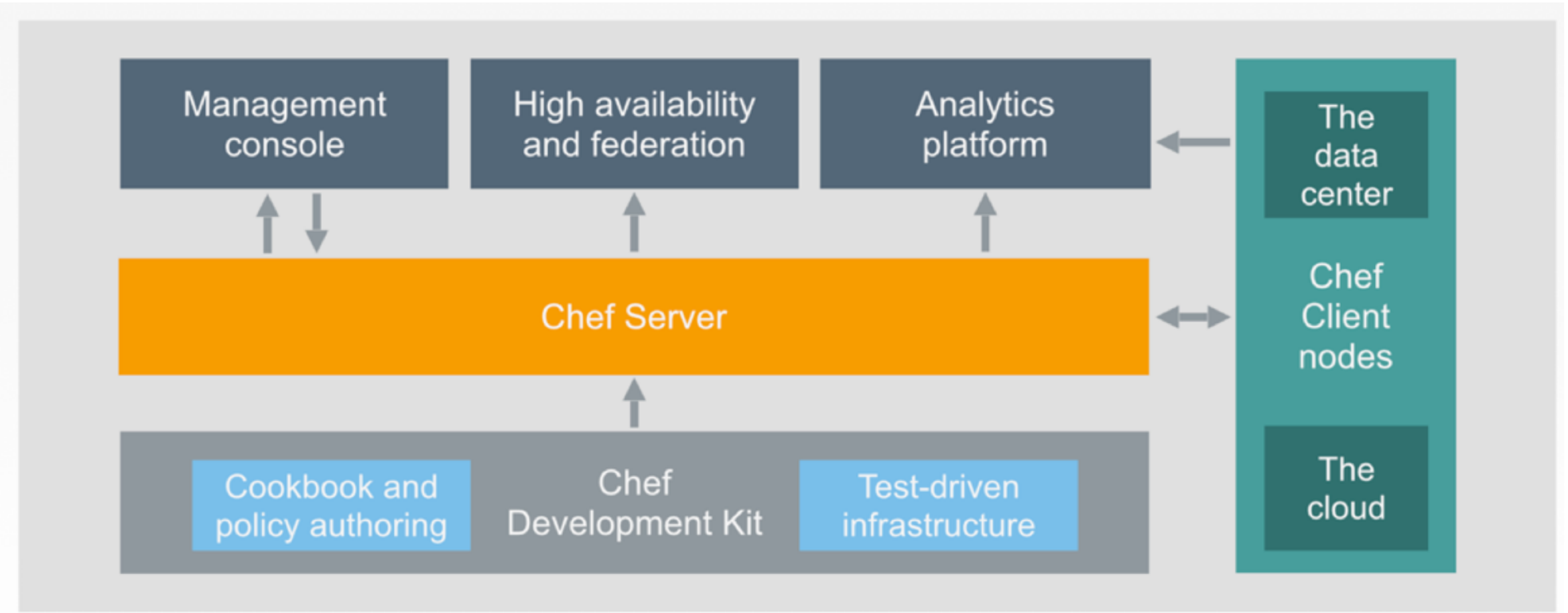
# Policy-based Management

- 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.

# Chef Components



# Chef Server Functions



# Workstation Setup

## Getting Started

# Pre-Configured Workstation

- http://bit.ly/15NJUR0
- chef-dk installed via:
  - curl -L https://www.getchef.com/chef/install.sh | sudo bash -s -- -P chefdk

# Chef Development Kit

- Contains everything you need to start using Chef
  - The new “chef” tool
  - Chef-client distribution
  - Testing frameworks
  - Linting tools
- https://downloads.chef.io/chef-dk/

# SSH to your instance

- Login: chef
- Password: chef

- ssh root@IPADDRESS

# Verify your node

```
$ touch NAME_INTRO; ls | grep INTRO
```

```
jenniferdavis_INTRO
```

# Verify your chef installation

```
$ chef verify
```

```
Running verification for component 'berkshelf'
Running verification for component 'test-kitchen'
Running verification for component 'chef-client'
Running verification for component 'chef-dk'
Running verification for component 'chefspec'
Running verification for component 'rubocop'
Running verification for component 'fauxhai'
Running verification for component 'knife-spork'
Running verification for component 'kitchen-vagrant'
Running verification for component 'package installation'
```

# Resources

## Fundamental Building Blocks

# What is infrastructure?

# Infrastructure Elements

- the individual components or building blocks we have access to automate.

# Infrastructure Elements

- files
- directories
- symlinks
- mounts
- users
- groups
- software packages
- external services
- filesystems

# Resources

- A **resource** represents an element or component of the **system** and the **desired state** of that element.

# Resources - Package

- Package that should be installed

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

# Resources - Service

- Service that should be running and enabled to start on reboot

```
service "iptables" do
  action [ :start, :enable ]
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"
  uid "500"
  gid "500"
  supports :manage_home => true
end
```

# Resources - DSC

- A Desired State Configuration setting that should be applied

```
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 => "Enable LUA”,
    :type => :dword,
    :data => 0
  }]
end
```

# Resources - Package

- Package that should be installed

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

# Resources

- A **resource** represents an element or component of the **system** and the **desired state** of that element.
- https://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

# SSH to your workstation

```
$ ssh chef@IPADDRESS
```

```
chef@54.88.161.76's password:
Last login: Fri Jan 30 06:18:29 2015 from 12.232.194.107
```

# Is $EDITOR installed?

```
$ which vim
```

```
[chef@ip-172-31-35-99 ~]$ which vim
/usr/bin/vim
```

# Is $EDITOR installed?

```
$ which nano
```

```
[chef@ip-172-31-35-99 ~]$ which nano
/usr/bin/which: no nano in (/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 a part of the ChefDK
- A great way to explore resources
- NOT how you will eventually use Chef in production

# What does chef-apply do?

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

```
Usage: chef-apply [RECIPE_FILE] [-e RECIPE_TEXT] [-s]
        --[no-]color                 Use colored output, 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

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

```
[chef@ip-172-31-35-99 ~]$ chef-apply -e "package 'vim'"
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * package[vim] action install (up to date)
```

# Install emacs

```
$ 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 vim

```
$ 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

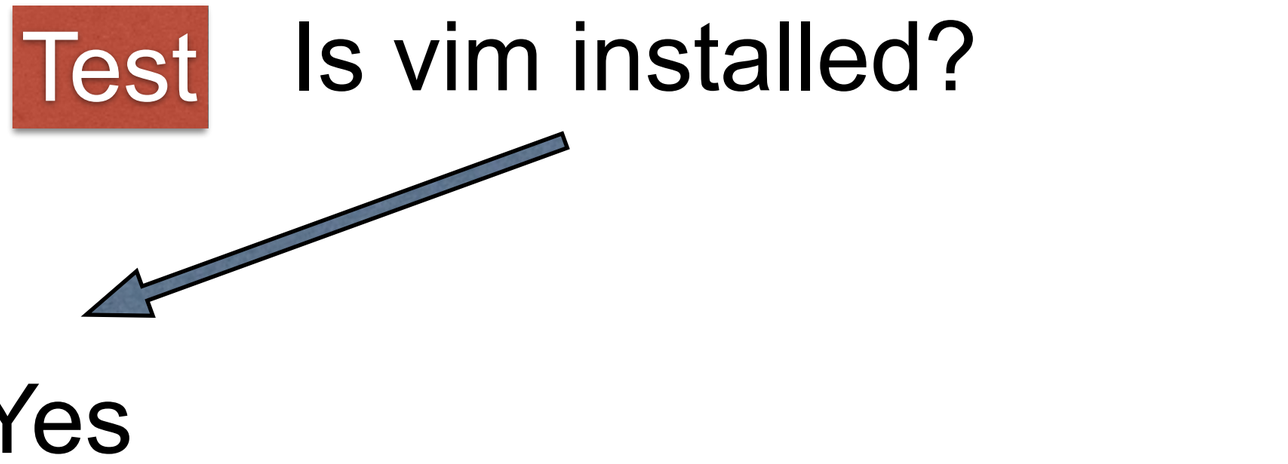
```
package "vim"
```

Test Is vim installed?

# Test and Repair

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

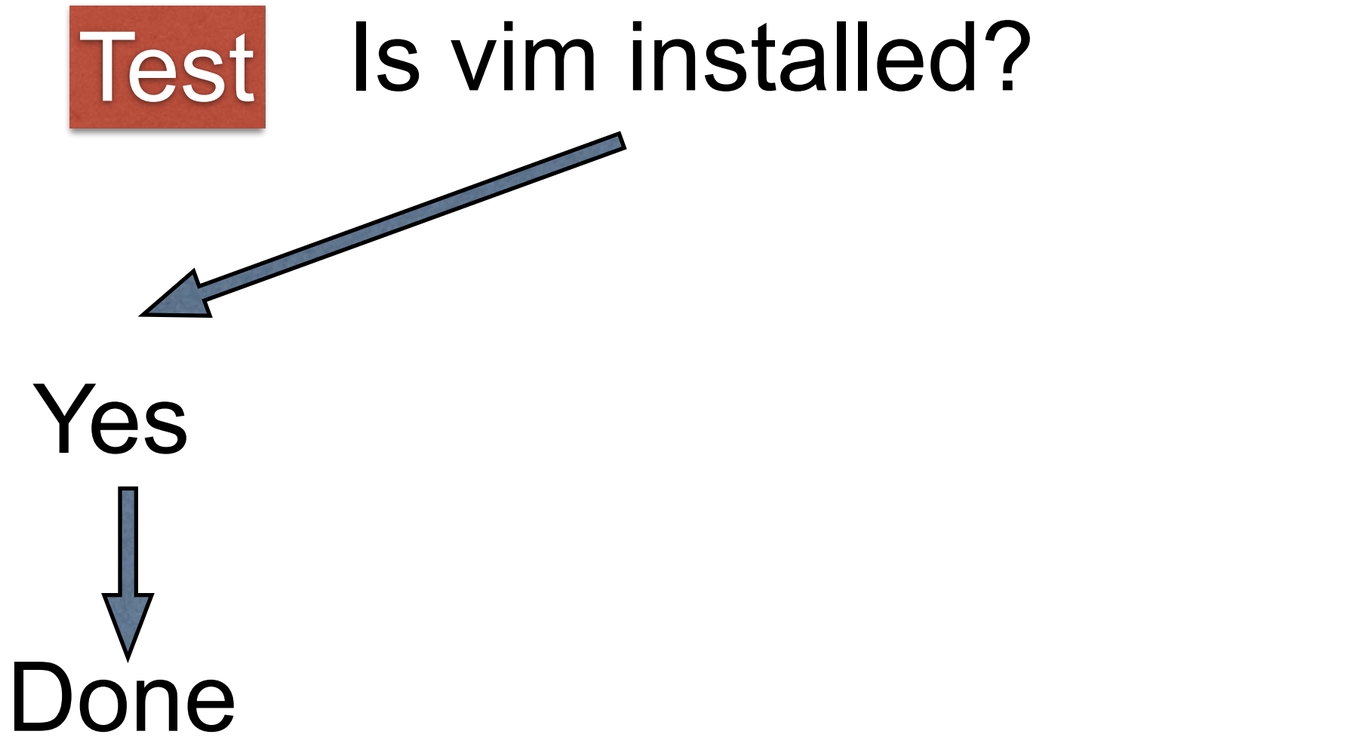
```
package "vim"
```



# Test and Repair

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

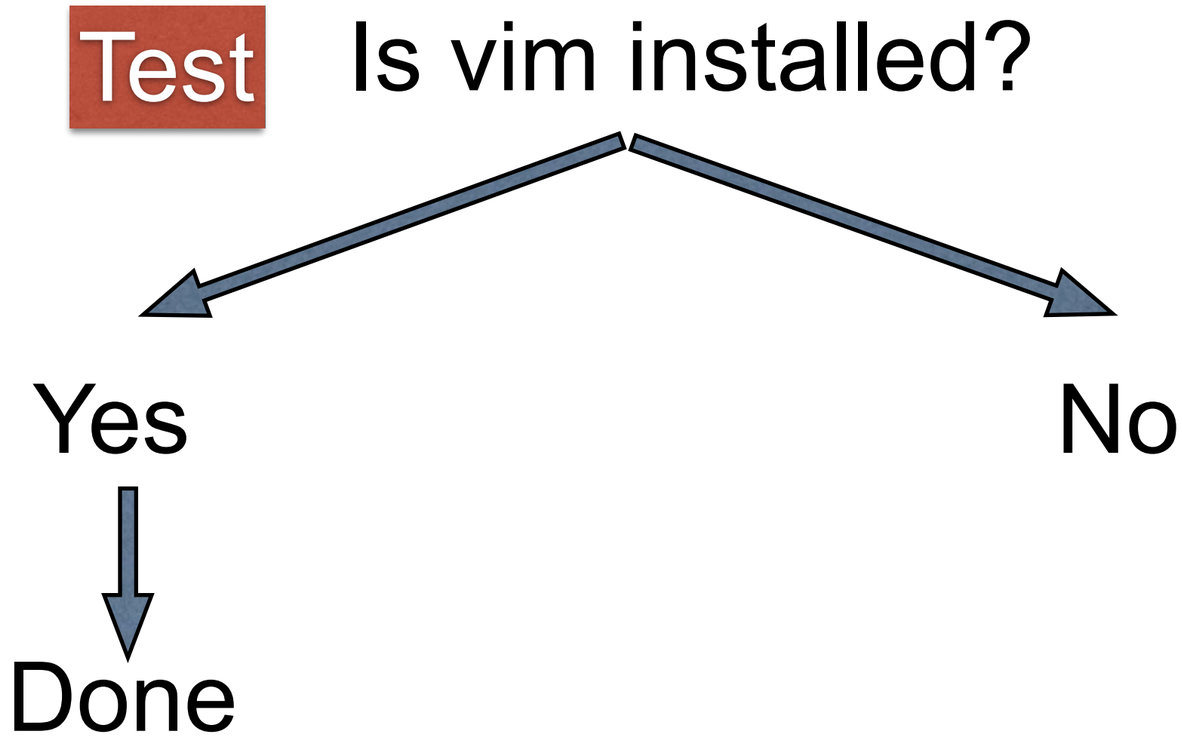
```
package "vim"
```



# Test and Repair

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

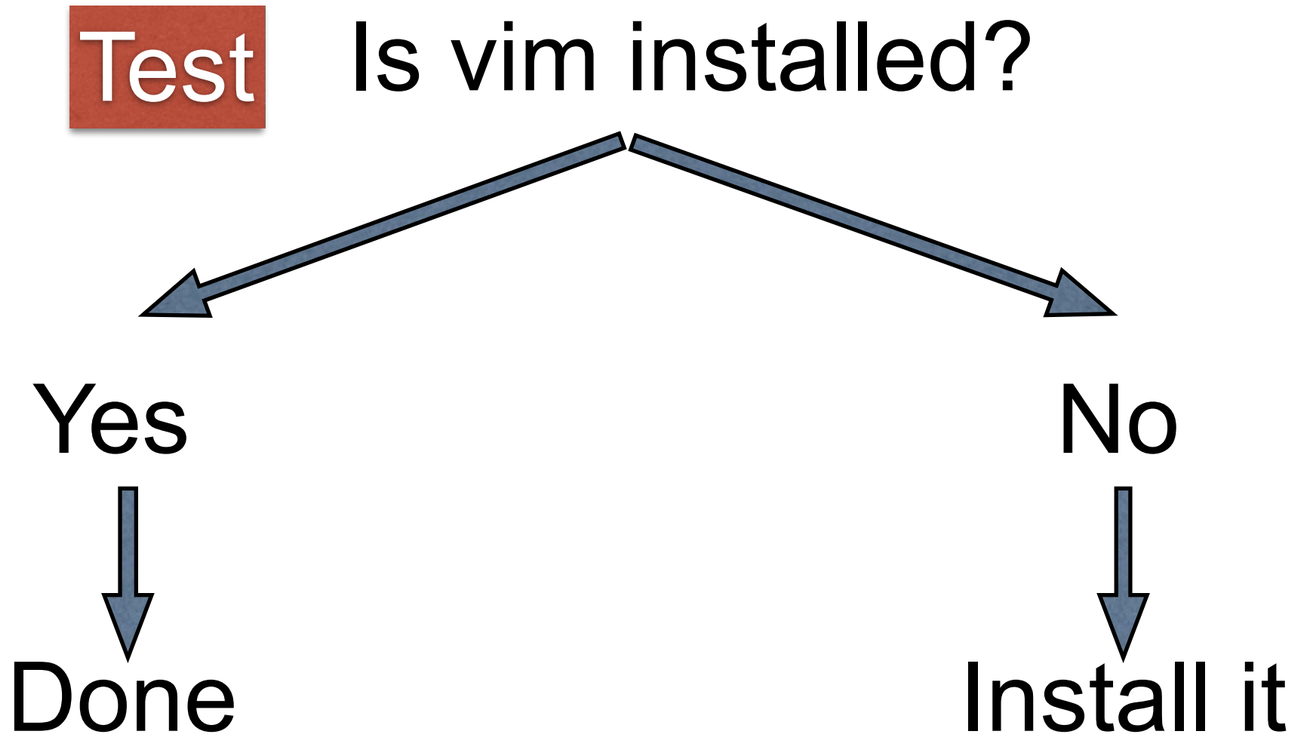
```
package "vim"
```



# Test and Repair

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

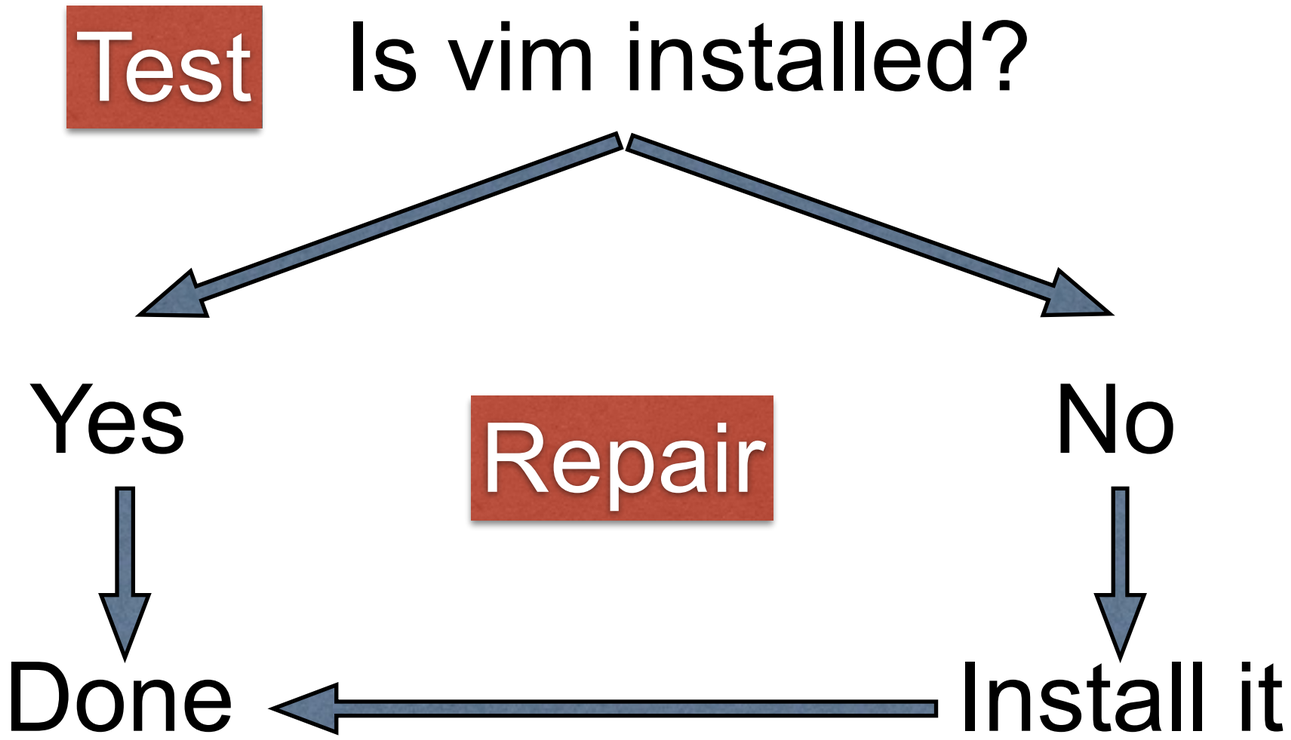
```
package "vim"
```



# Test and Repair

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

```
package "vim"
```



# Resources: Test and Repair

- Resources follow a test & 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

# Recipe

- A **recipe** is an **ordered list** of **resources**.
- Chef DSL is ruby.
- Recipe stored in file with **.rb** extension.

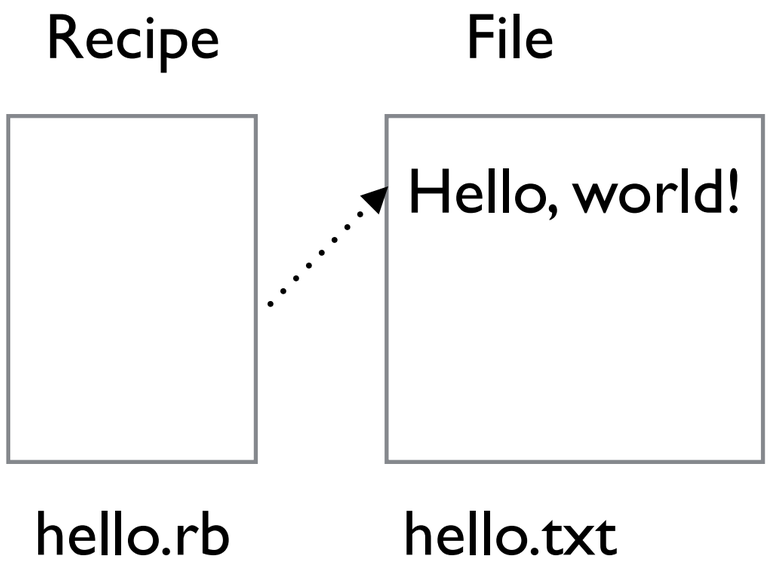
# Hello, World!

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

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

**SAVE FILE!**

# Review




- hello.rb - recipe we created.
- hello.txt - the file we want to manage through file resource.

# 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 dffd60
    --- hello.txt  2015-01-11 08:36:50.195712169 +0000
    +++ /tmp/.hello.txt20150111-5823-4tggr4 2015-01-11 08:36:50.196712169 +0000
    @@ -1 +1,2 @@
    +Hello, World!
    - change mode from '' to '0644'
    - change owner from '' to 'root'
    - 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 type
- Have a name

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

# Chef Resources

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

```
file "hello.txt" do


end
```

# Chef Resources

- Have a type
- Have a name
- Include details between keywords **do** and **end**
- Describe the state using keyword `action`

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

# Chef Resources — In Plain English

- The TYPE named NAME should be ACTION'ed
- The file named "hello.txt" should be created

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

# Chef Resources

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

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

# Chef Resources — In Plain English

- The TYPE named NAME should be ACTION'ed with ATTRIBUTES

```
file "hello.txt" do
  action :create
  content "Hello, World!"
  mode "0644"
  owner "root"
  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 be root user and root group

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

# Re-apply hello.rb

```
$ 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 & 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?

# Let’s look at the File Resource

- http://docs.chef.io/chef/resources.html#file

# Lab 2 - Hello World!

- **Problem:** Clean up “Hello World” file.

- **Success Criteria:** Deletion of hello.txt file.

# Lab 2 - Hello World!

- We will create a new recipe called goodbye.
  - Where will this be saved?
  - What type of resource will we use?
  - What is the name of the resource?
  - What action?

# Goodbye Hello, World!

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

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

**SAVE FILE!**

# Apply goodbye.rb

```
$ chef-apply goodbye.rb
```

# 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

- **Problem**: We want users logging into the system to see “Property of COMPANY NAME” upon login.
- **Success Criteria:** An updated /etc/motd with “Property of COMPANY NAME” that is displayed on login.

# Lab 3 - Manage a file

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

# Summary

- What questions can I answer for you?

# Describing Policies

## Recipes & Cookbooks

# Resources > Recipes > Cookbooks

- A resource is a piece of the system and its desired state
- A recipe is a collection of resources
- A cookbook is a collection of recipes
  - Cookbooks can also contain ingredients
  - A cookbook is a “package” of policy information

# Recipe - A collection of resources

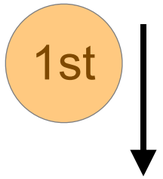
```
package "haproxy" do
  action :install
end

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
```

# Order Matters

- Resources are executed in order



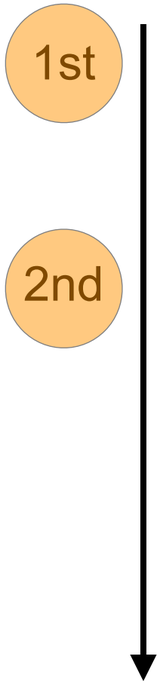

```
package "haproxy" do
  action :install
end

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
```

# Order Matters

- Resources are executed in order




```
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
```

# Order Matters

- Resources are executed in order


```
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
- Distribution unit
- Versioned
- Modular and 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 - It should be version 6 and listening on 8080

# Abstracting Data from Policy

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

# Lab 4 - Manage Data & Policy Separately

- **Problem:** The policy for managing file `/etc/motd` currently has state intermingled with content

- **Success Criteria:** We have a policy that manages the state and content of `/etc/motd` 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 independently of policy changes

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

# Cookbooks contain ingredients

- Cookbooks allow you to package up separate components in one distributable unit
- Cookbooks contain recipes (policy)
- Cookbooks can also contain files to distribute (content)

# Lab 4 - Manage Data & Policy Separately

- Required steps:
  - Create a repository to store our cookbooks
  - Create a cookbook
  - Separate policy from data

# The chef-repo

- Chef cookbooks should be stored in a version control system
- You have options for how best to manage your repository of chef code
- We will create a single chef-repo for all of our cookbooks

# The “chef” command

- 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 Chef 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
  policyfile   Generate a Policyfile for use with the install/push commands
(experimental)
```

# 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'
    -a, --generator-arg KEY=VALUE    Use to set arbitrary attribute KEY to VALUE in the
code_generator cookbook
    -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,      Use GENERATOR_COOKBOOK_PATH for the code_generator cookbook
        --generator-cookbook
```

# Start from home directory

```
$ cd ~
```

# Make a chef repo

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

```
Compiling Cookbooks...
Recipe: code_generator::repo
  * directory[/root/chef-repo] action create
    - create new directory /root/chef-repo
    - restore selinux security context
  * template[/root/chef-repo/LICENSE] action create
    - create new file /root/chef-repo/LICENSE
    - update content in file /root/chef-repo/LICENSE from none to aed48c
    (diff output suppressed by config)
    - restore selinux security context
  * cookbook_file[/root/chef-repo/README.md] action create
    - create new file /root/chef-repo/README.md
    - update content in file /root/chef-repo/README.md from none to 767ead
    (diff output suppressed by config)
    - restore selinux security context
  * cookbook_file[/root/chef-repo/Rakefile] action create
    - create new file /root/chef-repo/Rakefile
    - update content in file /root/chef-repo/Rakefile from none to 108932
    (diff output suppressed by config)
    - restore selinux security context
```

# Examine chef-repo

```
$ tree
```

```
└── chef-repo
    ├── certificates
    │   └── README.md
    ├── chefignore
    ├── config
    │   └── rake.rb
    ├── cookbooks
    │   └── README.md
    ├── data_bags
    │   └── README.md
    ├── environments
    │   └── README.md
    ├── LICENSE
    ├── Rakefile
    ├── README.md
    └── roles
        └── README.md
```

# Lab 4 - Manage Data & Policy Separately

- Required steps:

✔ Create a repository to store our cookbooks

  - Create a cookbook
  - Separate policy from data

# How do we generate a 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'
    -a, --generator-arg KEY=VALUE    Use to set arbitrary attribute KEY to VALUE in the
code_generator cookbook
    -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
```

# Generate a “motd” cookbook

```
$ chef generate cookbook cookbooks/motd
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/tmp/chef-repo/cookbooks/motd] action create
    - create new directory /tmp/chef-repo/cookbooks/motd
  * template[/tmp/chef-repo/cookbooks/motd/metadata.rb] action create_if_missing
    - create new file /tmp/chef-repo/cookbooks/motd/metadata.rb
    - update content in file /tmp/chef-repo/cookbooks/motd/metadata.rb from none to 811b2e
    (diff output suppressed by config)
  * template[/tmp/chef-repo/cookbooks/motd/README.md] action create_if_missing
    - create new file /tmp/chef-repo/cookbooks/motd/README.md
    - update content in file /tmp/chef-repo/cookbooks/motd/README.md from none to 918996
```

# Examine motd cookbook

```
$ tree
```

```
├── cookbooks
|   ├── motd
|   |   ├── Berksfile
|   |   ├── chefignore
|   |   ├── metadata.rb
|   |   ├── README.md
```

# Lab 4 - Manage Data & Policy Separately

- Required steps:

✔ Create a repository to store our cookbooks

✔ Create a cookbook

  - Separate policy from data

# Copy your motd.rb

```
$ cat ~/motd.rb >> ~/chef-repo/cookbooks/motd/recipes/default.rb
```

# Update the recipe

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

```
#
# Cookbook Name:: motd
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
file "/etc/motd" do
  content "Property of COMPANY NAME\n”
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

# Which 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
├── chefignore
├── files
│   └── default
│       └── motd
├── metadata.rb
├── README.md
└── 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 a webserver

```
remote_file "/etc/motd" do
  source "http://bit.ly/motd"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

# template

- A template in the cookbook is used to render the content of the file

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

# template

- A template is a mix of static and dynamic content

**motd/templates/default/motd.erb**

```
Property of <%= @company_name %>
```

# Which 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 - static file within the cookbook
- file - content managed inline
- remote_file - static file obtained from URL
- template - dynamic content rendered from ERB

# template

- An ERB template stored as part of our cookbook

# Update the recipe

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

```
#
# Cookbook Name:: motd
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
template "/etc/motd" do
  source "motd.erb"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

# How do we generate a 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'
    -a, --generator-arg KEY=VALUE    Use to set arbitrary attribute KEY to VALUE in the
code_generator cookbook
    -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
```

# Create the ERB template

```
$ cd ~/chef-repo
$ chef generate template cookbooks/motd motd -s /etc/motd
```

```
Compiling Cookbooks...
Recipe: code_generator::template
  * directory[cookbooks/motd/templates/default] action create
    - create new directory cookbooks/motd/templates/default
  * file[cookbooks/motd/templates/default/motd.erb] action create
    - create new file cookbooks/motd/templates/default/motd.erb
    - update content in file cookbooks/motd/templates/default/motd.erb from none to 166ed9
    (diff output suppressed by config)
```

# Examine cookbook

```
$ tree
```

```
|   |   ├── templates
|   |   |   └── default
|   |   |       └── motd.erb
```

# Check the template

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

```
Property of COMPANY NAME
```

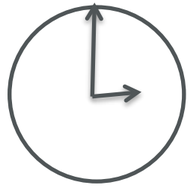
# chef-apply

- 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

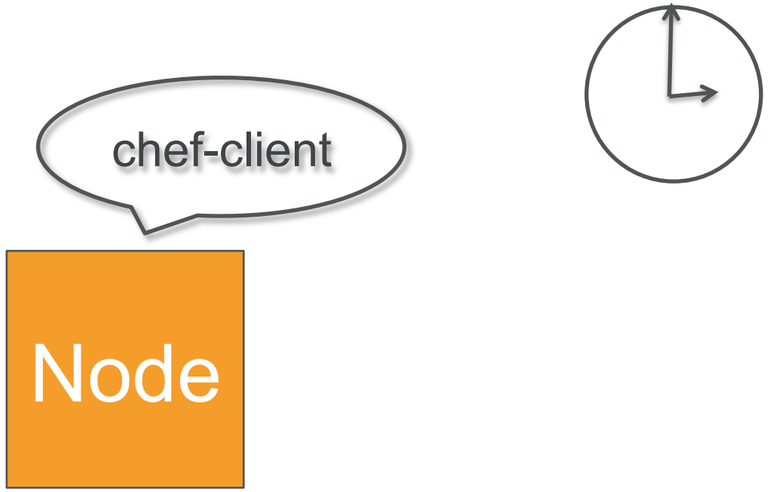
- chef-client is an executable
  - performs all actions required to bring a node into the desired state
  - typically runs on a regular basis
    - daemon
    - cron
    - Windows service
- Included with ChefDK

# chef-client applying policies

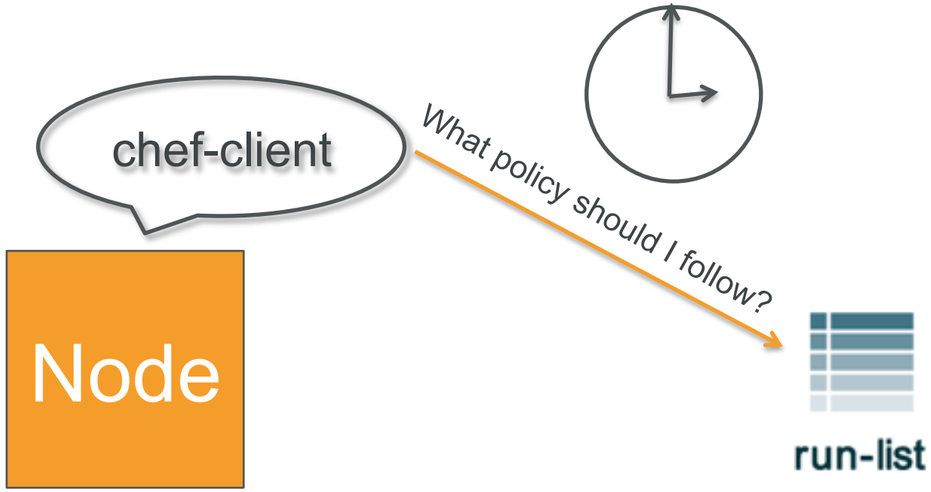


Node

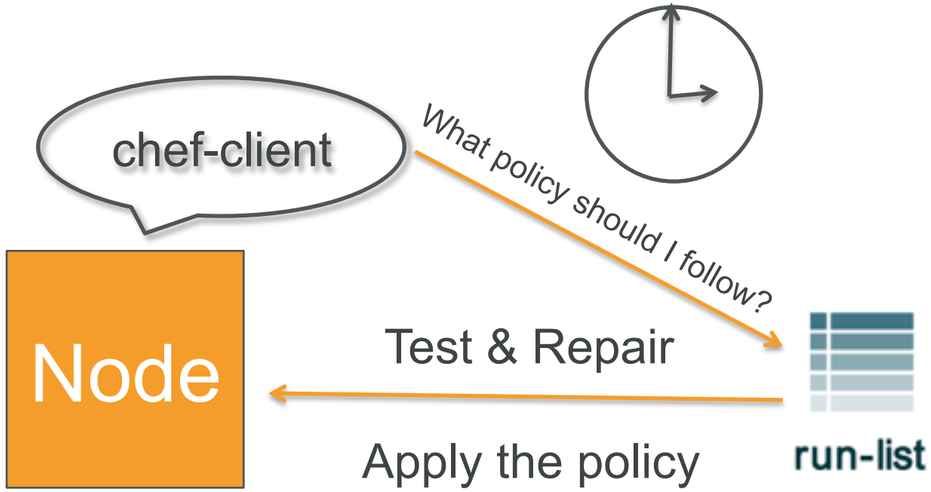
# chef-client applying policies



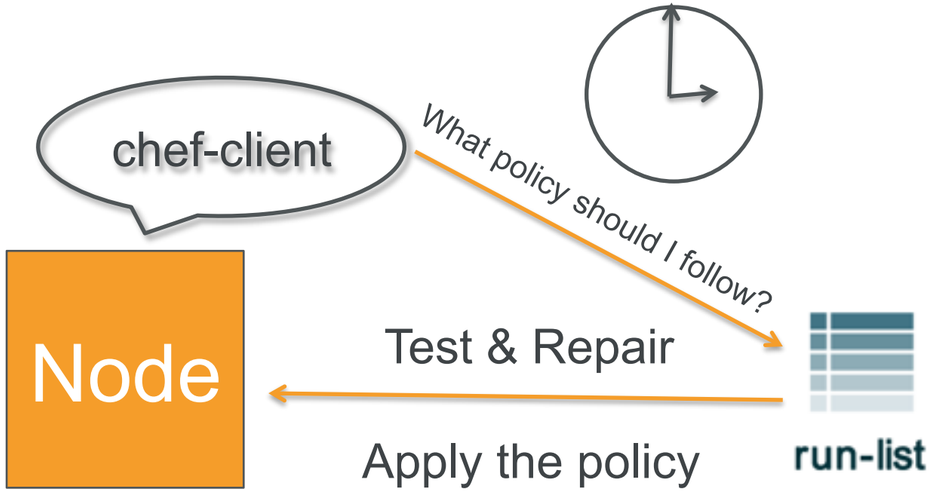
# chef-client applying policies

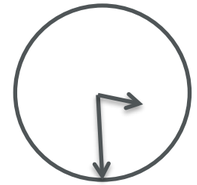


# chef-client applying policies



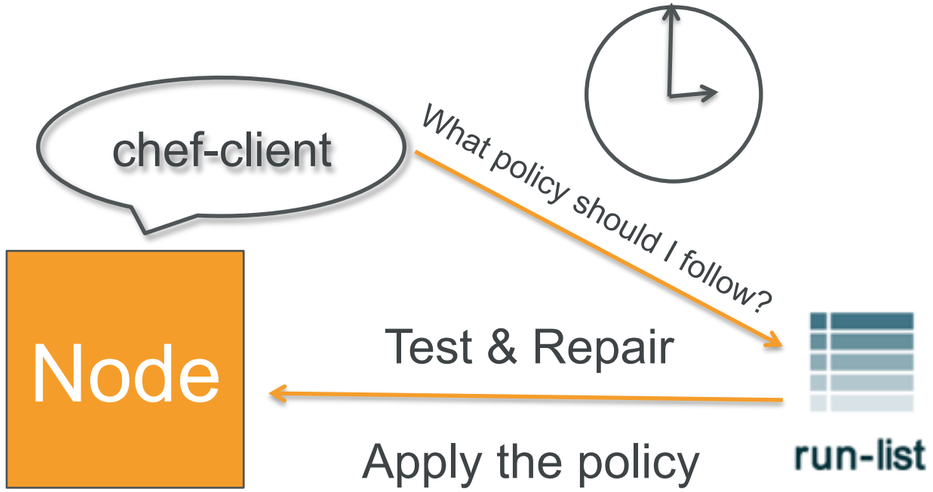
# chef-client applying policies



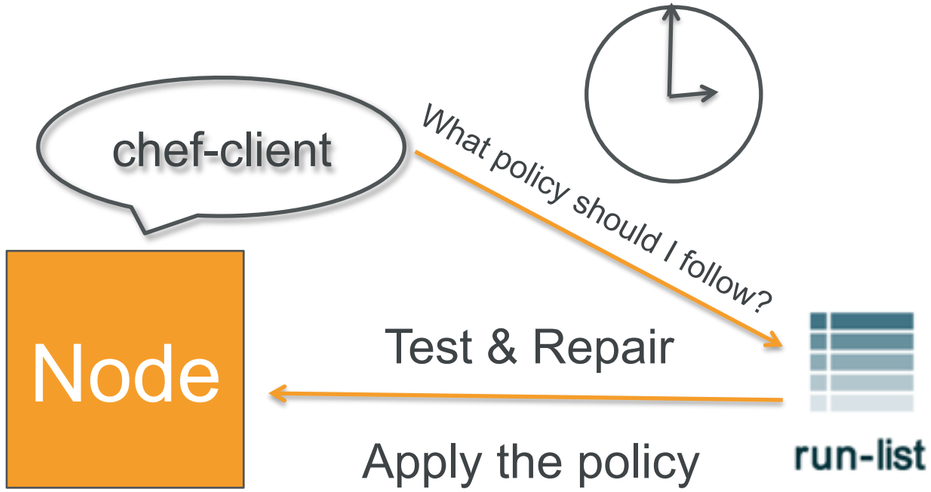



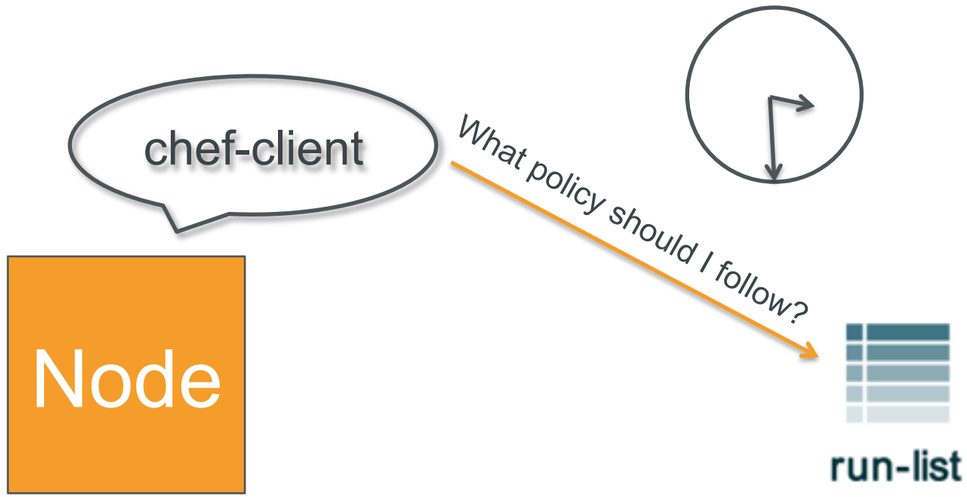
Node

# chef-client applying policies

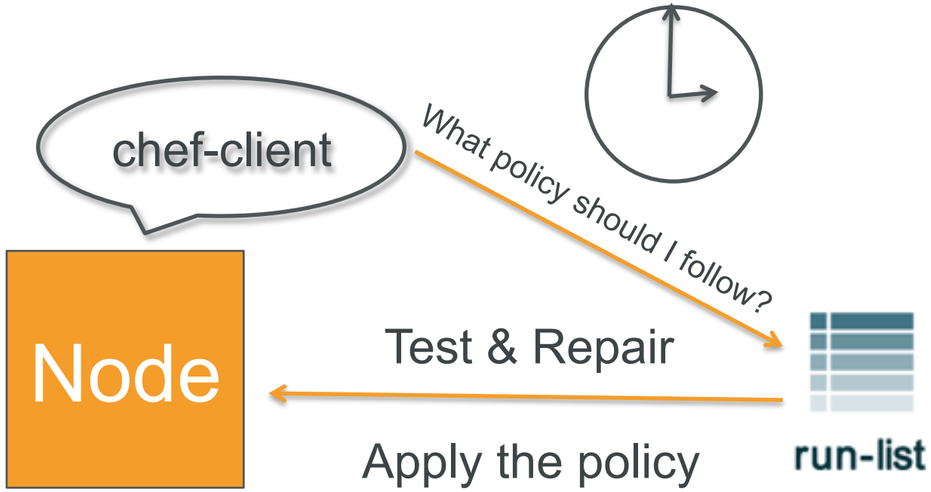


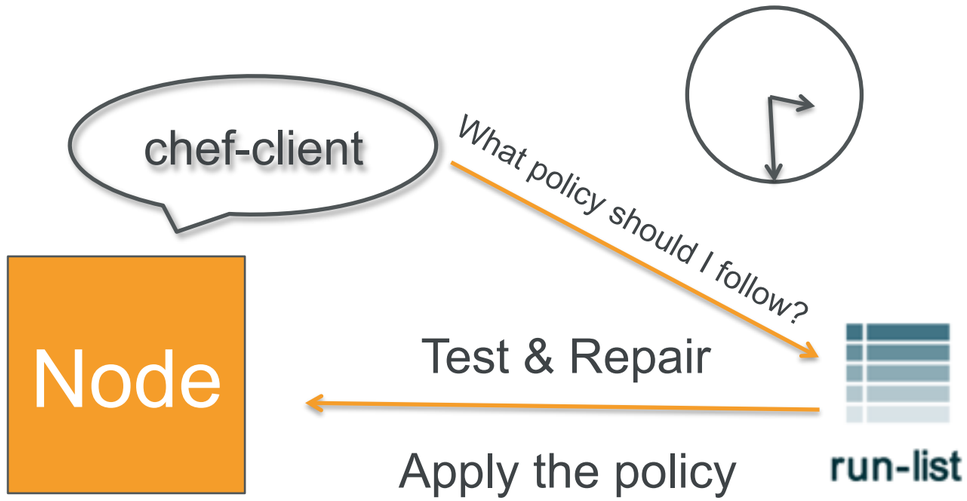

# chef-client applying policies



# chef-client applying policies



# 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 non-privileged user

# Apply our recipe using chef-client

```
$ cd ~/chef-repo
$ chef-client --local-mode -r "recipe[motd]"
```

```
2015-01-11T11:05:00+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 0 resources

Running handlers:
Running handlers complete
Chef Client finished, 0/0 resources updated in 3.99531239 seconds
```

# Pop Quiz

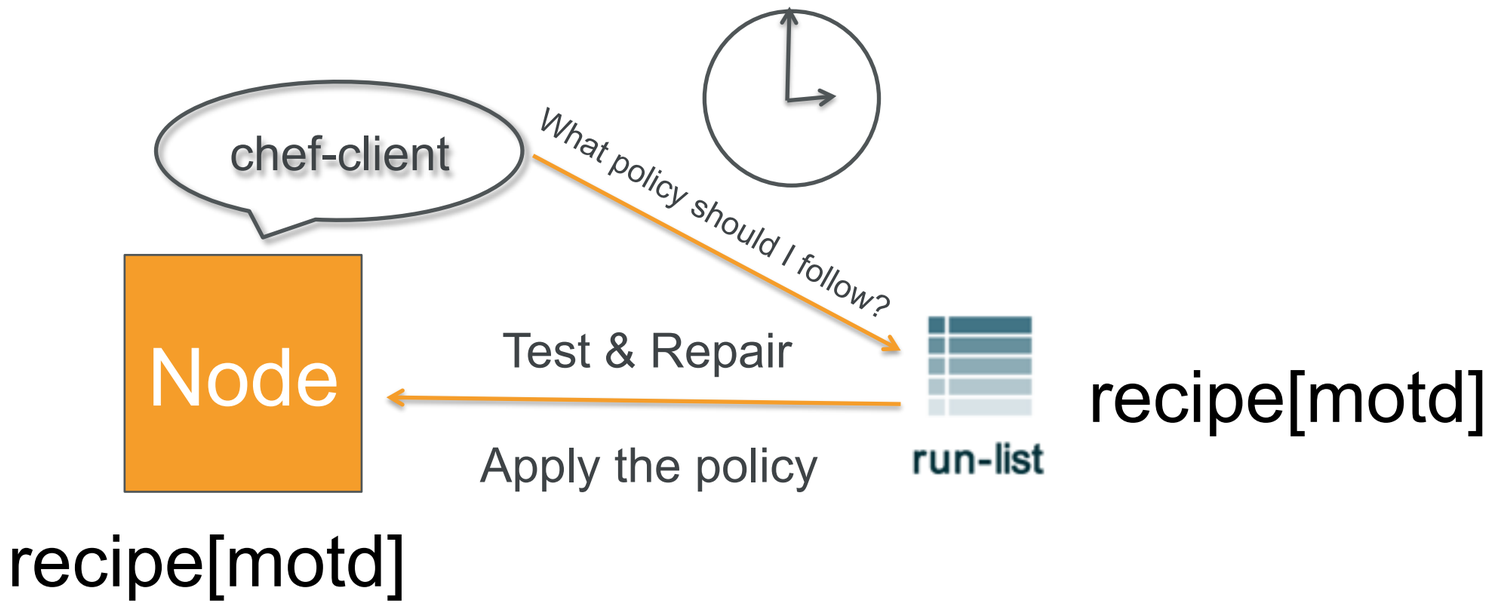
- Why did chef-client update zero resources?

# Pop Quiz

- Why did chef-client update zero resources?

- Resources test & repair
- Remember: we generated the template from an existing file on the filesystem
- The content of /etc/motd is already the content we wanted

# chef-client applying policies



# Lab 4 - Manage Data & Policy Separately

- Required steps:

✓ Create a repository to store our cookbooks

✓ Create a cookbook

✓ Separate policy from data

# Separating data from policy

- Storing file content directly in a recipe is not typically the best choice
- We can manage that content separately using a different resource
  - cookbook_file
  - remote_file
  - template

# Template resources

- An ERB template that generates files based on a mix of static content along with variables and logic to determine dynamic content
- If in doubt, start with a template

# What if…?

- The contents of the motd file should be pulled from a file in an S3 bucket?
- The motd file should have variable content based on different owners using the same cookbook?

# Summary

- What components can be in a cookbook?
- What program should we use to apply code from cookbooks?
- What resource should we look to first when managing a file?

# Summary

- What questions can I answer for you?

# Creating a simple web server

## An Example Policy

# Lab 5 - Install an Apache webserver

- **The Problem:** We need a web server configured to serve up a custom home page

- **Success Criteria:** We can see the custom homepage in a web browser

# Required steps

- Install Apache
- Write out the home page
- Start the Apache service
- Make sure the Apache service starts when the machine boots
- Stop and disable the iptables service

- *Please note we're teaching Chef primitives, not web server management! This is probably not the Apache HTTP server configuration you would use in production.*

# Create a new webserver cookbook

```
$ cd ~/chef-repo
$ chef generate cookbook webserver
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/root/chef-repo/webserver] action create
    - create new directory /root/chef-repo/webserver
    - restore selinux security context
  * template[/root/chef-repo/webserver/metadata.rb] action create_if_missing
    - create new file /root/chef-repo/webserver/metadata.rb
    - update content in file /root/chef-repo/webserver/metadata.rb from none to 4c0e2d
    (diff output suppressed by config)
    - restore selinux security context
  * template[/root/chef-repo/webserver/README.md] action create_if_missing
    - create new file /root/chef-repo/webserver/README.md
    - update content in file /root/chef-repo/webserver/README.md from none to 5c3d3a
    (diff output suppressed by config)
    - restore selinux security context
  * cookbook_file[/root/chef-repo/webserver/chefignore] action create
    - create new file /root/chef-repo/webserver/chefignore
    - update content in file /root/chef-repo/webserver/chefignore from none to 9727b1
    (diff output suppressed by config)
    - restore selinux security context
```

# Exercise: Edit the default recipe

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

```
#
# Cookbook Name:: webserver
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
```

# Which 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

- Install Apache
- Write out the home page
- Start the Apache service
- Make sure the Apache service starts when the machine boots
- Stop and disable the iptables service

# Which 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

- Install Apache
- Write out the home page
- Start the Apache service
- Make sure the Apache service starts when the machine boots
- Stop and disable the iptables service

# 3 most common resources - PST

- Install software (**package**)
- Manipulate files (**template**)
- Manipulate services (**service**)

- You will see this pattern very often

# Lab 5 - Install an Apache webserver

- Install Apache - the package is named “httpd”
- Write out the home page - /var/www/html/index.html
- Manage the apache service - named “httpd”
  - Make sure the service is started
  - Make sure the service is enabled to start on boot
- Manage the iptables service - named “iptables”
  - Make sure the service is stopped
  - Make sure the service is disabled to start on boot

# Checkpoint - Lab 5

**OPEN IN EDITOR:** ~/chef-repo/cookbooks/webserver/recipes/default.rb

```
#
# Cookbook Name:: webserver
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.

package "httpd"

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

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

service "iptables" do
  action [ :disable, :stop ]
end
```

# Create a template for index.html

```
$ cd ~/chef-repo
$ chef generate template cookbooks/webserver index.html
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/root/chef-repo/webserver] action create
    - create new directory /root/chef-repo/webserver
    - restore selinux security context
  * template[/root/chef-repo/webserver/metadata.rb] action create_if_missing
    - create new file /root/chef-repo/webserver/metadata.rb
    - update content in file /root/chef-repo/webserver/metadata.rb from none to 4c0e2d
    (diff output suppressed by config)
    - restore selinux security context
  * template[/root/chef-repo/webserver/README.md] action create_if_missing
<html>
    - create new file /root/chef-repo/webserver/README.md
    - update content in file /root/chef-repo/webserver/README.md from none to 5c3d3a
    (diff output suppressed by config)
    - restore selinux security context
  * cookbook_file[/root/chef-repo/webserver/chefignore] action create
    - create new file /root/chef-repo/webserver/chefignore
    - update content in file /root/chef-repo/webserver/chefignore from none to 9727b1
    (diff output suppressed by config)
    - restore selinux security context
```

# Checkpoint - Lab 5

**OPEN IN EDITOR:** `~/chef-repo/cookbooks/webserver/template/default/index.html.erb`

```
<html>
 <body>
  <h1>Hello, World!</h1>
 </body>
</html>
```

# Question: does it work???

- This is a great opportunity for test-driven development
- TDD is a way to automatically verify that your automation is producing expected results
- Test against cheap throwaway infrastructure
  - e.g. inside containers!

# Apply our recipe using chef-client

```
$ cd ~/chef-repo
$ chef-client --local-mode -r "recipe[webserver]"
```

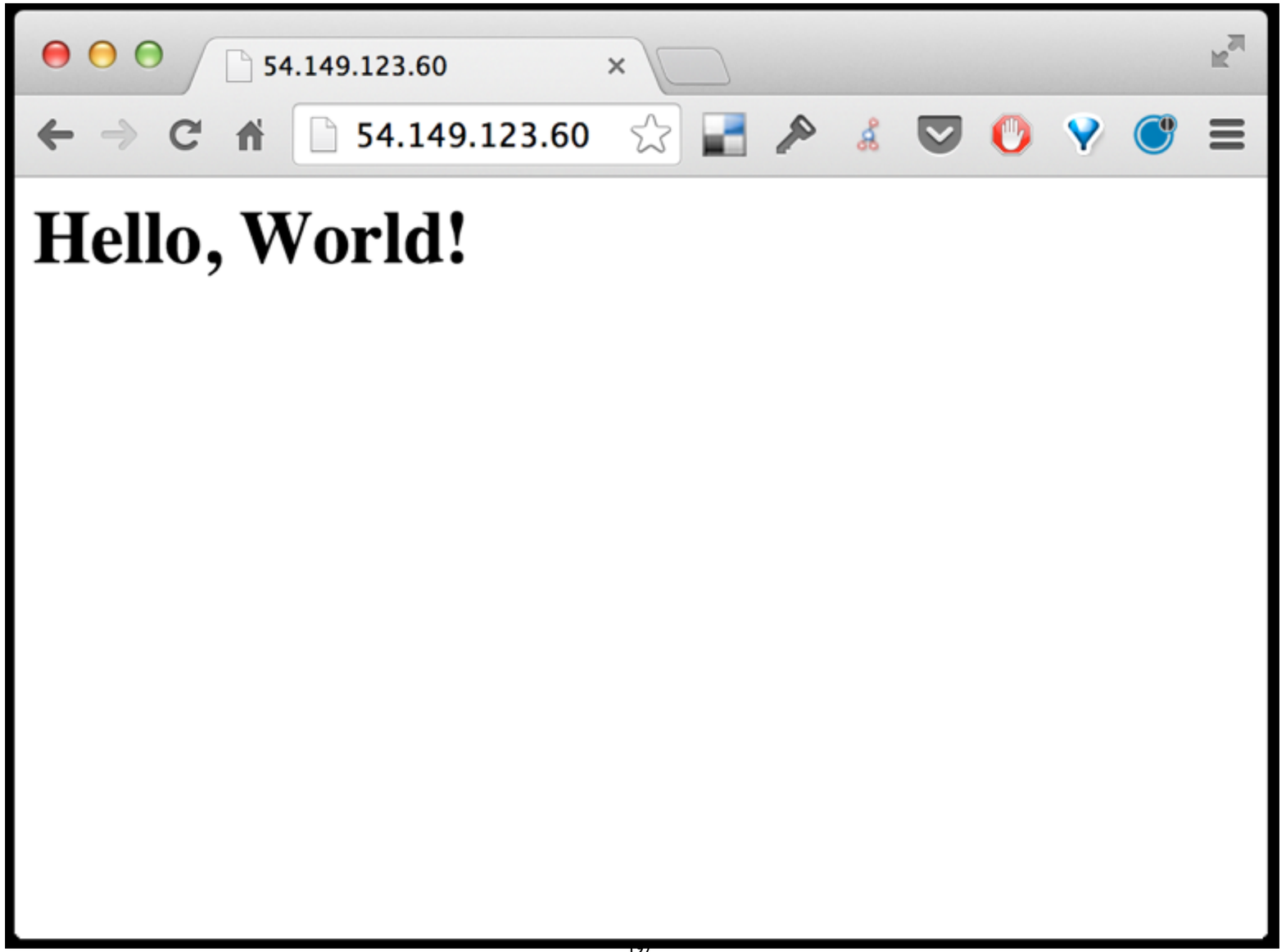
```
Recipe: webserver::default
  * package[httpd] action install
    - install version 2.2.15-39.el6.centos of package httpd
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
    - start service service[httpd]
  * template[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to e043d5
    --- /var/www/html/index.html     2015-01-11 12:05:11.281712169 +0000
    +++ /tmp/chef-rendered-template20150111-7749-3tgp7p    2015-01-11 12:05:11.283712169 +0000
    @@ -1 +1,6 @@
   +  <h1>Hello, World!</h1>
   - restore selinux security context
  * service[iptables] action disable
    - disable service service[iptables]
  * service[iptables] action stop
    - stop service service[iptables]
Running handlers:
Running handlers complete
Chef Client finished, 6/6 resources updated in 13.854983543 seconds
```

# Display your custom homepage

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

```
<html>
 <body>
  <h1>Hello, World!</h1>
 </body>
</html>
```

# Try it in a web browser



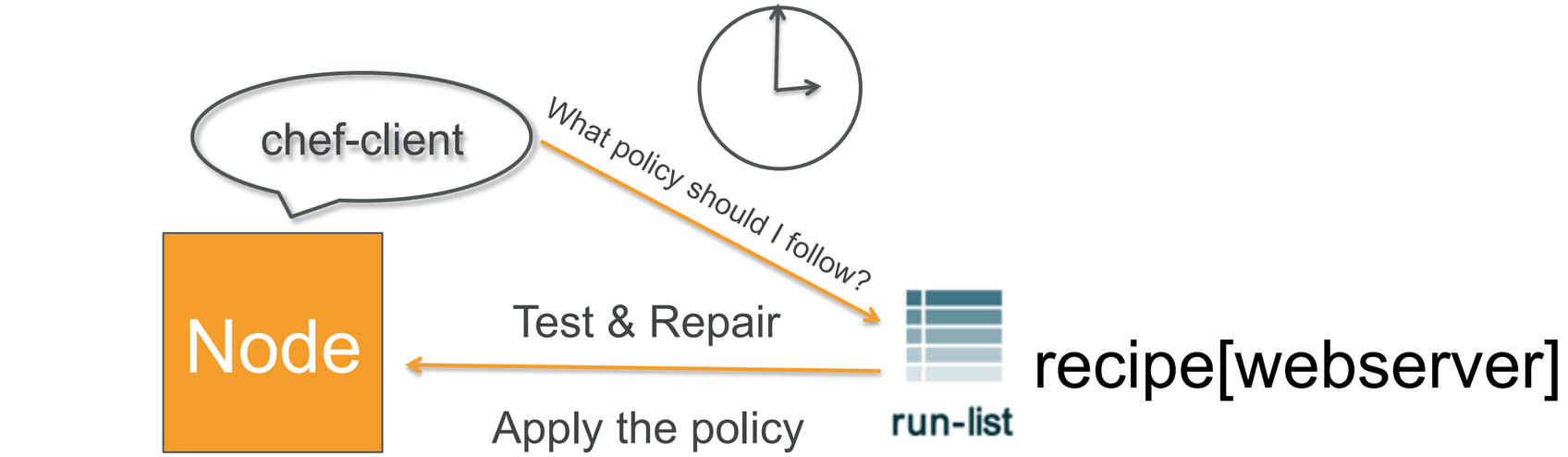
# Run chef-client again

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

```
Starting Chef Client, version 11.18.0.rc.1
resolving cookbooks for run list: ["webserver"]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
  * package[httpd] action install (up to date)
  * service[httpd] action enable (up to date)
  * service[httpd] action start (up to date)
  * template[/var/www/html/index.html] action create (up to date)
  * service[iptables] action disable (up to date)
  * service[iptables] action stop (up to date)

Running handlers:
Running handlers complete
Chef Client finished, 0/6 resources updated in 7.046113816 seconds
```

# chef-client applying policies




Everything is up to date!

# Test & Repair

- Gives you the ability to roll out new policies
- Helps avoid unintended changes being made to the system
- Prevent manual change
- Prevent natural process of configuration drift
- Encourage a culture of capturing all configuration

# Lab 6 - Display custom node data

- **The Problem:** We want our home page to display the hostname of our node

- **Success Criteria:** We can see the hostname of our node in a web browser

# How do we get our node name?

- We could hard code the hostname of our node
- That's a bummer because it doesn't scale
- chef-client keeps a record of every node it manages
- We should be able to get the name of any node managed by chef-client dynamically
- We can use the **knife** tool to view this chef-client data

# knife

- is an executable program used from the command line to communicate with a Chef server
- can be used in local mode to mock Chef server functionality
- in local-mode is a great way to explore some Chef server functionality without committing to running a Chef server just yet
- is included as a part of the ChefDK

# List nodes we know about

```
$ knife node list --local-mode
```

```
WARNING: No knife configuration file found
ip-172-31-25-147.us-west-2.compute.internal
```

# Node data from chef-client

- chef-client saves a node object at the end of a run
- In server mode, the Chef Server can be queried for information about the node
- In local mode, the node object is saved to the local filesystem
- A lot of this information comes from Ohai - we'll see Ohai later.....

# chef-client on our local vm

- We have a node record for our workstation because we ran chef-client locally
- Again, this sort of recursive management is not the way you'd do it in Production

# Each node must have a unique name

- Every node Chef knows about must have a unique name within an organization
- Chef defaults to the *Fully Qualified Domain Name* of the server, i.e. in the format `server.domain.com`
- In EC2, this typically defaults to something that looks like `ip-x-x-x-x.region.compute.internal`

# Show node details with knife

```
$ knife node show ip-172-31-25-147.us-west-2.compute.internal --local-mode
```

```
WARNING: No knife configuration file found
Node Name:   ip-172-31-25-147.us-west-2.compute.internal
Environment: _default
FQDN:        ip-172-31-25-147.us-west-2.compute.internal
IP:          172.31.25.147
Run List:    recipe[webserver]
Roles:
Recipes:     webserver, webserver::default
Platform:    centos 6.5
Tags:
```

# Pro-Tip: use $(hostname -f)

- We can get our node's *Fully Qualified Domain Name* in the right format using the hostname command
- hostname -f  returns the full FQDN

# Show node details with knife

```
$ knife node show $(hostname -f) --local-mode
```

```
WARNING: No knife configuration file found
Node Name:   ip-172-31-25-147.us-west-2.compute.internal
Environment: _default
FQDN:        ip-172-31-25-147.us-west-2.compute.internal
IP:          172.31.25.147
Run List:    recipe[webserver]
Roles:
Recipes:     webserver, webserver::default
Platform:    centos 6.5
Tags:
```

# What is the Node object

- Nodes are made up of Attributes
  - Many are discovered **automatically** (platform, ip address, number of CPUs)
  - Many other objects in Chef can also add Node attributes (Cookbooks, Roles and Environments, Recipes, Attribute Files)

# Where is the Node object

- In server mode, node objects are stored and indexed on the Chef Server
- In local mode, node objects are stored on the local filesystem

- Node objects are stored in JSON

# List existing node objects

```
$ ls ~/chef-repo/nodes
```

```
ip-172-31-25-147.us-west-2.compute.internal.json
```

# Peek inside the node object

```
$ less ~/chef-repo/nodes/$(hostname -f).json
```

```
{
  "name": "ip-172-31-7-47.us-west-2.compute.internal",
  "normal": {
    "tags": [

    ]
  },
  "automatic": {
    "keys": {
      "ssh": {
        "host_dsa_public": "AAAAB3NzaC1kc3MAAACBAIKx4i9XbyCQVvt2vStlkgNwLhHm5+kRHN3z
5s8q+zseajCk+OxVoMzWAY4fj/5UOXpmZ2IJuGMF21RouO7HcZorNxOK5Y/1ABOUHec3NdZKBuPJcR14774F
r2UpMFu5ugPlA0w+jgsS4HygHAharE2o5EFFrH9v09Pc0OTYqhpxAAAAFQDHEr5ZDB1Jze5Vc5hokWNKxkrS
XwAAAIAudW/pKR9Dh0G4GqfW5hXdav/4tPQswvWpWorlThvlHTGfGnd8/W09jKFfTT/YMwqtvBLBqXbQApH6
kCMa8SfNXVJwpi6TTusRTV2MJrPSp/lnJ3Ya/Qan/QFvI+axgTmhh9qZNEp9LiUzyOHrjLsoP8K38JX89A1j
5gIFjw4gZgAAAIAWRjt7EuD3S+rdkzqzWiXXy8t8taLMednUqV1cpweWg6nKzcIkoEG5r4Anp7VYatTaG9Ur
cR1XzCFsVgFULKA42r1skz1u8XLwZVhFR57j4A+NqyEiN17LHzPeQ/sfEVrZ67zWqkaxF3EdsqFMH1zhoE0q
IZYGjOwTg6jbgIEiDw==",
```

# Where does this data come from?

- Ohai is a discovery agent included with Chef
- It is executed every single time chef-client is run
- Extensible plugin model allows custom data
- Chef uses some information (e.g. ***platform*** ) to make decisions about how to implement resources

# Package Resource

```
package "git"
```

```
yum install git

apt-get install git

pacman sync git

pkg_add -r git
```

**Providers are determined by node's platform**

# Ohai


```
"languages": {
  "ruby": {

  },
  "perl": {
    "version": "5.14.2",
    "archname": "x86_64-
linux-gnu-thread-multi"
  },
  "python": {
    "version": "2.6.6",
    "builddate": "Jul 10
2013, 22:48:45"
  },
  "perl": {
    "version": "version",
    "archname": "x86_64-
linux-thread-multi"
  },
    "lua": {
      "version": "5.1.4"
},
```

```
"kernel": {
  "name": "Linux",
  "release": "3.2.0-32-virtual",
  "version": "#1 SMP Wed Oct 16
18:37:12 UTC 2013",
  "machine": "x86_64",
  "modules": {
    "isofs": {
      "size": "70066",
      "refcount": "2"
    },
    "des_generic": {
      "size": "16604",
      "refcount": "0"
    }
  },
  "os": "GNU/Linux"
},
"os": "linux",
"os_version":
"2.6.32-358.23.2.el6.x86_64",
"ohai_time": 1389105685.7735305,
```

```
"network": {
  "interfaces": {
    "lo": {
      "mtu": "16436",
      "flags": [
        "LOOPBACK", "UP","LOWER_UP"
      ],
      "encapsulation": "Loopback",
      "addresses": {
        "127.0.0.1": {
          "family": "inet",
          "netmask": "255.0.0.0",
          "scope": "Node"
        },
        "::1": {
          "family": "inet6",
          "scope": "Node"
        }
      },
    },
    "eth0": {
      "type": "eth",
      "number": "0",
```

# Run Ohai on node

```
$ ohai | more
```


```
{
  "languages": {
    "ruby": {
      "platform": "x86_64-linux",
      "version": "2.1.4",
      "release_date": "2014-10-27",
      "target": "x86_64-unknown-linux-gnu",
      "target_cpu": "x86_64",
      "target_vendor": "unknown",
      "target_os": "linux",
      "host": "x86_64-unknown-linux-gnu",
      "host_cpu": "x86_64",
      "host_os": "linux-gnu",
      "host_vendor": "unknown",
      "bin_dir": "/opt/chefdk/embedded/bin",
      "ruby_bin": "/opt/chefdk/embedded/bin/ruby",
```

# Node data

- To Chef, your infrastructure is just large hashes of data stored as JSON
- Much of that data comes from Ohai
  - e.g. The “languages” hash is detected via Ohai
- Some of that data is generated during a chef-client run
  - e.g. node name is generated by chef-client
- All of this data is indexed and available for search
- When using a Chef Server, all of your infrastructure configuration data is aggregated in one central location

# Show specific attributes with knife

```
$ knife node show $(hostname -f) --local-mode -a ipaddress
```

```
WARNING: No knife configuration file found
ip-172-31-7-47.us-west-2.compute.internal:
  ipaddress: 172.31.7.47
```

# Show specific attributes with knife

```
$ knife node show $(hostname -f) --local-mode -a cpu.0.model_name
```

```
WARNING: No knife configuration file found
ip-172-31-7-47.us-west-2.compute.internal:
  cpu.0.model_name: Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
```

# Show specific attributes with knife

```
$ knife node show $(hostname -f) --local-mode -a name
```

```
WARNING: No knife configuration file found
ip-172-31-25-147.us-west-2.compute.internal:
  name: ip-172-31-25-147.us-west-2.compute.internal
```

# Include node name in the index.html template

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

```
<html>
 <body>
 <h1>Hello from <%= node.name %></h1>
 </body>
</html>
```

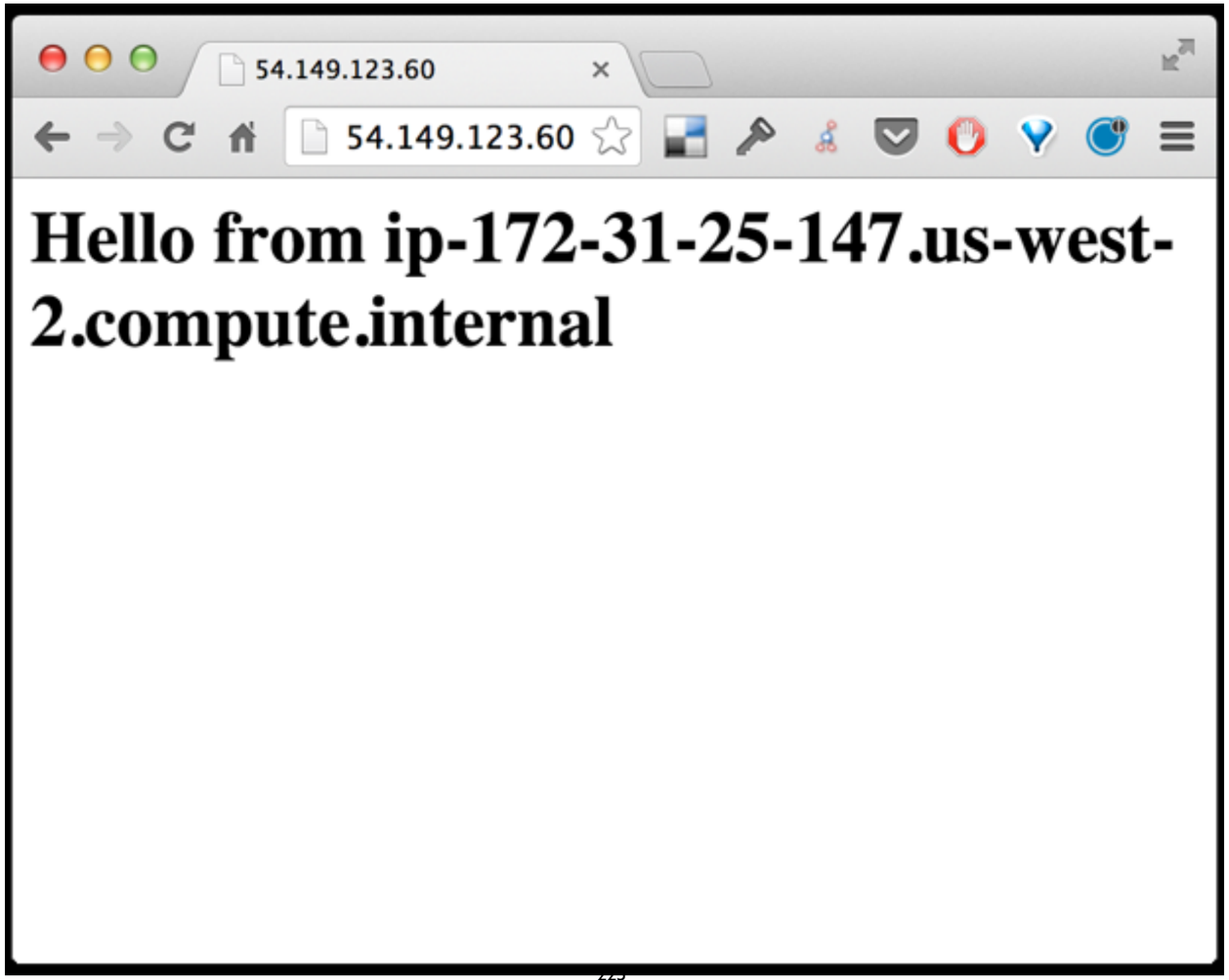
# Apply our recipe using chef-client

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

```
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
  * package[httpd] action install (up to date)
  * service[httpd] action enable (up to date)
  * service[httpd] action start (up to date)
  * template[/var/www/html/index.html] action create
    - update content in file /var/www/html/index.html from e043d5 to c1d241
    --- /var/www/html/index.html    2015-01-11 12:05:11.283712169 +0000
    +++ /tmp/chef-rendered-template20150111-8914-mzgycj   2015-01-11 13:05:09.112712168 +0000
    @@ -1,6 +1,6 @@
<html>
     <html>
      <body>
    -  <h1>Hello, World!</h1>
    +  <h1>Hello from ip-172-31-25-147.us-west-2.compute.internal
      </body>
     </html>
    - restore selinux security context
```

# Congratulations!

- You have just deployed a dynamic website!



# Summary

- How is infrastructure referenced by Chef?
- Where do we get most data in those JSON hashes?
- Which three resources are the most commonly found resources in configuration management?

# Summary

- What questions can I answer for you?

# Wrap Up

## Recap & Next Steps

# Tool Survey

- chef-apply
- chef
- chef-client in local mode
- knife in local mode

# Vocabulary

- Resources
- Recipes
- Cookbooks
- Nodes

# Resources

- Package
- File
- Template
- Service

# Continued Learning

- The LearnChef Site
  - Guided Tutorials
  - Chef Fundamentals intro

    http://learnchef.com
- How-To's, Conference Talks, Webinars, more

    http://youtube.com/user/getchef
- Attend a Chef Fundamentals Class

# Further Resources

- http://chef.io
- http://docs.chef.io
- http://supermarket.chef.io
- http://lists.opscode.com
- irc.freenode.net #chef
- Twitter @chef #getchef, @learnchef #learnchef

# Chef Fundamentals (Local)

- 2 Day Chef Fundamentals (San Francisco)
  - February 3-4, 2015, 9am-5pm
- 2 Day Chef Fundamentals (San Francisco)
  - March 5-6, 2015, 9am-5pm

- **25% OFF, use discount code: TSW25**

# Food Fight Show

- http://foodfightshow.org
- The Podcast Where DevOps Chef Do Battle
- Regular updates about new Cookbooks, Knife-plugins, and more
- Best Practices for working with Chef


@sigje

sigje@chef.io