

# Intermediate Chef

[training@chef.io](mailto:training@chef.io)

Copyright (C) 2014 Chef Software, Inc.

# Introductions

v1.1.6

# Introduce Yourselves

- Name
- Current job role
- Previous job roles/background
- Experience with Chef and/or config management
- Location
- What are you hoping to get out of the class?

# Course Objectives and Style

v1.1.6

# Course Objectives

- Upon completion of this course you will be able to:
  - Extend Chef with custom resources and providers
  - Understand the internals of a Chef Client run
  - Create, debug, and distribute custom Ohai plugins
  - Use the chef-shell debugger to interactively debug cookbooks
  - Configure report and exception handlers
  - Avoid common cookbook errors using Foodcritic and Rubocop
  - Gain an introduction to unit testing using ChefSpec

# Course Pre-Requisites

- Completed Chef Fundamentals or equivalent experience
- You should install the most recent version of Chef Development Kit from here
  - <http://downloads.chef.io/chef-dk/>
- You can check your version with the command `chef -v`

# How to learn 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 teach you how to express solutions to your problems with Chef

# Training is really a discussion

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

# Training is really a discussion

- I'll post objectives at the beginning of a section
- Ask questions when they come to you
- Ask for help when you need it
- You'll get the slides at the **end of class**

# Agenda

v1.1.6

# Topics

- Chef Fundamentals Refresher
- Building Custom Resources
- chef-shell - The Chef Debugger
- Writing Ohai Plugins
- Chef Client Run Internals
- Implementing Chef Handlers
- Correctness Checking Your Cookbooks
- An Introduction to ChefSpec
- Further Resources

# Breaks!

- We'll take a break between each section, or every hour, whichever comes first
- We'll obviously break for lunch :)

# Legend

v1.1.6

## Legend: Do I run that command on my workstation?

This is an example of a command to run on your workstation

```
$ whoami  
i-am-a-workstation
```

This is an example of a command to run on your target node via SSH.

```
user@hostname:~$ whoami  
i-am-a-chef-node
```

# Legend: Example Terminal Command and Output

```
$ ifconfig
```

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    options=3<RXCSUM,TXCSUM>
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        inet 127.0.0.1 netmask 0xff000000
            inet6 ::1 prefixlen 128
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    ether 28:cf:e9:1f:79:a3
    inet6 fe80::2acf:e9ff:fef1f:79a3%en0 prefixlen 64 scopeid 0x4
        inet 10.100.0.84 netmask 0xffffffff broadcast 10.100.0.255
            media: autoselect
            status: active
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
    ether 0a:cf:e9:1f:79:a3
    media: autoselect
    status: inactive
```

# Legend: Example of editing a file on your workstation



**OPEN IN EDITOR:** ~/hello\_world

Hi!

I am a friendly file.

**SAVE FILE!**

# Lab Environment

v1.1.6

# Lab VM

- <http://bit.ly/2zbi9Zw>

- User: chef
- Password: chef.io

# Lab - Login

```
$ ssh chef@<EXTERNAL_ADDRESS>
```

```
The authenticity of host 'uvolqrwls0jdgs3blvt.vm.cld.sr  
(69.195.232.110)' can't be established.  
RSA key fingerprint is d9:95:a3:b9:02:27:e9:cd:  
74:e4:a2:34:23:f5:a6:8b.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'uvolqrwls0jdgs3blvt.vm.cld.sr,  
69.195.232.110' (RSA) to the list of known hosts.  
chef@uvolqrwls0jdgs3blvt.vm.cld.sr's password:  
Last login: Mon Jan  6 16:26:24 2014 from  
host86-145-117-53.range86-145.btcentralplus.com  
[chef@CentOS63 ~]$
```

# Checkpoint

- At this point you should have
  - One virtual machine (VM) or server that you'll use for the lab exercises
  - The IP address or public hostname
  - An application for establishing an ssh connection
  - 'sudo' or 'root' permissions on the VM

# Chef Fundamentals Refresher

v1.1.6

# Lesson Objectives

- After this lesson the, you will be able to
  - Create a new Organization
  - Manually configure your certificates (.pem files) & knife.rb on your workstation so it can communicate with Chef Server
  - Bulk upload all cookbook, role, environment and data bag files to the Chef Server

# Problem Statement

- **Problem:** We want to pick up where we left off after Chef Fundamentals class
- **Proposed Solution:** We need to
  - Set up a new Organization in Hosted Chef
  - Configure your workstation to communicate with new Org
  - Download the Chef Fundamentals content from a GitHub repo
    - cookbooks, data bags, environments and roles
  - Upload all artifacts to your new Org
  - Configure server run list
  - Bootstrap server

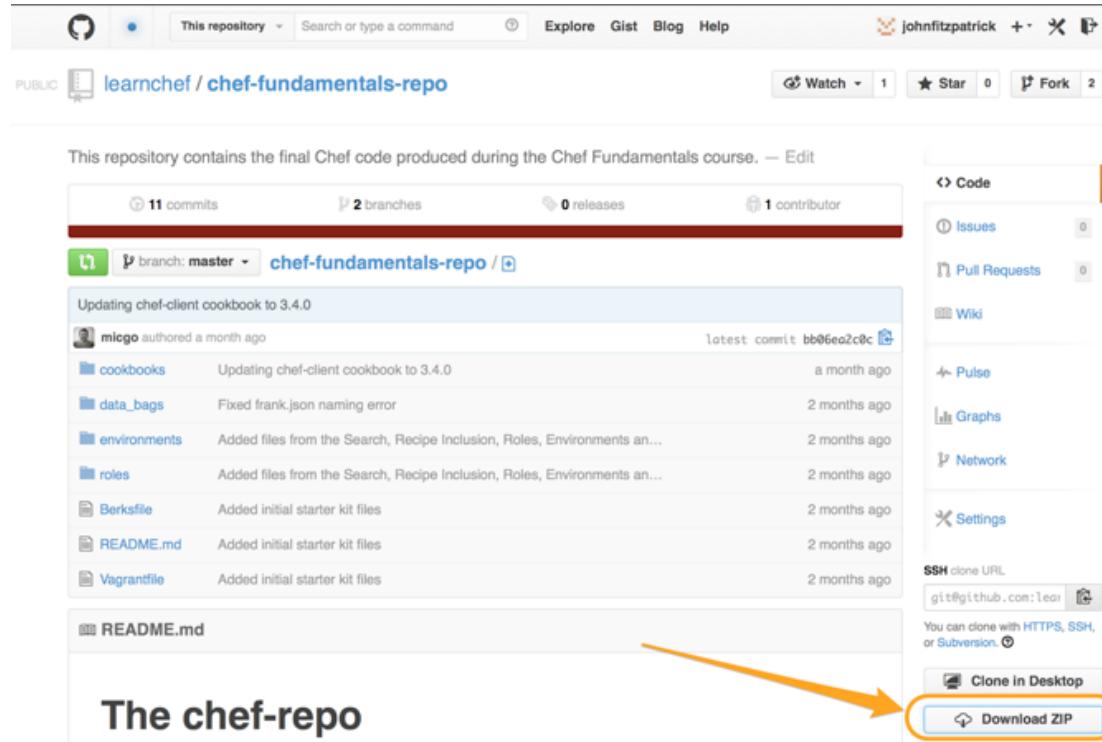
# Exercise: Set up a working directory

- For the purposes of this class make a working directory under your home directory called '~/intermediate', i.e.
  - Windows:-
    - C:\Users\you\intermediate
  - Mac/\*nix:-
    - /Users/you/intermediate
- Navigate to this working Directory

# Exercise: Download the Chef Fundamentals Repo

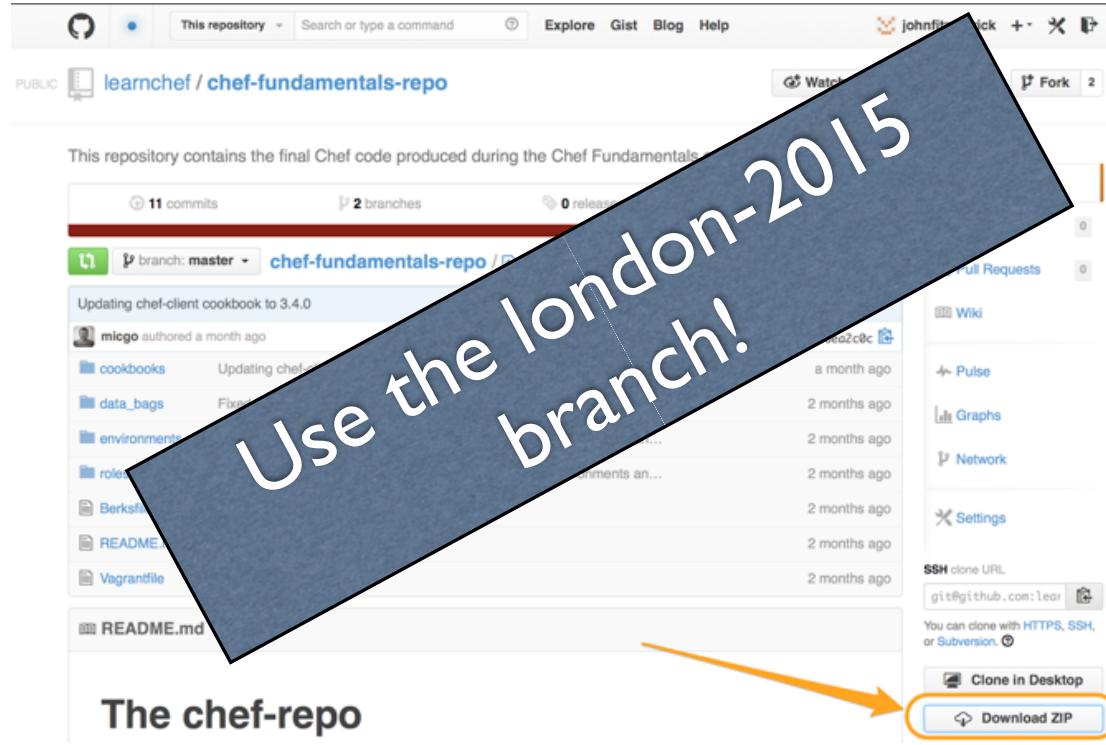
- Download the Chef Fundamentals working repo

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



# Exercise: Download the Chef Fundamentals Repo

- Download the Chef Fundamentals working repo  
<https://github.com/learnchef/chef-fundamentals-repo>



# Exercise: Extract the repo to your working directory

```
$ cp ~/Downloads/chef-fundamentals-repo-london-2015.zip .
$ unzip chef-fundamentals-repo-london-2015.zip
$ mv chef-fundamentals-repo-london-2015 chef-repo
```

```
Archive: chef-fundamentals-repo-london-2015.zip
bb06ea2c0cabaa855e4cb1d1c43bbe4d75caf70d
  creating: chef-fundamentals-repo-london-2015/
  inflating: chef-fundamentals-repo-london-2015/Berksfile
  inflating: chef-fundamentals-repo-london-2015/README.md
  inflating: chef-fundamentals-repo-london-2015/Vagrantfile
  creating: chef-fundamentals-repo-london-2015/cookbooks/
  creating: chef-fundamentals-repo-london-2015/cookbooks/apache/
  inflating: chef-fundamentals-repo-london-2015/cookbooks/apache/CHANGELOG.md
  inflating: chef-fundamentals-repo-london-2015/cookbooks/apache/README.md
  creating: chef-fundamentals-repo-london-2015/cookbooks/apache/attributes/
  creating: chef-fundamentals-repo-london-2015
```

...

# Exercise: View your working directory

```
$ cd chef-repo  
$ ls -a
```

```
.  Berksfile  Vagrantfile  data_bags      roles  
..  README.md  cookbooks   environments
```

- Notice there is no '.chef' directory here
- You need to create one and place your 'knife.rb' file, 'validator.pem' and your 'client.pem' in it

# So what's in our working directory now?

```
cookbooks
├── apache
├── chef-client
├── cron
├── logrotate
├── motd
├── ntp
├── pci
├── starter
└── users

9 directories
```

```
environments
├── dev.rb
└── production.rb

2 files
```

```
roles
├── base.rb
├── starter.rb
└── web.rb
```

3 files

```
data_bags
├── groups
│   └── clowns.json
└── users
    ├── bobo.json
    └── frank.json
```

2 directories, 3 files

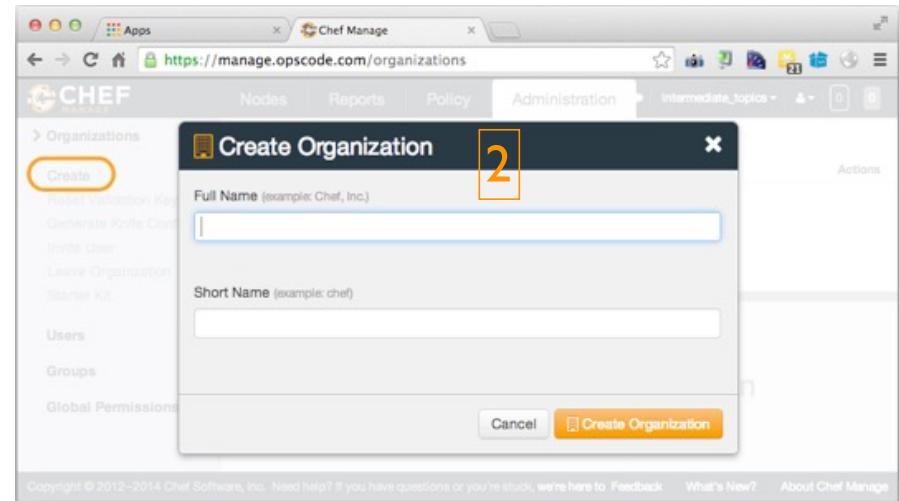
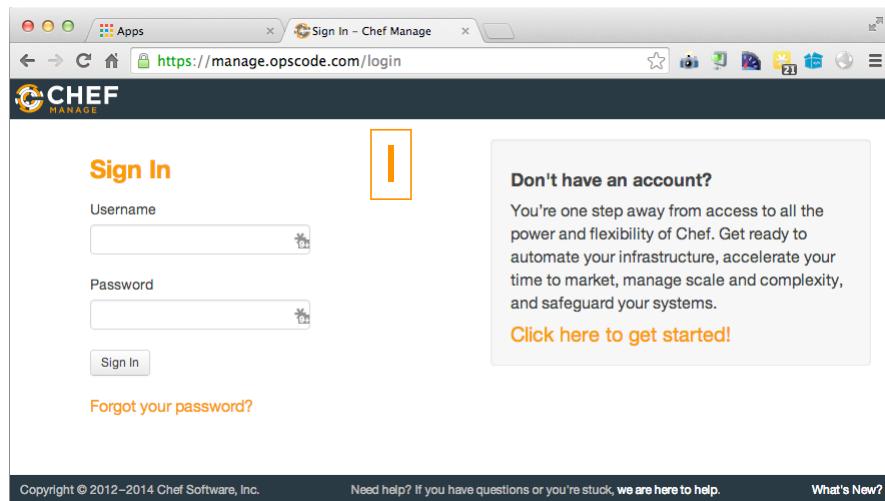
- These are the artifacts created in Chef Fundamentals
- Everything should be uploaded to the Chef Server

# Exercise: Set-up Hosted Chef

- Create a new account on Hosted Chef
- Configure your workstation to connect to Hosted Chef
  - What is required for this?
- `knife client list` should show your validator client

# Exercise: Create a New Org

- Visit Hosted Chef ([manage.chef.io](https://manage.chef.io))
- Sign in or create a new account
- Create a new Organization



# Configuring your Workstation

- In Chef Fundamentals you set your workstation up the easy way using 'Starter Kit'
- In this class you will download your .pem files and knife.rb and configure it manually

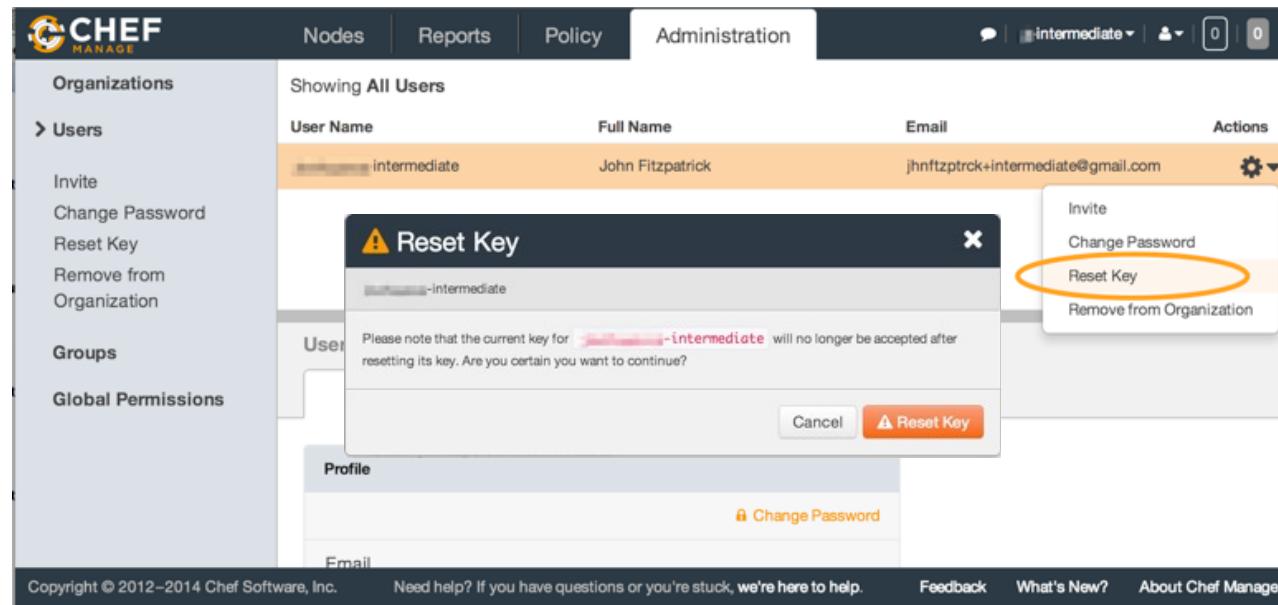
# Exercise: Download knife.rb and validator pem

- Download the knife.rb and validator pem for your Org

The screenshot shows the Chef Manage interface. On the left, the sidebar includes 'Organizations', 'Create', 'Reset Validation Key' (highlighted with an orange circle), 'Generate Knife Config' (highlighted with an orange circle), 'Invite User', 'Leave Organization', and 'Starter Kit'. Under 'Users', 'Groups', and 'Global Permissions', there are no items listed. The main content area shows 'Showing All Organizations' with a single entry: 'Organization: jf-intermediate' (highlighted with an orange box). Below it, the 'Members' tab is selected, showing a table with one row: 'jhntftzptrck-intermediate' with a 'Remove' button. A search bar with 'Search Members...' and a magnifying glass icon is also present. At the bottom, there are links for 'Copyright © 2012–2014 Chef Software, Inc.', 'Need help? If you have questions or you're stuck, we're here to help.', 'Feedback', 'What's New?', and 'About Chef Manage'. A modal window titled 'Reset Key' is open over the organization list. It contains a large block of RSA private key text, starting with '-----BEGIN RSA PRIVATE KEY-----' and ending with '-----END RSA PRIVATE KEY-----'. An orange arrow points from the 'Download' button at the bottom right of the modal to the end of the key text.

# Exercise: Download your client pem

- Reset and download your private client pem file
- **Only do this if you don't already have the .pem file available on your laptop**



# Exercise: Create and populate a .chef directory

- knife.rb & .pem files reside in the .chef directory which can be in
  1. <current-directory>/ .chef
  2. /etc/.chef
  3. ~/ .chef

# Exercise: Create and populate a .chef directory

```
$ cd ~/intermediate/chef-repo  
$ mkdir .chef  
$ cp ~/Downloads/<yourname>.pem .chef  
$ cp ~/Downloads/<your-org>-validator.pem .chef  
$ cp ~/Downloads/knife.rb .chef
```

- knife.rb & .pem files reside in the .chef directory which can be in
  1. <current-directory>/ .chef
  2. /etc/.chef
  3. ~/ .chef

# What have we just done?

- You should now have these 3 files from Hosted Chef
  - <yourname>.pem
  - <your-org>-validator.pem
  - knife.rb
- Later you will move these to your .chef directory

# Exercise: Test your workstation

```
$ knife client list
```

<your-org>-validator.pem

# Exercise: Upload Cookbooks

```
Uploading apache          [ 0.2.0 ]
Uploading chef-client      [ 3.4.0 ]
Uploading cron             [ 1.2.8 ]
Uploading logrotate         [ 1.4.0 ]
Uploading motd              [ 0.1.0 ]
Uploading ntp                [ 1.5.4 ]
Uploading pci                [ 0.1.0 ]
Uploading starter            [ 1.0.0 ]
Uploading users              [ 0.1.0 ]
Uploaded all cookbooks.
```

# Exercise: Upload Cookbooks

```
$ knife cookbook upload -a
```

```
Uploading apache          [ 0.2.0 ]
Uploading chef-client      [ 3.4.0 ]
Uploading cron             [ 1.2.8 ]
Uploading logrotate         [ 1.4.0 ]
Uploading motd              [ 0.1.0 ]
Uploading ntp                [ 1.5.4 ]
Uploading pci                [ 0.1.0 ]
Uploading starter            [ 1.0.0 ]
Uploading users              [ 0.1.0 ]
Uploaded all cookbooks.
```

# Exercise: Upload data\_bags

```
Created data_bags/groups
Created data_bags/users
Created data_bags/groups/clowns.json
Created data_bags/users/bobo.json
Created data_bags/users/frank.json
```

# Exercise: Upload data\_bags

```
$ knife upload data_bags
```

```
Created data_bags/groups
Created data_bags/users
Created data_bags/groups/clowns.json
Created data_bags/users/bobo.json
Created data_bags/users/frank.json
```

# Exercise: Upload Roles

Updated Role base!  
Updated Role starter!  
Updated Role web!

# Exercise: Upload Roles

```
$ knife role from file base.rb starter.rb web.rb
```

```
Updated Role base!
Updated Role starter!
Updated Role web!
```

# Exercise: Upload Environments

Updated Environment dev  
Updated Environment production

# Exercise: Upload Environments

```
$ knife environment from file dev.rb production.rb
```

```
Updated Environment dev
Updated Environment production
```

# Exercise - bootstrap your node

- username: chef
- password: chef.io
- It should have the ‘web’ role assigned

# "Bootstrap" the Target Instance

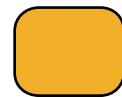
```
$ knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```

```
uv0164727i3mvh1jup2.vm.cld.sr --2014-05-13 04:31:10-- https://www.opscode.com/chef/install.sh
uv0164727i3mvh1jup2.vm.cld.sr Resolving www.opscode.com... 184.106.28.90
uv0164727i3mvh1jup2.vm.cld.sr Connecting to www.opscode.com|184.106.28.90|:443... connected.
uv0164727i3mvh1jup2.vm.cld.sr HTTP request sent, awaiting response... 200 OK
uv0164727i3mvh1jup2.vm.cld.sr Length: 15934 (16K) [application/x-sh]
uv0164727i3mvh1jup2.vm.cld.sr Saving to: `STDOUT'
uv0164727i3mvh1jup2.vm.cld.sr
100%[=====] 15,934      --.-K/s   in 0s
uv0164727i3mvh1jup2.vm.cld.sr
uv0164727i3mvh1jup2.vm.cld.sr 2014-05-13 04:31:10 (538 MB/s) - written to stdout [15934/15934]
uv0164727i3mvh1jup2.vm.cld.sr
uv0164727i3mvh1jup2.vm.cld.sr Downloading Chef 11.8.2 for el...
uv0164727i3mvh1jup2.vm.cld.sr downloading https://www.opscode.com/chef/metadata?
v=11.8.2&prerelease=false&nightlies=false&p=el&pv=6&m=x86_64
uv0164727i3mvh1jup2.vm.cld.sr  to file /tmp/install.sh.41533/metadata.txt
uv0164727i3mvh1jup2.vm.cld.sr trying wget...
uv0164727i3mvh1jup2.vm.cld.sr url https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/
chef-11.8.2-1.el6.x86_64.rpm
...
```

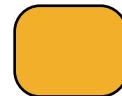
# knife bootstrap



Chef Server



Workstation



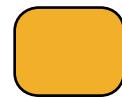
Node

# knife bootstrap

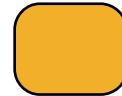
```
knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```



Chef Server



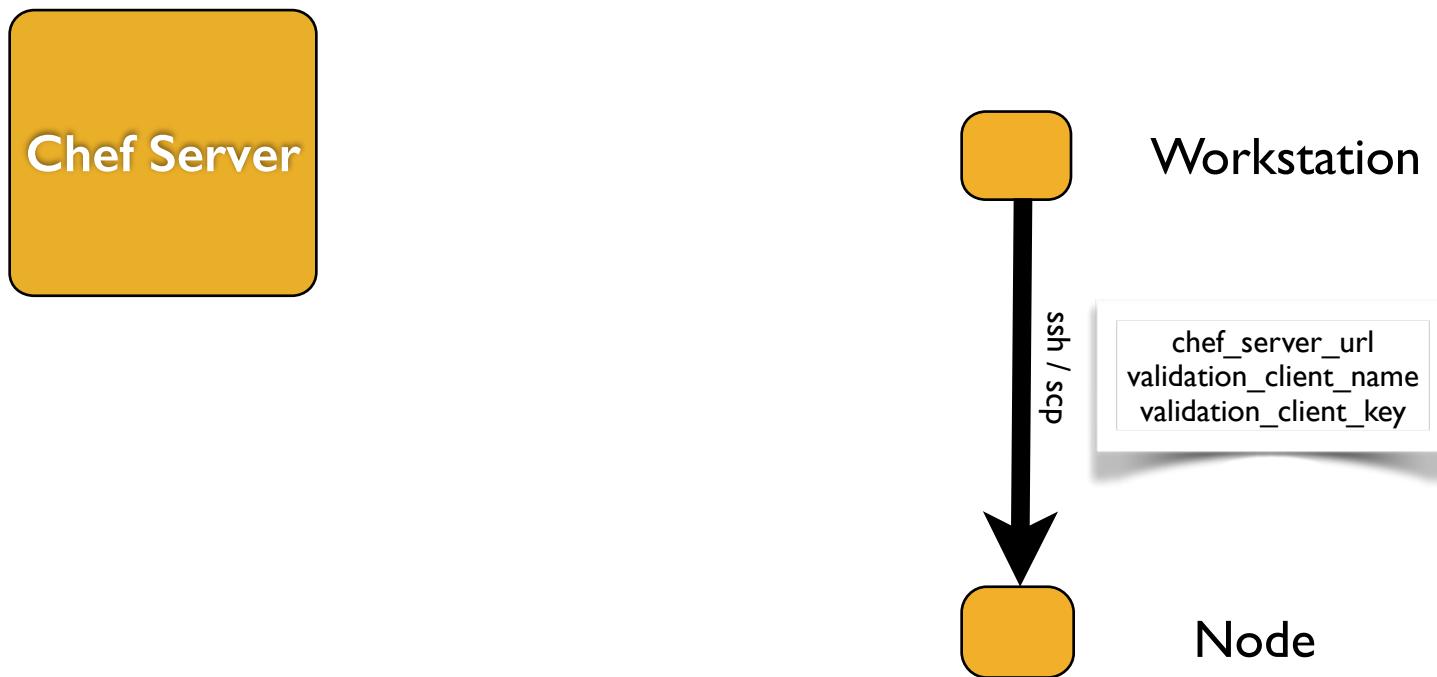
Workstation



Node

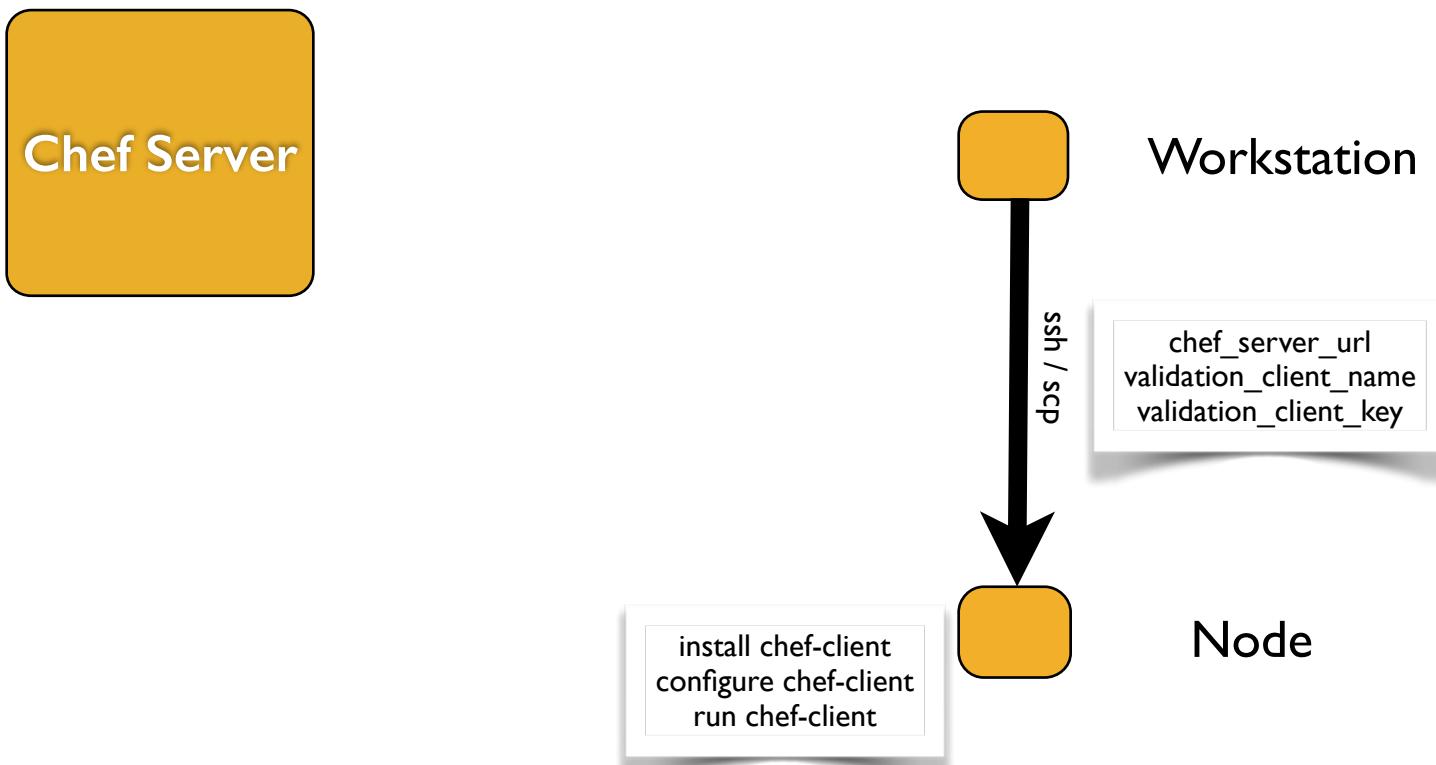
# knife bootstrap

```
knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```



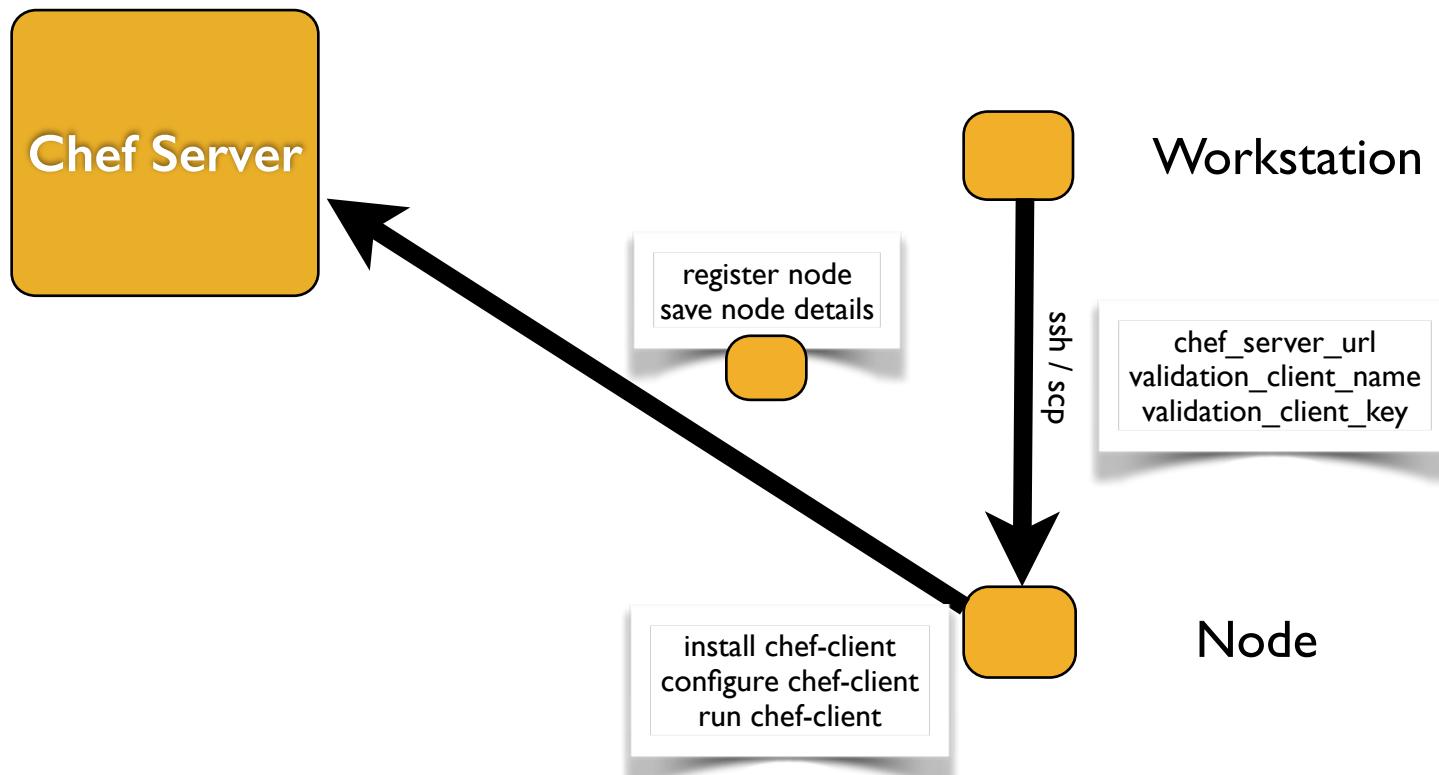
# knife bootstrap

```
knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```



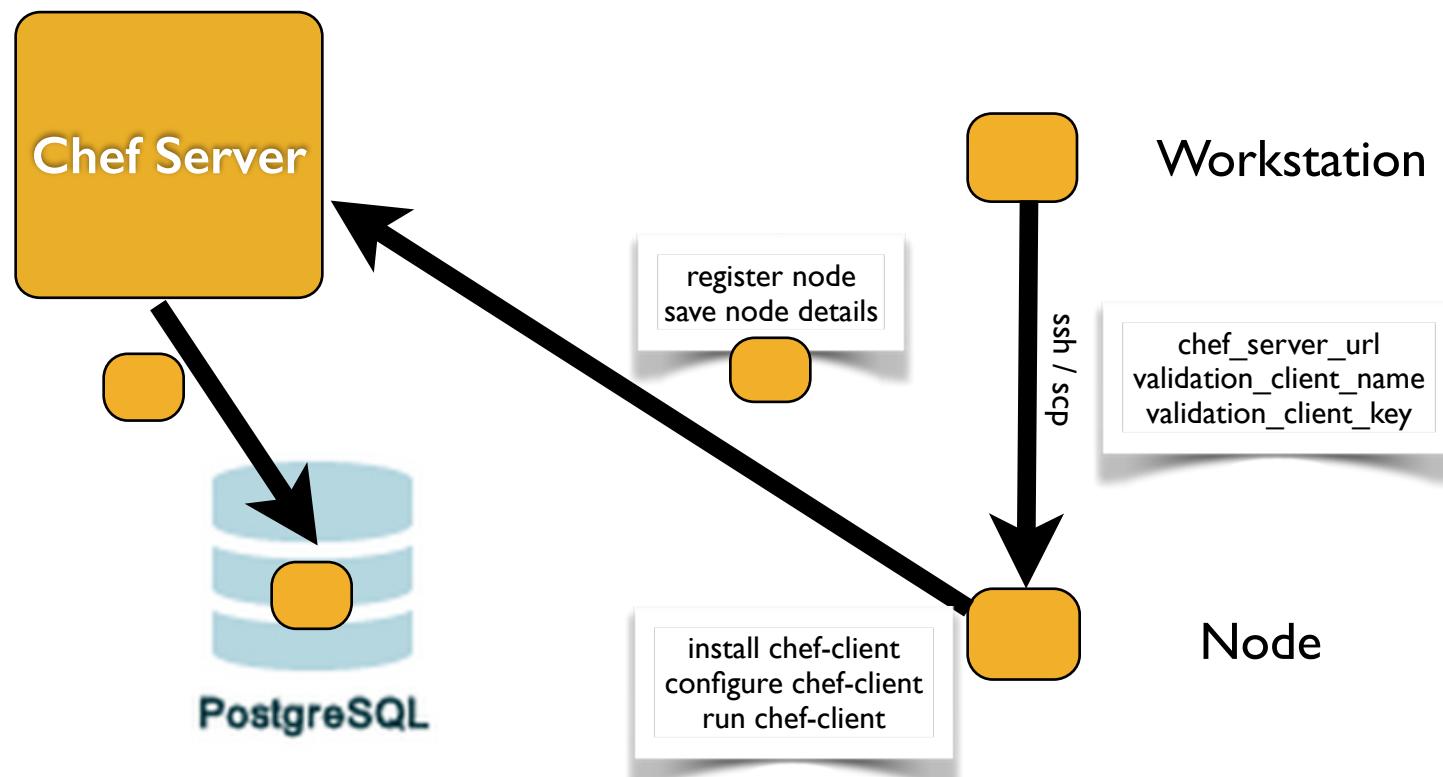
# knife bootstrap

```
knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```



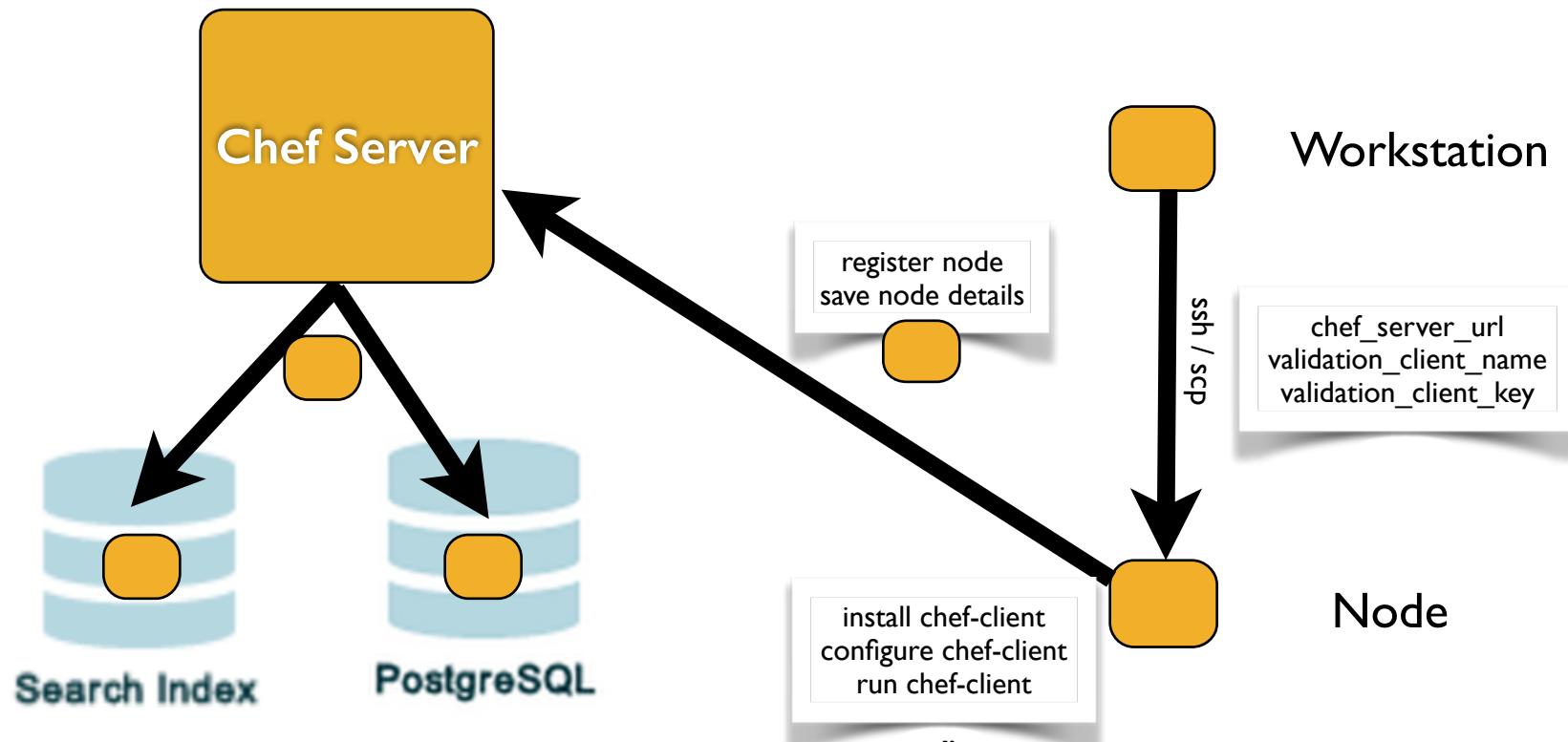
# knife bootstrap

```
knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```



# knife bootstrap

```
knife bootstrap <EXTERNAL_ADDRESS> --sudo -x chef -P chef.io \
-N "node1" -r "role[web]"
```



# What just happened?

- Chef and all of its dependencies installed via an operating system-specific package ("omnibus installer")
- Installation includes
  - The Ruby language - used by Chef
  - knife - Command line tool for administrators
  - chef-client - Client application
  - ohai - System profiler
  - ...and more

## Verify Your Target Instance's Chef-Client is Configured Properly

```
$ ssh chef@<EXTERNAL_ADDRESS>

chef@node1:~$ ls /etc/chef
client.pem  client.rb  first-boot.json validation.pem

chef@node1:~$ which chef-client
/usr/bin/chef-client

chef@node1:~$ chef-client -v
Chef: 11.12.8
```

# SSL Problem?

- You may have seen an SSL error

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
SSL validation of HTTPS requests is disabled. HTTPS connections are still  
encrypted, but chef is not able to detect forged replies or man in the middle  
attacks.
```

To fix this issue add an entry like this to your configuration file:

```
...  
# Verify all HTTPS connections (recommended)  
ssl_verify_mode :verify_peer  
  
# OR, Verify only connections to chef-server  
verify_api_cert true  
...
```

To check your SSL configuration, or troubleshoot errors, you can use the  
`knife ssl check` command like so:

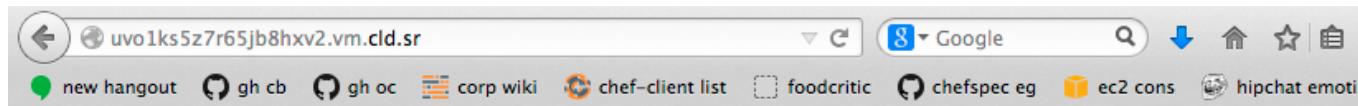
- This is harmless for the purpose of this course
- But, we will fix it in the next section!

# Exercise: Verify chef-client is running

```
chef@node1$ ps awux | grep chef-client
```

```
root      8933  0.3  2.2 130400 37816 ?          S1
03:19  0:01 /opt/chef/embedded/bin/ruby /usr/bin/
chef-client -d -c /etc/chef/client.rb -L /var/log/
chef/client.log -P /var/run/chef/client.pid -i 1800
-s 300
```

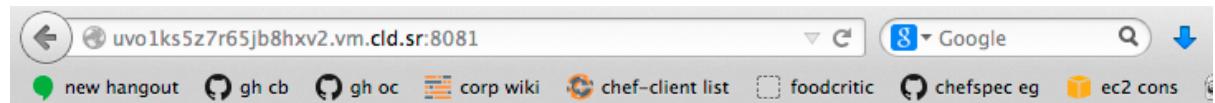
# Exercise: Verify the two sites are working!



Welcome to Chef

We love clowns

10.160.201.90:80



Welcome to Chef

We love bears

10.160.201.90:8081

# Review Questions

- What files did we download from Chef Server to configure our workstation?
- Where did we place these files?
- What other way could we have copied the repo files from GitHub?
  - Bonus point - why did we not do this?
- How did we upload all cookbooks to the Chef Server?

# Configuring Chef Clients

SSL Verification and chef-client management

v1.1.6

# Lesson Objectives

- After completing the lesson, you will be able to
  - Use the chef-client cookbook to manage the Chef Client service on your nodes
  - Configure nodes to use SSL Certificate Validation

# The Problem and Success Criteria

- **The Problem:** We want to make sure the Chef Client service is started and enabled on boot. We also want to make sure Chef Client is validating all SSL connections.
- **Success Criteria:** We will use the chef-client cookbook to accomplish these tasks.

# Using the chef-client cookbook

- The chef-client cookbook manages the Chef Client's:
  - `client.rb` configuration
  - service (runit, systemd, init, SMF, etc.)
  - validation key deletion

# Examining the chef-client cookbook

- We're already using two recipes on the node from the chef-client cookbook
  - `chef-client::service`  
**(via** `chef-client::default`)
  - `chef-client::delete_validation`

# Exercise: View node1's Run List

```
Node Name:    node1
Environment:  _default
FQDN:        centos63.example.com
IP:          10.160.201.90
Run List:    role[web]
Roles:       web, base
Recipes:     chef-client::delete_validation, chef-client, ntp, motd, users,
             apache, chef-client::default, chef-client::service, chef-client::init_service,
             ntp::default, motd::default, users::default, users::groups, apache::default
Platform:    centos 6.4
Tags:
```

# Exercise: View node1's Run List

```
$ knife node show node1
```

```
Node Name:      node1
Environment:    _default
FQDN:          centos63.example.com
IP:            10.160.201.90
Run List:       role[web]
Roles:          web, base
Recipes:        chef-client::delete_validation, chef-client, ntp, motd, users,
                apache, chef-client::default, chef-client::service, chef-client::init_service,
                ntp::default, motd::default, users::default, users::groups, apache::default
Platform:       centos 6.4
Tags:
```

# Exercise: View the chef-client::config recipe



**OPEN IN EDITOR:** cookbooks/chef-client/recipes/config.rb

```
template "#{node['chef_client']['conf_dir']}/client.rb" do
  source 'client.rb.erb'
  owner d_owner
  group d_group
  mode 00644
  variables(
    :chef_config => node['chef_client']['config'],
    :chef_requires => chef_requires,
    :ohai_disabled_plugins => node['ohai']['disabled_plugins'],
    :start_handlers => node['chef_client']['config']['start_handlers'],
    :report_handlers => node['chef_client']['config']['report_handlers'],
    :exception_handlers => node['chef_client']['config']
    ['exception_handlers']
  )
  notifies :create, 'ruby_block[reload_client_config]', :immediately
end
```

# Managing client.rb settings

- The config recipe is used to dynamically generate the `/etc/chef/client.rb` config file
- The template walks all attributes in `node['chef_client']['config']` and writes them out as key:value pairs
- The key should be the configuration directive
- The `chef_server_url`, `node_name` and `validation_client_name` are set by default in the attributes file from `Chef::Config`

# Managing client.rb settings

- For example, the following attributes (in a role):

```
default_attributes(  
  "chef_client" => {  
    "config" => {  
      "ssl_verify_mode" => ":verify_peer",  
      "log_level" => ":info"  
    }  
  }  
)
```

- Will render the following configuration (/etc/chef/client.rb):

```
chef_server_url "https://api.opscode.com/organizations/MYORG"  
validation_client_name "MYORG-validator"  
ssl_verify_mode :verify_peer  
node_name "config-ubuntu-1204"  
log_level :info
```

# SSL Certificate Validation

- Chef defaults to not verifying certificates when it makes HTTPS connections - hence the error when running chef-client
- Enable SSL Verification by setting `ssl_verify_mode :verify_peer` in your config file
- Two useful knife commands:-
  - **knife ssl check** - makes an SSL connection to your Chef server or any other HTTPS server and tells you if the server presents a valid certificate
  - **knife ssl fetch** - allows you to automatically fetch a server's certificates to your trusted certs directory

# Log Levels

- chef-client defaults to the log level **:auto**
- chef-client supports additional levels
  - **:debug**
  - **:info**
  - **:warn**
  - **:error**
  - **:fatal**
- Reference: [https://docs.chef.io/config\\_rb\\_client.html](https://docs.chef.io/config_rb_client.html)

# Exercise: Add chef-client::config to the base role



**OPEN IN EDITOR:** roles/base.rb

```
name "base"
description "Base Server Role"
run_list "recipe[chef-client::delete_validation]",
"recipe[chef-client::config]", "recipe[chef-client]",
"recipe[ntp]", "recipe[motd]", "recipe[users]"
default_attributes({
  "chef_client" => {
    "config" => {
      "ssl_verify_mode" => ":verify_peer",
      "log_level" => ":info"
    }
  }
})
```

**SAVE FILE!**

# Exercise: Upload the base role

```
$ knife role from file base.rb
```

Updated Role base!

# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
Starting Chef Client, version 11.12.8
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client::config",
"chef-client", "ntp", "motd", "users", "apache"]
Synchronizing Cookbooks:
 - chef-client
 - cron
 - logrotate
 - ntp
 - motd
 - pci
 - users
 - apache
 - ohai
Compiling Cookbooks...
...
```

# Exercise: Verify config updated

```
chef@node1$ sudo cat /etc/chef/client.rb
```

```
...
ssl_verify_mode :verify_peer
log_level :info
...
```

# Special Handling for Non-CA Signed Certificates

- Hosted Chef's certificate is signed by a known CA
- However, if you run your own CA, or your Chef Server's certificate is self-signed, you can use knife to import it

# Exercise: Fetch Hosted Chef's SSL certificate

```
$ knife ssl fetch
```

WARNING: Certificates from api.opscode.com will be fetched and placed in your trusted\_cert directory (C:/Users/you/.chef/trusted\_certs).

Knife has no means to verify these are the correct certificates. You should verify the authenticity of these certificates after downloading.

```
Adding certificate for *.opscode.com in C:/Users/you/.chef/trusted_certs/wildcard_opscode_com.crt
```

```
Adding certificate for DigiCert Secure Server CA in C:/Users/you/.chef/trusted_certs/DigiCert_Secure_Server_CA.crt
```

# Review Questions

- How do you remove the validation .pem on the node?
- What recipe is used to configure chef-client?
- How do you enable SSL Verification on chef-client?
- How do you fetch a Chef Server's certificates to your trusted certs directory?

# Building Custom Resources

Expanding functionality in the Chef Framework

v1.1.6

# Lesson Objectives

- After completing the lesson, you will be able to
  - Describe the components of the Lightweight Resource/Provider (LWRP) framework
  - Explain the Resource DSL
  - Explain the Provider DSL
  - Build a resource and provider from scratch
  - Look at examples of other LWRPs

# A Brief Review...

- **Resources** are declarative interfaces that describe *what* we want to happen, rather than *how*
- Resources take action through **Providers** that perform the *how*
- Resources
  - have a **type**
  - have a **name**
  - can have one or more **parameters**
  - take **action** to put the resource into a desired state
  - can send **notifications** to other resources

# Other Ways To Extend Chef

- **Definitions**
  - “recipe macro”
  - stored in `definitions/`
  - cannot receive notifications
- **Heavyweight Resources**
  - pure Ruby code
  - stored in `libraries/`
  - cannot use core resources (by default)

# Components of an LWRP

- LWRPs have two components: the resource and the provider
  - **Resources** are used in Recipes to declare the state to configure on our system
  - **Providers** configure that state on the system during convergence
- They are defined in the `resources/` and `providers/` directories of cookbooks

# The Problem and the Success Criteria

- **The Problem:** We want to abstract the Apache virtual host configuration pattern.
- **Success Criteria:** We will create an `apache_vhost` resource that will let us manage Apache virtual hosts.

## Exercise: Change the cookbook's version number in the metadata



**OPEN IN EDITOR:** cookbooks/apache/metadata.rb

```
maintainer          "apache"
maintainer_email    "YOUR_EMAIL"
license             "All rights reserved"
description         "Installs/Configures apache"
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version            "0.3.0"
```

**SAVE FILE!**

- Major, Minor, Patch
- Semantic Versioning Policy: <http://semver.org/>

# Resource & Provider Naming

```
+ cookbooks
  |
  + apache
    |
    + resources
      |
      + vhost.rb
```

- Name of new resource is implied by its name in the cookbook
- In the above scenario, the new resource will be called `apache_vhost`

# The Resource DSL

- Three methods: `actions`, `attribute`, `default_action`
  - `actions` defines a list of allowed actions by the resource
  - `attribute` defines a new parameter for the resource block
  - `default_action` defines the one action to use if no action is specified in the resource block

# Exercise: Review the apache::default recipe



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

```
# Iterate over the apache sites
node["apache"]["sites"].each do |site_name, site_data|
# Set the document root
  document_root = "/srv/apache/#{site_name}"
```

- Note the file cookbooks/apache/attributes/default.rb contains the following

```
default["apache"]["sites"]["clowns"] = { "port" => 80 }
default["apache"]["sites"]["bears"] = { "port" => 81 }
```

# Exercise: Review the apache::default recipe



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

```
# Add a template for Apache virtual host configuration
template "/etc/httpd/conf.d/#{site_name}.conf" do
  source "custom.erb"
  mode "0644"
  variables(
    :document_root => document_root,
    :port => site_data["port"]
  )
  notifies :restart, "service[httpd]"
end
```

# Exercise: Review the apache::default recipe



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

```
# Add a directory resource to create the document_root
directory document_root do
  mode "0755"
  recursive true
end
```

# Exercise: Review the apache::default recipe



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

```
# Add a template resource for the virtual host's
index.html
template "#{document_root}/index.html" do
  source "index.html.erb"
  mode "0644"
  variables(
    :site_name => site_name,
    :port => site_data["port"]
  )
end
```

## Exercise: Create an apache\_vhost resource with two allowed actions



**OPEN IN EDITOR:** cookbooks/apache/resources/vhost.rb

```
actions :create, :remove
```

**SAVE FILE!**

- We have allowed two resource actions

# The Provider DSL

- One method: `action`
  - Specify the action name with a Ruby symbol
  - Name maps to allowed actions defined in the resource `actions`
- You can also re-use any Chef Resources inside your providers

# Exercise: Create provider for the :create action

 **OPEN IN EDITOR:** cookbooks/apache/providers/vhost.rb

```
action :create do
  puts "My name is #{new_resource.name}"
end
```

**SAVE FILE!**

- When our apache\_vhost resource calls action :create, execute this block.

# Exercise: Set an action in our apache::default recipe



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

```
# Disable the default virtual host
execute "mv /etc/httpd/conf.d/
welcome.conf /etc/httpd/conf.d/
welcome.conf.disabled" do
  only_if do
    File.exist?("/etc/httpd/conf.d/
welcome.conf")
  end
  notifies :restart, "service[httpd]"
end

# Enable an Apache Virtualhost
apache_vhost "lions" do
  action :create
end
```

- We call resource **apache\_vhost**
- With name **“lions”**
- With action **:create** in the parameter block

**SAVE FILE!**

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0.3.0 ]  
Uploaded 1 cookbook.
```

# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
Starting Chef Client, version 11.10.4
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "ntp", "motd", "users", "apache"]
Synchronizing Cookbooks:
  - chef-client
  - cron
  - logrotate
  - ntp
  - motd
  - pci
  - users
  - apache
...
* apache_vhost[lions] action createMy name is lions
  (up to date)
...
Running handlers:
Running handlers complete

Chef Client finished, 12/35 resources updated in 12.045321863 seconds
```

# Exercise: Use a Chef Resource within your provider

 **OPEN IN EDITOR:** cookbooks/apache/providers/vhost.rb

```
use_inline_resources
```

```
action :create do
  log "My name is #{new_resource.name}"
end
```

**SAVE FILE!**

- The log resource uses Chef's logger object to print messages at `Chef::Config[:log_level]`

# A word on multiphase execution

v1.1.6

# Multiphase Execution - Compile Phase

- During the compile phase, Chef
  1. Loads all cookbooks from the run list
  2. Reads each recipe to build the resource collection

# Multiphase Execution - Execute Phase

- During the execute phase chef takes the resource collection and for each resource it will
  1. Check if the resource is in the required state
    - If 'yes' - do nothing
    - If 'no' - bring resource in line with required state
  2. Move on to next resource

# Resource Collection Phase 1 - Compile Phase

## Recipe

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

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

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

## Resource Collection

```
resource_collection = [
```

# Resource Collection Phase 1 - Compile Phase

## Recipe

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

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

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

## Resource Collection

```
resource_collection = [
  package[ "httpd" ],
```

# Resource Collection Phase 1 - Compile Phase

## Recipe

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

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

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
```

# Resource Collection Phase 1 - Compile Phase

## Recipe

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

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

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

# Resource Collection Phase 2 - Execute Phase

## Recipe

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

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

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

## Execution

### Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

# Resource Collection Phase 2 - Execute Phase

## Recipe

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

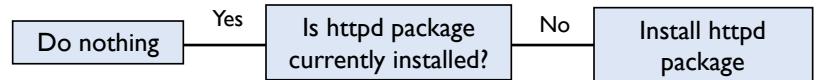
cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

## Execution



# Resource Collection Phase 2 - Execute Phase

## Recipe

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

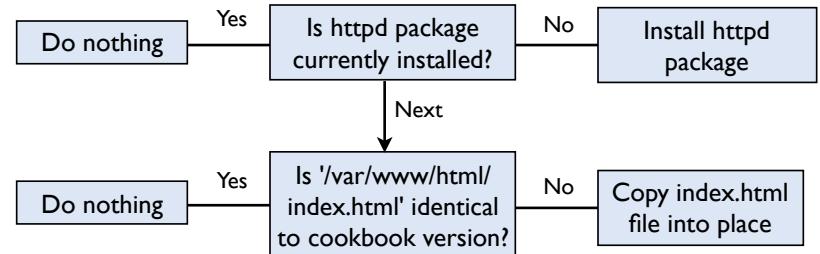
cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

## Execution



# Resource Collection Phase 2 - Execute Phase

## Recipe

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

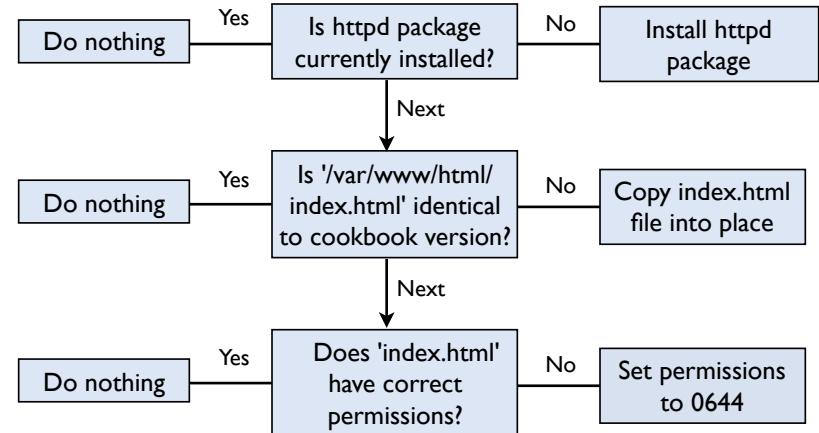
cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

## Execution



# Resource Collection Phase 2 - Execute Phase

## Recipe

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

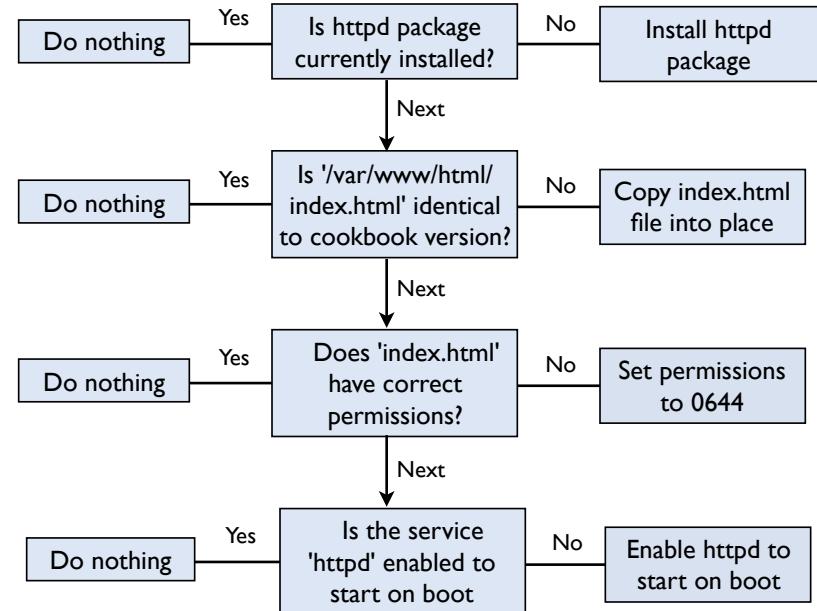
cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

## Execution



# Resource Collection Phase 2 - Execute Phase

## Recipe

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

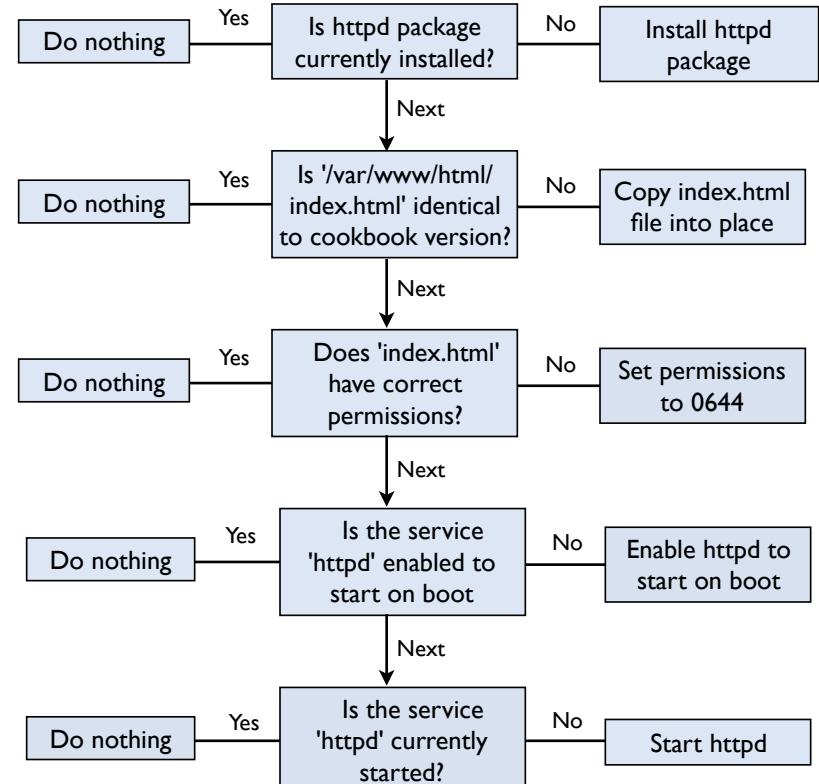
cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end

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

## Resource Collection

```
resource_collection = [
  package["httpd"],
  cookbook_file["/var/www/html/index.html"],
  service ["httpd"]
]
```

## Execution



# Recipe order is important!

- Recipes are executed in the order they appear in the run list

```
Run List: recipe[ntp::client], recipe[openssh::server], recipe[apache::server]
```

- These recipes are invoked in the following order



# Recipe order is important!

- Recipes are executed in the order they appear in the run list

```
Run List: recipe[ntp::client], recipe[openssh::server], recipe[apache::server]
```

- These recipes are invoked in the following order
  1. recipe[ntp::client]
  2. recipe[openssh::server]
  3. recipe[apache::server]

# Resource Collection - Multiple Recipes

1. recipe[ntp::client]

Resource Collection

```
resource_collection [
```

# Resource Collection - Multiple Recipes

## 1. recipe[ntp::client]

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

template "/etc/ntp.conf" do
  source "ntp.conf.erb"
  owner "root"
  mode "0644"
end

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

## Resource Collection

```
resource_collection [
  package[ntp],
  template[/etc/ntp.conf],
  service[ntp]
```

# Resource Collection - Multiple Recipes

2. recipe[openssh::client]

## Resource Collection

```
resource_collection [  
  package[ntp],  
  template[/etc/ntp.conf],  
  service[ntp],
```

# Resource Collection - Multiple Recipes

## 2. recipe[openssh::client]

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

template "/etc/sshd/sshd_config" do
  source "sshd_config.erb"
  owner "root"
  mode "0644"
end

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

## Resource Collection

```
resource_collection [
  package[ntp],
  template[/etc/ntp.conf],
  service[ntp],
  package[openssh],
  template[/etc/sshd/sshd_config],
  service[openssh]
```

# Resource Collection - Multiple Recipes

3. recipe[httpd::server]

## Resource Collection

```
resource_collection [  
  package[ntp],  
  template[/etc/ntp.conf],  
  service[ntp],  
  package[openssh],  
  template[/etc/sshd/sshd_config],  
  service[openssh],
```

# Resource Collection - Multiple Recipes

## 3. recipe[httpd::server]

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

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

cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end
```

## Resource Collection

```
resource_collection [
  package[ntp],
  template[/etc/ntp.conf],
  service[ntp],
  package[openssh],
  template[/etc/sshd/sshd_config],
  service[openssh],
  package[httpd],
  service[httpd],
  cookbook_file[/var/www/html/index.html]
]
```

# The final resource collection

- So the resources are invoked in the following order during the execute phase

```
package[ntp]
template[/etc/ntp.conf]
service[ntp]
package[openssh]
template[/etc/sshd/sshd_config]
service[openssh]
package[httpd]
service[httpd]
cookbook_file[/var/www/html/index.html]
```

# Multiphase Execution

- Plain ruby is executed in the compile phase
- Chef DSL is executed in the execute phase

Recipe

```
%w[sites-available sites-enabled mods-available].each do |dir|
  directory "/var/www/#{dir}" do
    action :create
    mode  '0755'
    owner 'root'
    group node["apache"]["root_group"]
  end
end
```

Resource Collection

# Multiphase Execution

- Plain ruby is executed in the compile phase
- Chef DSL is executed in the execute phase

Recipe

```
%w[sites-available sites-enabled mods-available].each do |dir|
  directory "/var/www/#{dir}" do
    action :create
    mode  '0755'
    owner 'root'
    group node["apache"]["root_group"]
  end
end
```

Resource Collection

```
resource_collection [
  directory[ "/var/www/sites-available" ],
  directory[ "/var/www/sites-enabled" ],
  directory[ "/var/www/mods-available" ],
```

# `use_inline_resources`

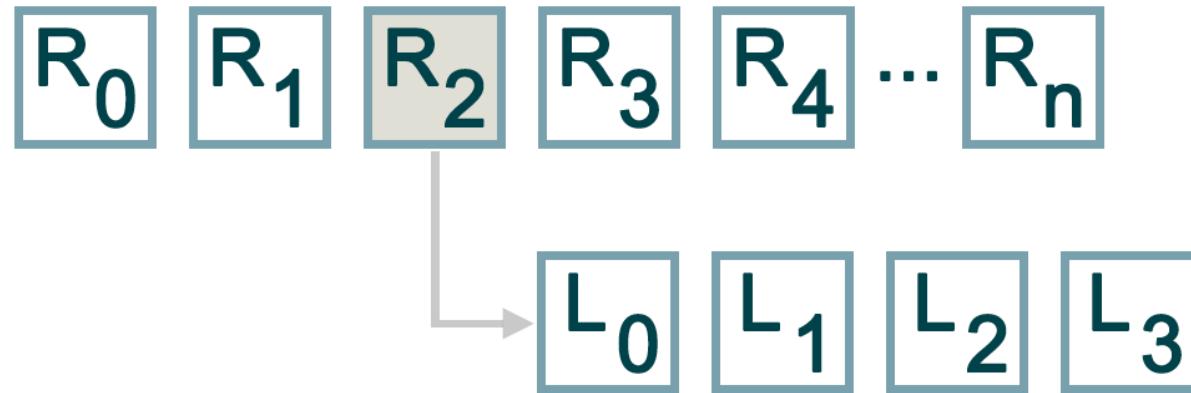
- `use_inline_resources` instructs the embedded resources (“`log`”) to notify the parent (“`apache_vhost`”) if their states change
- More info: [`docs.chef.io/lwrp\_custom\_provider.html#use-inline-resources`](https://docs.chef.io/lwrp_custom_provider.html#use-inline-resources)

# LWRPs and the Resource Collection



- A regular resource collection is just a big array of resources.

# LWRPs and the Resource Collection



- `use_inline_resources` creates a mini resource collection (execution context)
- Notifications are rolled up to the master resource collection.

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0.3.0 ]
Uploaded 1 cookbook.
```

# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
Starting Chef Client, version 11.10.4
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "ntp", "motd", "users", "apache"]
Synchronizing Cookbooks:
  - chef-client
  - cron
  - logrotate
  - ntp
  - motd
  - pci
  - users
  - apache
...
* apache_vhost[lions] action createRecipe: <Dynamically Defined Resource>
* log[My name is lions] action write
...
Running handlers:
Running handlers complete

Chef Client finished, 12/35 resources updated in 12.045321863 seconds
```

## Exercise: Create attribute parameters for the apache\_vhost resource

 **OPEN IN EDITOR:** cookbooks/apache/resources/vhost.rb

```
actions :create, :remove

attribute :site_name, :name_attribute => true, :kind_of => String
attribute :site_port, :default => 80, :kind_of => Fixnum
```

**SAVE FILE!**

- attribute takes the name of the attribute and an (optional) hash of validation parameters
- These validation parameters are specific to the LWRP Resource DSL

# Resource Validation Parameters



**OPEN IN EDITOR:** cookbooks/apache/resources/vhost.rb

```
actions :create, :remove

attribute :site_name, :name_attribute => true, :kind_of => String
attribute :site_port, :default => 80, :kind_of => Fixnum
```

**SAVE FILE!**

- **:name\_attribute** set the site\_name to the resource name
- **:default** sets the default value for this parameter
- **:kind\_of** ensures the value is a particular class

# Resource Validation Parameters

Validation	Meaning
:callbacks	Hash of Procs, should return true
:default	Default value for the parameter
:equal_to	Match the value with ==
:kind_of	Ensure the value is of a particular class
:name_attribute	Set to the resource name
:regex	Match the value against the regex
:required	The parameter must be specified
:respond_to	Ensure the value has a given method

# :kind\_of examples

- `:kind_of` accepts many types, either built-in Ruby classes, or your own

- `:kind_of => String # String`
- `:kind_of => Array # Array`
- `:kind_of => Fixnum # Fixnum`
- `:kind_of => [:some, :list, :of, :symbols]`
- `:kind_of => [TrueClass, FalseClass]`
- `:kind_of => [String, Array] # composite`

# Exercise: Extend :create action



**OPEN IN EDITOR:** cookbooks/apache/providers/vhost.rb

```
use_inline_resources

action :create do
  # Set the document root
  document_root = "/srv/apache/#{new_resource.site_name}"

  # Add a template for Apache virtual host configuration
  template "/etc/httpd/conf.d/#{new_resource.site_name}.conf" do
    source "custom.erb"
    mode "0644"
    variables(
      :document_root => document_root,
      :port => new_resource.site_port
    )
  end
```

**SAVE FILE!**

# Exercise: Extend :create action



**OPEN IN EDITOR:** cookbooks/apache/providers/vhost.rb

```
# Add a directory resource to create the document_root
directory document_root do
  mode "0755"
  recursive true
end

# Add a template resource for the virtual host's index.html
template "#{document_root}/index.html" do
  source "index.html.erb"
  mode "0644"
  variables(
    :site_name => new_resource.site_name,
    :port => new_resource.site_port
  )
end
end
```

**SAVE FILE!**

# Exercise: Use apache\_vhost in a recipe



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

```
# Enable an Apache Virtualhost
apache_vhost "lions" do
  site_port 8080
  action :create
  notifies :restart, "service[httpd]"
end

# Iterate over the apache sites
node["apache"]["sites"].each do |site_name, site_data|
```

**SAVE FILE!**

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0.3.0 ]  
Uploaded 1 cookbook.
```

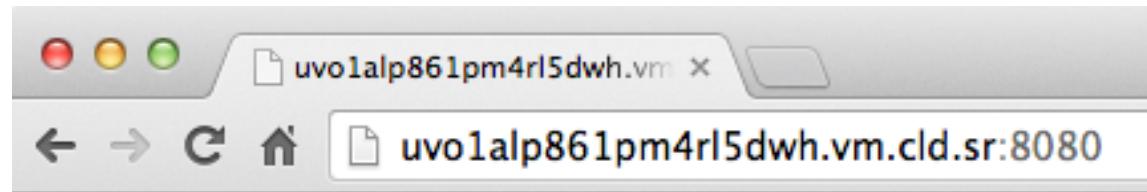
# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
Starting Chef Client, version 11.10.4
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "ntp", "motd", "users", "apache"]
Synchronizing Cookbooks:
  - chef-client
  - cron
  - logrotate
  - ntp
  - motd
  - pci
  - users
  - apache
...
* apache_vhost[lions] action createRecipe: <Dynamically Defined Resource>
* template[/etc/httpd/conf.d/lions.conf] action create
...
Running handlers:
Running handlers complete

Chef Client finished, 7/38 resources updated in 9.724844315 seconds
```

# Exercise: Verify new ‘lions’ site



**Welcome to Chef**

**We love lions**

10.160.157.59:8080

# The resource collection - inline resources

Resource Collection

All notifications in inline resources are rolled up to 'master' resource collection



# The resource collection - inline resources

## Resource Collection

```
resource_collection = [
    ...,
    package[httpd],
    service[httpd],
    apache_vhost[lions],
    resource_collection = [
        template["/etc/httpd/conf.d/lions.conf"],
        directory["/srv/apache/lions"],
        template["/srv/apache/lions/index.html"],
    ],
    template["/etc/httpd/conf.d/clowns.conf"],
    directory["/srv/apache/clowns"],
    template["/srv/apache/clowns/index.html"],
    template["/etc/httpd/conf.d/bears.conf"],
    directory["/srv/apache/bears"],
    template["/srv/apache/clowns/bears.html"]
]
```

All notifications in inline resources are rolled up to 'master' resource collection



# Houston, we have a problem!

- Now that we have an easy way to add virtual hosts, we will quickly run into a data management problem
- We could keep adding more attributes to the apache cookbook, but...
  - that doesn't scale well across teams
  - you can end up with a long, messy list of attributes

# Let's use Data Bags to drive our new LWRP

```
default[ "apache" ][ "sites" ][ "clowns" ] = { "port" => 80 }
default[ "apache" ][ "sites" ][ "bears" ] = { "port" => 81 }
```

- How should we structure our data bags?
  - Data bag for each site?
  - One data bag for apache with items for each site?

## Exercise: Create the apache\_sites data bag

```
$ knife data_bag create apache_sites
```

```
Created data_bag[apache_sites]
```

## Exercise: Create the apache\_sites data bag

```
$ mkdir -p data_bags/apache_sites
```

No output...

# Exercise: Create clowns data bag item



**OPEN IN EDITOR:** data\_bags/apache\_sites/clowns.json

```
{  
  "id": "clowns",  
  "port": 80  
}
```

**SAVE FILE!**

# Exercise: Upload the clowns data bag item

```
$ knife data_bag from file apache_sites clowns.json
```

```
Updated data_bag_item[apache_sites::clowns]
```

# Exercise: Create bears data bag item



**OPEN IN EDITOR:** data\_bags/apache\_sites/bears.json

```
{  
  "id": "bears",  
  "port": 8081  
}
```

**SAVE FILE!**

# Exercise: Upload the bears data bag item

```
$ knife data_bag from file apache_sites bears.json
```

```
Updated data_bag_item[apache_sites::bears]
```

# Exercise: Create lions data bag item



**OPEN IN EDITOR:** data\_bags/apache\_sites/lions.json

```
{  
  "id": "lions",  
  "port": 8080  
}
```

**SAVE FILE!**

# Exercise: Upload the bears data bag item

```
$ knife data_bag from file apache_sites lions.json
```

```
Updated data_bag_item[apache_sites::lions]
```

# Exercise: Refactor apache::default recipe



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

```
# Iterate over the apache sites
all_sites = search("apache_sites", "*:*")
all_sites.each do |site|
  # Enable an Apache Virtualhost
  apache_vhost site['id'] do
    site_port site['port']
    action :create
    notifies :restart, "service[httpd]"
  end
end
```

- Delete existing node[“apache”][“sites”] loop
- Move apache\_vhost LWRP inside a search loop
- Change variable names

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0.3.0 ]  
Uploaded 1 cookbook.
```

# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
Starting Chef Client, version 11.10.4
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "ntp", "motd", "users", "apache"]
Synchronizing Cookbooks:
  - chef-client
  - cron
  - logrotate
  - ntp
  - motd
  - pci
  - users
  - apache
...
* apache_vhost[clowns] action createRecipe: <Dynamically Defined Resource>
..
* apache_vhost[bears] action createRecipe: <Dynamically Defined Resource>
..
* apache_vhost[lions] action createRecipe: <Dynamically Defined Resource>
..
```

```
Chef Client finished, 2/39 resources updated in 7.004059993 seconds
```

# Exercise: Extend :remove action



**OPEN IN EDITOR:** cookbooks/apache/providers/vhost.rb

```
# Add a template resource for the virtual host's index.html
template "#{document_root}/index.html" do
  source "index.html.erb"
  mode "0644"
  variables(
    :site_name => new_resource.site_name,
    :port => new_resource.site_port
  )
end
end

action :remove do
  file "/etc/httpd/conf.d/#{new_resource.site_name}.conf" do
    action :delete
  end
end
```

**SAVE FILE!**

# Exercise: Refactor apache::default recipe



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

```
# Disable the default virtual host
apache_vhost "welcome" do
  action :remove
  notifies :restart, "service[httpd]"
end

# Iterate over the apache sites
search("apache_sites", "*:*").each do |site|
  # Enable an Apache Virtualhost
  apache_vhost site['id'] do
```

- Delete existing **execute** resource that disables the welcome site
- Add a new **apache\_vhost** resource with **action :remove**

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0.3.0 ]
Uploaded 1 cookbook.
```

# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
Starting Chef Client, version 11.10.4
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "ntp", "motd", "users", "apache"]
Synchronizing Cookbooks:
- chef-client
- cron
- logrotate
- ntp
- motd
- pci
- users
- apache
..
* apache_vhost[welcome] action removeRecipe: <Dynamically Defined Resource>
* file[/etc/httpd/conf.d/welcome.conf] action delete (up to date)
(up to date)
..

Chef Client finished, 2/39 resources updated in 7.004059993 seconds
```

# Other ways to write resources & providers

- We wrote an LWRP using out-of-the-box Chef resources
- You can write pure Ruby in LWRPs as well
  - [http://docs.chef.io/lwrp\\_custom\\_provider\\_ruby.html](http://docs.chef.io/lwrp_custom_provider_ruby.html)
- You can also write pure Chef resources/providers by inheriting from `Chef::Resource::Base` and `Chef::Provider::Base` and putting them in libraries/

# Use Cases for LWRPs

- Hide unnecessary implementation details
- Example: app admin team decides how to create Apache virtual hosts, JBoss JVMs, etc.
- Write LWRPs to allow people to safely instantiate them
  - Parameter validation for correctness/conformance (e.g. “we don’t name JVMs with punctuation characters”)
  - Reject invalid values (e.g. “port -1”)
  - Restrict what tunables are available to consumers

# Review Questions

- What are the two components of an LWRP?
- What is the difference between these two components?
- What language can LWRPs written in?
- What classes can you inherit from Chef to write HWRPs?

# Writing Ohai Plugins

Getting more data from your systems

v1.1.6

# Lesson Objectives

- After completing the lesson, you will be able to
  - Explain the function Ohai and Ohai Plugins
  - Explain what's new in Ohai 7?
  - Write your own Ohai Plugins
  - Force Ohai Plugins to run
  - Disable Ohai Plugins

# Ohai!

- Ohai writes automatic attributes - they can't be changed by other components of Chef
- Automatic attributes form the base for every node object at the beginning of every Chef run
- Many plugins are enabled by default, but can be disabled if desired

# Running Ohai

```
chef@node1:~$ ohai
```

```
{  
  "languages": {  
    "ruby": {  
      "platform": "x86_64-darwin13.0.0",  
      "version": "1.9.3",  
      "release_date": "2013-11-22",  
      ...  
    }  
  }  
}
```

# Ohai Data Collection

- Platform details: `redhat`, `windows`, etc.
- Processor information
- Kernel data
- FQDNs
- Cloud provider data (EC2, Azure, Rackspace, etc)
- Windows device drivers
- and more...

# What's new in Ohai 7?

- New DSL
  - **collect\_data()** - block of Ruby code called by Ohai when it runs
  - **depends** - comma-separated list of attributes collected by another plugin
  - **provides** - comma-separated list of attributes defined by a plugin
- In Chef 11.12.0 or newer

# Check Ohai Version

```
chef@node1:~$ ohai -v
```

```
Ohai: 7.0.2
```

# Why Write Ohai Plugins?

- Collect specialized data not “in the box”
- Examples:
  - Information about some daemon you want to expose
  - Hardware inventory information from external sources (BMC, IPMI, Remedy ARS?)
  - Statistics collection on-the-cheap (e.g. shove netstat attributes into automatic data)

# Writing Ohai Plugins: Apache Modules

- **Problem:** We want to capture all installed Apache modules as part of our node data
- **Success:** Apache module data is visible in our node attributes

# Writing Ohai Plugins: apache.modules

```
chef@node1:~$ apachectl -t -D DUMP_MODULES
```

```
Loaded Modules:  
core_module (static)  
mpm_prefork_module (static)  
http_module (static)  
so_module (static)  
auth_basic_module (shared)  
auth_digest_module (shared)  
authn_file_module (shared)  
authn_alias_module (shared)  
...
```

# Installing Ohai Plugins

- Custom Ohai plugins are installed with the Ohai cookbook
  - Configures plugin path for Ohai
  - Delivers plugin to Ohai plugins directory with Chef
  - <http://supermarket.chef.io/cookbooks/ohai>
  - <http://docs.chef.io/ohai.html>

# Exercise: Download the Ohai Cookbook

```
$ knife cookbook site download ohai 2.0.1
```

Downloading ohai from the cookbooks site at  
version 2.0.1 to /Users/YOU/chef-repo/  
cookbooks/ohai-2.0.1.tar.gz

Cookbook saved: /Users/YOU/chef-repo/  
cookbooks/ohai-2.0.1.tar.gz

# Exercise: Download the Ohai Cookbook

```
$ tar -zxvf ohai-2.0.1.tar.gz -C cookbooks/
```

```
x ohai/
x ohai/CHANGELOG.md
x ohai/README.md
x ohai/attributes
x ohai/attributes/default.rb
x ohai/files
x ohai/files/default
x ohai/files/default/plugins
x ohai/files/default/plugins/README
x ohai/metadata.json
x ohai/metadata.rb
x ohai/providers
x ohai/providers/hint.rb
x ohai/recipes
x ohai/recipes/default.rb
x ohai/resources
x ohai/resources/hint.rb
```

# Exercise: Upload the Ohai Cookbook

```
$ knife cookbook upload ohai
```

```
Uploading ohai [2.0.1]
Uploaded 1 cookbook.
```

# Exercise: Inspect the Ohai Cookbook



**OPEN IN EDITOR:** cookbooks/ohai/attributes/default.rb

```
# FHS location would be /var/lib/chef/ohai_plugins or similar.
case node["platform_family"]
when "windows"
  default["ohai"]["plugin_path"] = "C:/chef/ohai_plugins"
else
  default["ohai"]["plugin_path"] = "/etc/chef/ohai_plugins"
end

# The list of plugins and their respective file locations
default["ohai"]["plugins"]["ohai"] = "plugins"
```

# Exercise: Update apache cookbook

- We could create our modules.rb file and add it to the ohai cookbook, but...
- We now have to maintain a forked version of the community cookbook
- Let's use our apache cookbook instead!

# Exercise: Update the metadata.rb



**OPEN IN EDITOR:** cookbooks/apache/metadata.rb

```
name                  'apache'
maintainer           'Your Name'
maintainer_email     'your_email@example.com'
license               'All rights reserved'
description          'Installs/Configures apache'
long_description    IO.read(File.join(File.dirname(__FILE__),  
'README.md'))
version              '0.4.0'
depends              'ohai'
```

**SAVE FILE!**

# Basic Ohai DSL

- **Ohai.plugin(:Name)** - used to identify the plugin

```
Ohai.plugin(:Name) do
  provides "attribute_a", "attribute_b"
  depends "attribute_x", "attribute_y"

  collect_data(:default) do
    # some Ruby code
    attribute_a Mash.new
  end

  collect_data(:platform...) do
    # platform-specific Ruby code
    attribute_b Mash.new
  end
end
```

# Basic Ohai DSL

- **Ohai.plugin(:Name)** - used to identify the plugin
- **provides** - comma-separated list of attributes defined by the plugin

```
Ohai.plugin(:Name) do
  provides "attribute_a", "attribute_b"
  depends "attribute_x", "attribute_y"

  collect_data(:default) do
    # some Ruby code
    attribute_a Mash.new
  end

  collect_data(:platform...) do
    # platform-specific Ruby code
    attribute_b Mash.new
  end
end
```

# Basic Ohai DSL

- **Ohai.plugin(:Name)** - used to identify the plugin
- **provides** - comma-separated list of attributes defined by the plugin
- **depends** - comma-separated list of attributes collected by another plugin

```
Ohai.plugin(:Name) do
  provides "attribute_a", "attribute_b"
  depends "attribute_x", "attribute_y"

  collect_data(:default) do
    # some Ruby code
    attribute_a Mash.new
  end

  collect_data(:platform...) do
    # platform-specific Ruby code
    attribute_b Mash.new
  end
end
```

# Basic Ohai DSL

- **Ohai.plugin(:Name)** - used to identify the plugin
- **provides** - comma-separated list of attributes defined by the plugin
- **depends** - comma-separated list of attributes collected by another plugin
- **collect\_data(:default)** - the default code run if the platform is not defined

```
Ohai.plugin(:Name) do
  provides "attribute_a", "attribute_b"
  depends "attribute_x", "attribute_y"

  collect_data(:default) do
    # some Ruby code
    attribute_a Mash.new
  end

  collect_data(:platform...) do
    # platform-specific Ruby code
    attribute_b Mash.new
  end
end
```

# Basic Ohai DSL

- **Ohai.plugin(:Name)** - used to identify the plugin
- **provides** - comma-separated list of attributes defined by the plugin
- **depends** - comma-separated list of attributes collected by another plugin
- **collect\_data(:default)** - the default code run if the platform is not defined
- **collect\_data(:platform)** - the code run for a specific platform

```
Ohai.plugin(:Name) do
  provides "attribute_a", "attribute_b"
  depends "attribute_x", "attribute_y"

  collect_data(:default) do
    # some Ruby code
    attribute_a Mash.new
  end

  collect_data(:platform...) do
    # platform-specific Ruby code
    attribute_b Mash.new
  end
end
```

# Exercise: Create modules.rb



**OPEN IN EDITOR:** apache/files/default/plugins/modules.rb

```
Ohai.plugin(:Apache) do
  require 'mixlib/shellout'
  provides "apache/modules"

  collect_data(:default) do
    modules = Mixlib::ShellOut.new("apachectl -t -D DUMP_MODULES")
    modules.run_command
    apache Mash.new
    apache[:modules] = modules.stdout
  end
end
```

**SAVE FILE!**

# Mixlib::Shellout

- Cross-platform library for safely executing shell commands from within Chef, Ohai, etc.
- Deals with passing arguments, capturing STDERR and STDOUT, timing-out hung processes
- Deals with cross-platform (Linux, UNIX, Windows) subprocess quirks
- **Never use backticks/Process.spawn/system!**
- **Use Mixlib::Shellout!**

# Exercise: Write the ohai\_plugin recipe



**OPEN IN EDITOR:** cookbooks/apache/recipes/ohai\_plugin.rb

```
ohai 'reload_apache' do
  plugin 'apache'
  action :nothing
end

cookbook_file "#{node['ohai']['plugin_path']}/modules.rb" do
  source 'plugins/modules.rb'
  owner 'root'
  group 'root'
  mode '0644'
  notifies :reload, 'ohai[reload_apache]', :immediately
end

include_recipe 'ohai::default'
```

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0 . 4 . 0 ]  
Uploaded 1 cookbook.
```

# Add the recipe to the web role



**OPEN IN EDITOR:** roles/web.rb

```
name "web"
description "Web Server"
run_list "role[base]", "recipe[apache::ohai_plugin]",
"recipe[apache]"
```

**SAVE FILE!**

# Upload the web role

```
$ knife role from file web.rb
```

Updated Role web!

# Exercise: Run chef-client

```
chef@node1:~$ sudo chef-client
```

```
Starting Chef Client, version 11.10.4.ohai7.0
resolving cookbooks for run list: ["apache::ohai_plugin"]
Synchronizing Cookbooks:
  - apache
  - ohai
Compiling Cookbooks...
Recipe: ohai::default
* remote_directory[/etc/chef/ohai_plugins] action create
  - create new directory /etc/chef/ohai_plugins
  - change mode from '' to '0755'
  - restore selinux security context
Recipe: <Dynamically Defined Resource>
```

# Exercise: Run chef-client

```
* cookbook_file[/etc/chef/ohai_plugins/README] action create
- create new file /etc/chef/ohai_plugins/README
- update content in file /etc/chef/ohai_plugins/README from none to 775fa7
  --- /etc/chef/ohai_plugins/README      2014-02-27 09:31:04.558032870 -0500
  +++ /tmp/.README20140227-33098-tpmpwc  2014-02-27 09:31:04.621065128
-0500
    @@ -1 +1,2 @@
    +This directory contains custom plugins for Ohai.
- change mode from '' to '0644'
- restore selinux security context

Recipe: ohai::default
* ohai[custom_plugins] action reload/opt/chef/embedded/lib/ruby/gems/1.9.1/
gems/ohai-7.0.0.rc.0/lib/ohai/plugins/linux/network.rb:49: warning: already
initialized constant IPROUTE_INT_REGEX

- re-run ohai and merge results into node attributes
```

# Exercise: Run chef-client

```
Converging 3 resources
Recipe: apache::ohai_plugin
* cookbook_file[/etc/chef/ohai_plugins/modules.rb] action create
- create new file /etc/chef/ohai_plugins/modules.rb
- update content in file /etc/chef/ohai_plugins/modules.rb from none to 2d1a51
  --- /etc/chef/ohai_plugins/modules.rb2014-02-27 09:31:09.667647850 -0500
  +++ /tmp/.modules.rb20140227-33098-savqid2014-02-27 09:31:09.718673962 -0500
  @@ -1 +1,9 @@
+Ohai.plugin(:Apache) do
+  provides "apache/modules"
+
+  collect_data(:default) do
+    modules = shell_out("apachectl -t -D DUMP_MODULES")
+    apache Mash.new
+    apache[:modules] = modules
+  end
+end
- change mode from '' to '0755'
- change owner from '' to 'root'
- change group from '' to 'root'
- restore selinux security context
```

# Exercise: Run chef-client

```
Recipe: ohai::default
  * ohai[custom_plugins] action reload/opt/chef/embedded/lib/ruby/gems/
1.9.1/gems/ohai-7.0.0.rc.0/lib/ohai/plugins/linux/network.rb:49: warning:
already initialized constant IPROUTE_INT_REGEX

  - re-run ohai and merge results into node attributes

  * remote_directory[/etc/chef/ohai_plugins] action nothing (skipped due to
action :nothing)
  * ohai[custom_plugins] action nothing (skipped due to action :nothing)

Running handlers:
Running handlers complete

Chef Client finished, 7/53 resources updated in 16.185836857 seconds
```

# Exercise: Show apache attribute

```
$ knife node show node1 -a apache
```

```
node1:  
  apache:  
    indexfile: index1.html  
    modules:   Loaded Modules:  
      core_module (static)  
      mpm_prefork_module (static)  
      http_module (static)  
      so_module (static)  
      auth_basic_module (shared)  
    ...
```

# Exercise: View plugin

```
chef@node1:~$ cat /etc/chef/ohai_plugins/modules.rb
```

```
Ohai.plugin(:Apache) do
  require 'mixlib/shellout'
  provides "apache/modules"

  collect_data(:default) do
    modules = Mixlib::ShellOut.new("apachectl -t -D DUMP_MODULES")
    modules.run_command
    apache Mash.new
    apache[:modules] = modules.stdout
  end
end
```

# Can We Make It Better?

- We have Apache module data!
- It's saved as one big string... that's a bummer
- Let's sort this into a group of static and shared modules

# Exercise: Refactor modules.rb



**OPEN IN EDITOR:** apache/files/default/plugins/modules.rb

```
Ohai.plugin(:Apache) do
  require 'mixlib/shellout'
  provides "apache/modules"

  collect_data(:default) do
    apache Mash.new
    apache[:modules] = { :static => [ ], :shared => [ ] }
```

- Let's start by setting up our node[ 'apache' ] Mash

# Exercise: Refactor modules.rb



**OPEN IN EDITOR:** apache/files/default/plugins/modules.rb

```
modules = shell_out("apachectl -t -D DUMP_MODULES")
modules.stdout.each_line do |line|
  fullkey, value = line.split(/\(/, 2).map { |i| i.strip}
  apache_mod = fullkey.gsub(/_module/, "")
  dso_type = value.to_s.chomp("\\")

  if dso_type.match(/shared/)
    apache[:modules][:shared] << apache_mod
  elsif dso_type.match(/static/)
    apache[:modules][:static] << apache_mod
  end
end
end
end
```

**SAVE FILE!**

# Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [ 0 . 4 . 0 ]  
Uploaded 1 cookbook.
```

# Exercise: Run chef-client

```
chef@node1:~$ sudo chef-client
```

```
Starting Chef Client, version 11.12.8
resolving cookbooks for run list: [ "apache::ohai_plugin" ]
Synchronizing Cookbooks:
  - apache
  - ohai
Compiling Cookbooks...
Recipe: apache::ohai_plugin
  * ohai[reload_apache] action nothing (skipped due to action :nothing)
  * cookbook_file[/etc/chef/ohai_plugins/modules.rb] action create
    - update content in file /etc/chef/ohai_plugins/modules.rb from e6cf9a to
809f46
      --- /etc/chef/ohai_plugins/modules.rb  2014-04-02 23:22:30.727649249
-0400
```

# Exercise: Show apache.modules attributes

```
$ knife node show node1 -a apache.modules
```

```
node1:  
  apache.modules:  
    shared:  
      auth_basic  
      auth_digest  
      authn_file  
      authn_alias  
      authn_anon  
...  
    static:  
      core  
      mpm_prefork  
      http  
      so
```

# Troubleshooting

```
$ sudo chef-client -l debug
```

```
WARN: Unresolved specs during Gem::Specification.reset:
      nokogiri (>= 1.4.0, ~> 1.5)
WARN: Clearing out unresolved specs.
Please report a bug if this causes problems.
[chef@ip-172-31-40-146 ~]$ head out.log
[2015-02-06T13:04:15+00:00] DEBUG: Sleeping for 0 seconds
[2015-02-06T13:04:15+00:00] INFO: Forking chef instance to converge...
[2015-02-06T13:04:15+00:00] DEBUG: Fork successful. Waiting for new chef pid: 14670
[2015-02-06T13:04:15+00:00] DEBUG: Forked instance now converging
[2015-02-06T13:04:15+00:00] INFO: *** Chef 12.0.3 ***
[2015-02-06T13:04:15+00:00] INFO: Chef-client pid: 14670
[2015-02-06T13:04:15+00:00] DEBUG: Chef-client request_id: 4f40289a-644c-4529-a50c-7e59b78aca37
[2015-02-06T13:04:16+00:00] DEBUG: Plugin Go threw #<Errno::ENOENT: No such file or directory -
go>
[2015-02-06T13:04:16+00:00] DEBUG: /opt/chefdk/embedded/lib/ruby/gems/2.1.0/gems/mixlib-
```

# Plugin Debugging Notes

- Ohai will **ignore plugin failures/exceptions**
- This is to avoid breaking Chef Client during such an early phase
- If you get no data from your plugins, this is probably why
- **Write plugins defensively. Check for all error conditions!**

# Ohai Hints

- Ohai plugins can be forced to run, even if detected system state says that they shouldn't!
- For example, its impossible to reliably detect whether a server is in EC2, so EC2 plugin may not run
- Solution: Force the EC2 plugin to run by placing an empty JSON document ("{}") in /etc/chef/ohai/hints/ec2.json

# Disabling Ohai Plugins

- Some plugins can execute slowly on different platforms
- You can disable plugins whose data you don't use to save speed things up!
- Only disables plugins for Ohai when run with chef-client

# The Problem & Success Criteria

- **The Problem:** Our systems are LDAP/Active Directory joined and thus the Ohai data contains thousands of user accounts, slowing down the Chef run.
- **Success Criteria:** We will disable the :Passwd plugin to avoid collecting this information

# Exercise: Disable :Passwd plugin

```
$ knife node show node1 -a etc.passwd
```

```
node1:  
etc.passwd:  
abrt:  
  dir:  /etc/abrt  
  gecos:  
  gid:  173  
  shell: /sbin/nologin  
  uid:  173  
adm:  
  dir:  /var/adm  
  gecos: adm  
  gid:  4  
  shell: /sbin/nologin  
  uid:  3  
...
```

# Update the base role



**OPEN IN EDITOR:** roles/base.rb

```
name "base"
description "Base Server Role"
run_list "recipe[chef-client::delete_validation]", "recipe[chef-client::config]", "recipe[chef-client]", "recipe[ntp]", "recipe[motd]", "recipe[users]"
default_attributes(
  "chef_client" => {
    "config" => {
      "ssl_verify_mode" => ":verify_peer"
      "log_level" => ":info"
    }
  },
  "ohai" => {
    "disabled_plugins" => [":Passwd"]
  }
)
```

# Exercise: Upload the base role

Updated Role base!

# Exercise: Upload the base role

```
$ knife role from file base.rb
```

Updated Role base!

# Exercise: Disable :Passwd plugin

```
chef@node1:~$ sudo chef-client && sudo chef-client
```

```
Starting Chef Client, version 11.12.4
resolving cookbooks for run list: [ "apache::ohai_plugin",
"chef-client::delete_validation", "chef-client", "ntp",
"motd", "users", "apache" ]
...
Running handlers:
Running handlers complete

Chef Client finished, 1/4 resources updated in 9.690084663
seconds
```

# Exercise: Disable :Passwd plugin

```
$ knife node show node1 -a etc.passwd
```

```
node1:  
etc.passwd:
```

# Exercise: Disable passwd plugin

```
chef@node1:~$ sudo cat /etc/chef/client.rb
```

```
chef_server_url "https://api.opscode.com/organizations/intermediate050614"
validation_client_name "intermediate050614-validator"
verify_api_cert true
ssl_verify_mode :verify_peer
node_name "node1"

Ohai::Config[:plugin_path] << "/etc/chef/ohai_plugins"
Ohai::Config[:disabled_plugins] = [:Passwd]

Dir.glob(File.join("/etc/chef", "client.d", "*.rb")).each do |conf|
  Chef::Config.from_file(conf)
end
```

# Other Tips

- Extreme situations: whitelist only the attributes you want to send: <http://ckbk.it/whitelist-node-attrs>
  - Minimize the amount of JSON you are sending across the wire on every run
  - Useful if you have many (>10,000) nodes
- Docs: <http://docs.chef.io/ohai.html>

# Special Upgrading Note

- The syntax for disabling Ohai plugins has changed slightly from 6 to 7
- **Ohai 6**
  - `Ohai::Config[ :disabled_plugins ] = [ "passwd" ]`
- **Ohai 7**
  - `Ohai::Config[ :disabled_plugins ] = [ :Passwd ]`

# Review Questions

- What command can you run at the command line to invoke Ohai?
- How are Ohai plugins installed?
- What is wrong with this first line of an Ohai definition: "Ohai.plugin(:name) do"?

# Chef Client Run Internals

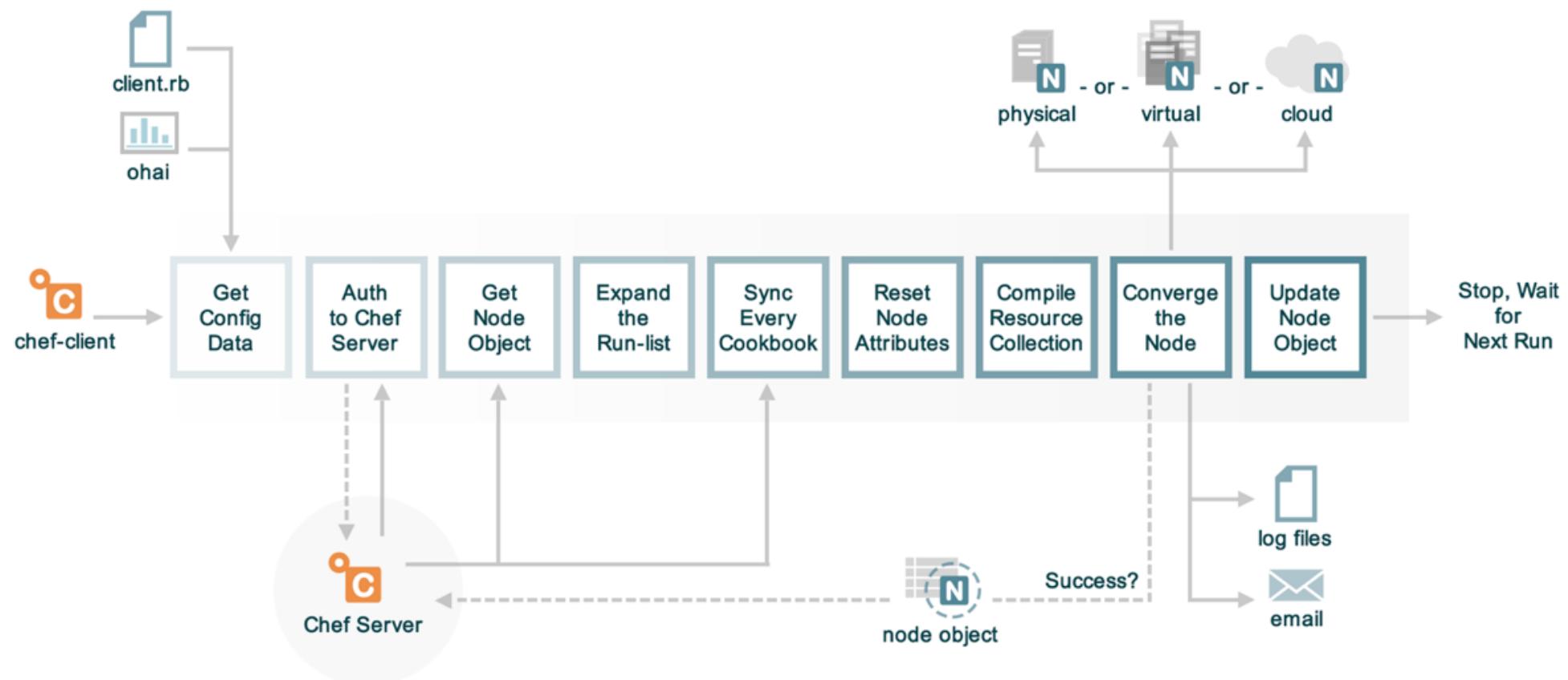
What really happens when Chef runs?

v1.1.6

# Lesson Objectives

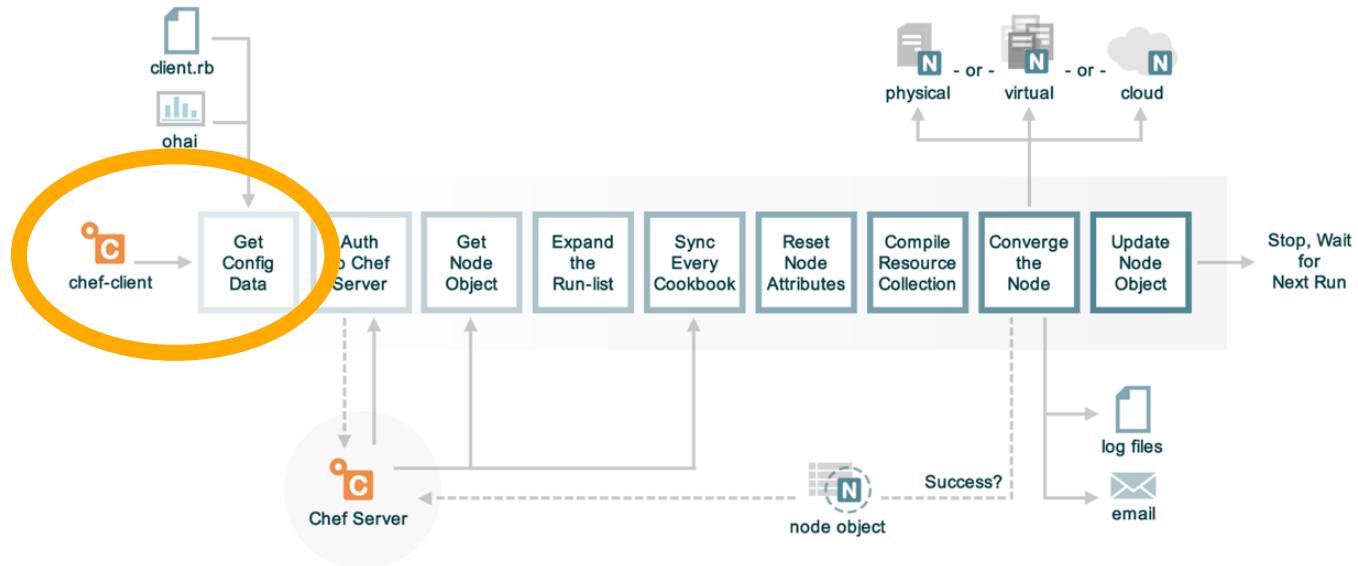
- After completing the lesson, you will be able to:
  - Understand what happens under the hood when Chef Client runs
  - Describe the purpose of the `run_context` and `run_status` objects
  - Use `node.run_state` to pass information around the recipe at execution time
  - Describe the event subsystem and how it can be used

# Remember This From Fundamentals?



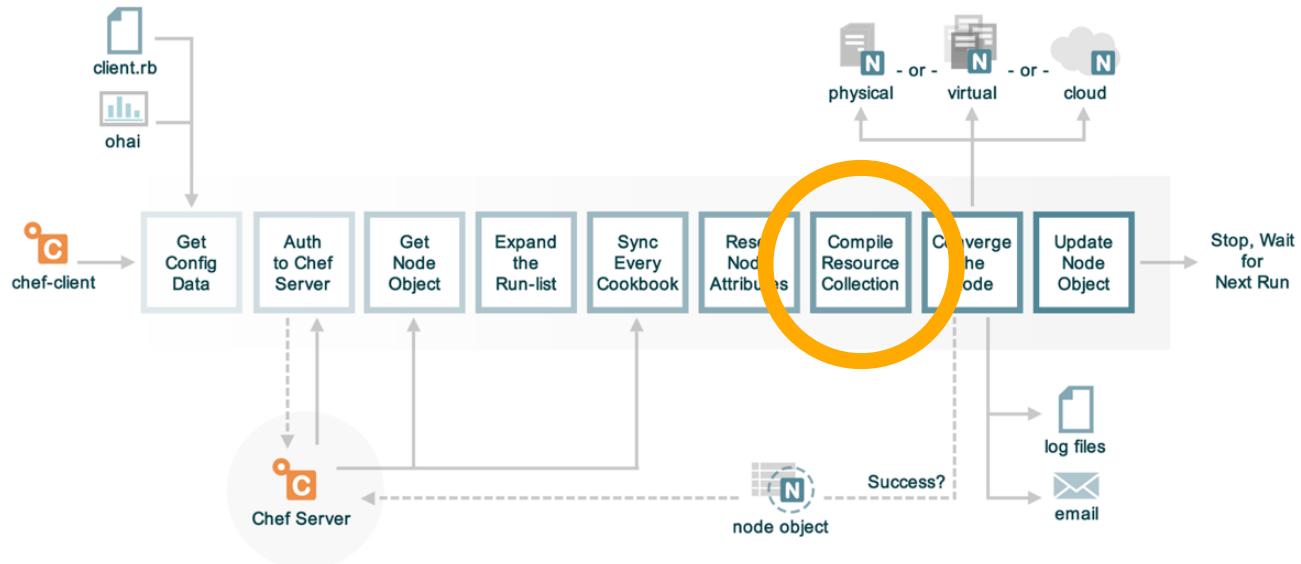
# Customizations

- Ohai Plugins
- Start Handlers
- Event Dispatcher

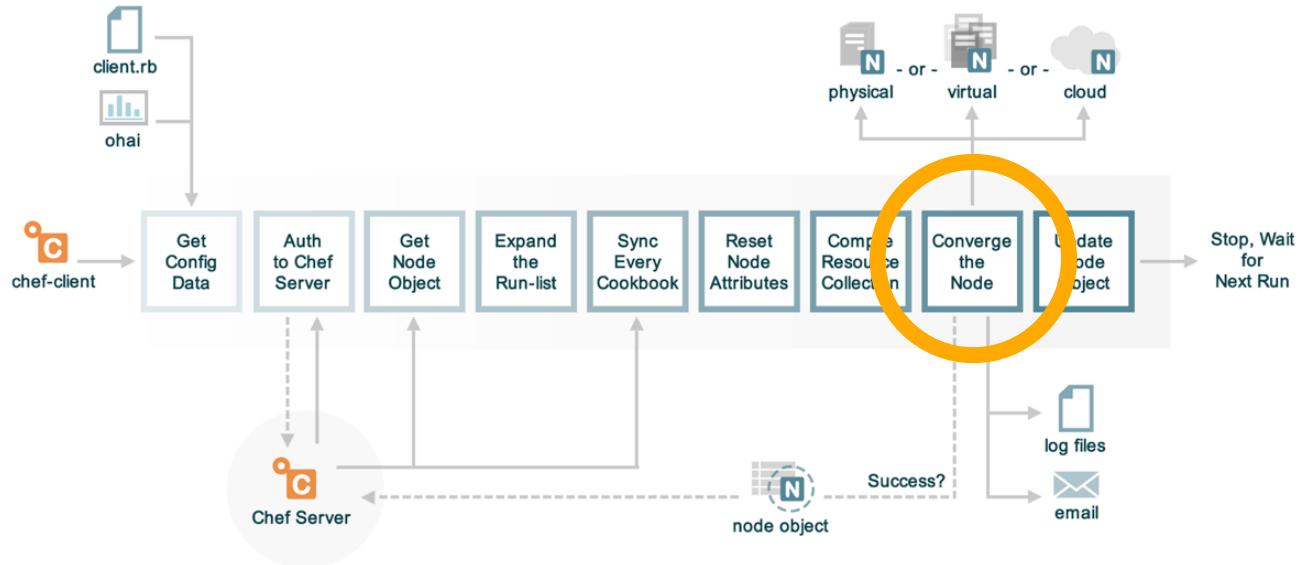


# Customizations

- LWRPs
- HWRPs
- Libraries



# Customizations



- Exception Handlers
- Report Handlers

# What is the run\_context?

- Master object for the Chef run for this node
- Important parts of the hierarchy:
  - `run_context`
    - `node`
      - `chef_environment`
      - `run_status`
      - `run_state`
    - `cookbook_collection`
    - `resource_collection`
    - `events`

# What is the run\_status?

- Hangs off node
- Stores current status of run
- Resources that are converged, resources that changed during this run
- Run status methods (success? / failed?)
- We'll use this object later when we write event handlers

# What is the run\_state?

- A Hash that hangs off node
- A place for you to stash transient data during the run
- Transmits data between resources



# The Problem and Success Criteria

- **Problem:** We can't decide what scripting language we want to use to write our web application.
- **Success Criteria:** Chef has randomly chosen one for us each time it runs.



or



Practical Extraction and Report Language

# Exercise: Edit Apache recipe



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

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

ruby_block "randomly_choose_language" do
  block do
    if Random.rand > 0.5
      node.run_state['scripting_language'] = 'php'
    else
      node.run_state['scripting_language'] = 'perl'
    end
  end
end

package "scripting_language" do
  package_name lazy { node.run_state['scripting_language'] }
  action :install
end
```

**SAVE FILE!**

# ruby\_block resource

```
ruby_block "randomly_choose_language" do
  block do
    if Random.rand > 0.5
      node.run_state['scripting_language'] = 'php'
    else
      node.run_state['scripting_language'] = 'perl'
    end
  end
end
```

- `ruby_block` declares a block of Ruby to run at **execute** time
- It has direct access to the `node` structure and other aspects of the run
- Store something on `node.run_state` for later use

# node.run\_state

```
node.run_state['scripting_language'] = 'php'
```

- `node.run_state` is an empty Hash
- Discarded at the end of the run

# Lazy parameter evaluation

```
package "scripting_language" do
  package_name lazy { node.run_state['scripting_language'] }
  action :install
end
```

- Normally resource parameters must be specified completely at **compile** time
- Sometimes their value is not known until **execute** time
- Use the `lazy{ }` block to have parameters evaluated then

# Upload the new apache cookbook

```
$ knife cookbook upload apache
```

```
Uploading apache [0.4.0]
Uploaded 1 cookbook.
```

# Run Chef Client

```
chef@node1$ sudo chef-client
```

- \* ruby\_block[randomly\_choose\_language] action run
  - execute the ruby block randomly\_choose\_language
- \* package[scripting\_language] action install
  - install version 5.3.3-27.el6\_5 of package php

# Event Dispatcher Subsystem

- Used by logging, Chef reporting
- Simple queue-free publish/subscribe model
- All listeners get all events
- Log formatters (:min, :doc, etc.) are examples of Event Dispatcher subclasses

# Implementing Event Dispatchers

- Write a subclass of  
`Chef::EventDispatch::Base`
  - Base methods are all set to take no action
  - Implement methods in subclass to be called
    - Examples: `run_start`, `run_completed`
  - Create a start handler to load your event dispatcher
  - Register your handler with Chef (next chapter)

# Review Questions

- What is the name of the root object of the Chef run?
- Where would you stash transient data during the run?
- What object stores the current status of the run?
- What can be used to populate resource attributes at execute time?

# Implementing Chef Handlers

Communicating the status of a Chef Client run

v1.1.6

# Lesson Objectives

- After completing this lesson, you will be able to
  - Describe Chef handlers
  - Write custom report handlers
  - Distribute handlers to nodes
  - Configure Chef to use custom handlers

# Handlers

- Handlers run at the start or end of a chef-client run
- They are *Ruby programs* that can read the status information of your chef-client run
- Three types:
  - Report
  - Exception
  - Start

# Report Handlers

- Used when a chef-client run succeeds
- Runs when the `run_status.success?` is `true`
- Community Examples:
  - [http://ampledata.org/splunk\\_storm\\_chef\\_handler.html](http://ampledata.org/splunk_storm_chef_handler.html)
  - <http://onddo.github.io/chef-handler-zookeeper/>
  - [https://github.com/realityforge/chef-graphite\\_handler](https://github.com/realityforge/chef-graphite_handler)

# Exception Handlers

- Used when a chef-client run fails
- Runs when the `run_status.failed?` is `true`
- Community Examples:
  - [https://github.com/morgoth/airbrake\\_handler](https://github.com/morgoth/airbrake_handler)
  - [https://github.com/jblaine/syslog\\_handler](https://github.com/jblaine/syslog_handler)
  - <http://onddo.github.io/chef-handler-sns/>

# Start Handlers

- Used to run events at the beginning of the chef-client run
- May **NOT** be loaded using the `chef_handler` resource
- Install with `chef_gem` or add to the `start_handlers` setting in the `client.rb`
- <https://github.com/chef/chef-reporting>

# Writing Custom Handlers

- Use any ruby gem or library
- Distribute to nodes with the **chef\_handler** or **chef-client** cookbooks
- Optionally configure through **client.rb** settings

# Exercise: Download the `chef_handler` cookbook

```
$ knife cookbook site download chef_handler 1.1.6
```

Downloading `chef_handler` from the cookbooks site  
at version 1.1.6 to `/Users/YOU/chef-repo/  
chef_handler-1.1.6.tar.gz`

Cookbook saved: `/Users/YOU/chef-repo/  
chef_handler-1.1.6.tar.gz`

# Exercise: Download the chef-client cookbook

```
$ tar -zxvf chef_handler-1.1.6.tar.gz -C cookbooks/
```

```
x chef_handler/
x chef_handler/CHANGELOG.md
x chef_handler/README.md
x chef_handler/attributes
x chef_handler/attributes/default.rb
x chef_handler/files
x chef_handler/files/default
x chef_handler/files/default/handlers
x chef_handler/files/default/handlers/README
x chef_handler/libraries
x chef_handler/libraries/matchers.rb
```

# Exercise: Upload the chef\_handler cookbook

```
$ knife cookbook upload chef_handler
```

```
Uploading chef_handler [1.1.6]
Uploaded 1 cookbook.
```

# The Problem and Success Criteria

- **Problem:** When a chef-client run ends we want to be notified via email of only the resources that have changed on the node.
- **Success Criteria:** We receive an email containing the resource changed on the node.

# Let's Write a Handler

- We're going to write a new handler that will **email** us when chef-client runs
- It will send us all of the resources that changed during the chef-client run
- <http://docs.chef.io/handlers.html>

# Dear Ruby, How Do You Email?

- Handlers are *Ruby programs* that can read the status information of your chef-client run.
- We could write a library to send an email or we could look for one:
  - Ruby's Standard Library
  - Ruby's Extended Library
  - Ruby gems

# Dear Ruby, How Do You Email?

- <http://www.rubydoc.info/stdlib/core/1.9.3>
- <http://www.rubydoc.info/stdlib>
- <https://www.rubygems.org/>
- [https://www.ruby-toolbox.com/categories/e\\_mail](https://www.ruby-toolbox.com/categories/e_mail)

# Library Cookbook Pattern

- We could modify the **chef\_handler** cookbook to add a recipe for the email handler
- However, we'd basically be “**forking**” an upstream cookbook
- Instead, let's create our own cookbook and use **chef\_handler** as a “**library**”
- Code reusability!

# Exercise: Create a cookbook named ‘email\_handler’

```
$ chef generate cookbook cookbooks/email_handler
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/Users/nathenharvey/chef-repo/cookbooks/email_handler] action create
    - create new directory /Users/nathenharvey/chef-repo/cookbooks/email_handler
  * template[/Users/nathenharvey/chef-repo/cookbooks/email_handler/metadata.rb] action
create_if_missing
    - create new file /Users/nathenharvey/chef-repo/cookbooks/email_handler/metadata.rb
    - update content in file /Users/nathenharvey/chef-repo/cookbooks/email_handler/metadata.rb
from none to ff8ffb
      (diff output suppressed by config)
  * template[/Users/nathenharvey/chef-repo/cookbooks/email_handler/README.md] action
create_if_missing
    - create new file /Users/nathenharvey/chef-repo/cookbooks/email_handler/README.md
    - update content in file /Users/nathenharvey/chef-repo/cookbooks/email_handler/README.md from
none to 2f2837
```

# Exercise: Edit the default recipe



**OPEN IN EDITOR:** cookbooks/email\_handler/recipes/default.rb

```
chef_gem "pony" do
  action :install
end

include_recipe "chef_handler"
```

**SAVE FILE!**

# The `chef_handler` Resource

- `chef_handler` is a resource packaged with the `chef_handler` cookbook
- It has two actions, `:enable` and `:disable`
- It has three arguments
  - `source` - the file containing the handler code
  - `arguments` - any pieces of information needed to initialize the handler
  - `supports` - `:report`, `:exception`
- Defaults:
  - `:enable`
  - `:report => true`, `:exception => true`

# Exercise: Setup the handler



**OPEN IN EDITOR:** cookbooks/email\_handler/recipes/default.rb

```
chef_gem "pony" do
  action :install
end

include_recipe "chef_handler"

cookbook_file "#{node['chef_handler']['handler_path']}/email_handler.rb" do
  source "handlers/email_handler.rb"
  owner "root"
  group "root"
  mode "0644"
end
```

**SAVE FILE!**

# Exercise: Setup the handler



**OPEN IN EDITOR:** cookbooks/email\_handler/recipes/default.rb

```
cookbook_file "#{node['chef_handler']['handler_path']}/email_handler.rb" do
  source "handlers/email_handler.rb"
  owner "root"
  group "root"
  mode "0644"
end
```

```
chef_handler "MyCompany::EmailMe" do
  source "#{node['chef_handler']['handler_path']}/email_handler.rb"
  arguments [node['email_handler']['from_address'],
             node['email_handler']['to_address']]
  action :enable
end
```

**SAVE FILE!**

# Exercise: create attributes

```
$ chef generate attribute cookbooks/email_handler default
```

```
Compiling Cookbooks...
Recipe: code_generator::attribute
 * directory[cookbooks/email_handler/attributes] action create
   - create new directory cookbooks/email_handler/attributes
 * template[cookbooks/email_handler/attributes/default.rb] action
create
   - create new file cookbooks/email_handler/attributes/default.rb
   - update content in file cookbooks/email_handler/attributes/
default.rb from none to e3b0c4
 (diff output suppressed by config)
```

# Exercise: Set the attributes



**OPEN IN EDITOR:** cookbooks/email\_handler/attributes/default.rb

```
default['email_handler']['from_address'] = "chef@localhost"
default['email_handler']['to_address'] = "chef@localhost"
```

**SAVE FILE!**

# Exercise: Write the handler



**OPEN IN EDITOR:** ../email\_handler/files/default/handlers/email\_handler.rb

```
require 'rubygems'  
require 'pony'  
  
module MyCompany  
  class EmailMe < Chef::Handler
```

**SAVE FILE!**

- All custom exception and report handlers are defined using Ruby and must be a subclass of the **Chef::Handler** class.
- The module and class match what we defined as the name of the `chef_handler` in the recipe

# The initialize Method



**OPEN IN EDITOR:** ../email\_handler/files/default/handlers/email\_handler.rb

```
class EmailMe < Chef::Handler

  def initialize(from_address, to_address)
    @from_address = from_address
    @to_address = to_address
  end
```

**SAVE FILE!**

- Initialize the handler with the arguments we passed in the definition
- You can create the method with any args you need to meet your requirements

# The report Method



**OPEN IN EDITOR:** ../email\_handler/files/default/handlers/email\_handler.rb

```
@to_address = to_address
end

def report
  status = "Failed"
  if success?
    status = "Successful"
  end
  subject = "#{status} Chef run report from #{node.name}"
  report_string = ""      SAVE FILE!
```

- The **report** interface is used to define how a handler will behave and is a required part of any custom handler

# The updated\_resources Hash



**OPEN IN EDITOR:** `../email_handler/files/default/handlers/email_handler.rb`

```
# report on changed resources
if ! run_status.updated_resources.empty?
  # get some info about all the changed resources!
  run_status.updated_resources.each do |r|
    report_string += "The resource #{r.name} was changed in cookbook
#{r.cookbook_name} at #{r.source_line}\n"
  end
else
  report_string += "No resources changed by chef-client\n"
end
```

**SAVE FILE!**

- **updated\_resources** records information about all the resources changed during a chef-client run
- read through this hash with `.each`, pull interesting information out about each resource

# Exercise: Finish email\_handler.rb



**OPEN IN EDITOR:** ../email\_handler/files/default/handlers/email\_handler.rb

```
    report_string += "No resources changed by chef-client\n"
end

Pony.mail(:to => @to_address,
           :from => @from_address,
           :subject => subject,
           :body => report_string)
end
end
end
```

**SAVE FILE!**

- Use **Pony.mail** to send a message containing info on changed resources

# Other Dependencies

- Make sure all necessary functions are available!
  - Some sort of mail transfer agent (MTA)
  - Some sort of email reader (MUA)
- These are distinct functional pieces
  - May have uses other than the handler
  - Get to be their own cookbooks!

# Exercise: Download the postfix cookbook

```
$ knife cookbook site download postfix 3.6.2
```

Downloading **postfix** from the cookbooks site  
at version 3.6.2 to /Users/YOU/chef-repo/  
**postfix-3.6.2.tar.gz**

Cookbook saved: /Users/YOU/chef-repo/  
**postfix-3.6.2.tar.gz**

# Exercise: Download the postfix cookbook

```
$ tar -zxvf postfix-3.6.2.tar.gz -C cookbooks/
```

```
x postfix/
x postfix/CHANGELOG.md
x postfix/README.md
x postfix/attributes
x postfix/attributes/default.rb
x postfix/files
x postfix/files/default
x postfix/files/default/tests
x postfix/files/default/tests/minitest
x postfix/files/default/tests/minitest/support
```

# Exercise: Upload the postfix cookbook

```
$ knife cookbook upload postfix
```

Uploading postfix  
upload complete

[ 3 . 6 . 2 ]

# MUA: mailutils

- There isn't a community cookbook for **mailx**, a command line mail reader
- On Debian-family systems, **mailx** is included with the **mailutils** package
- On Red Hat-family systems, **mailx** is included by the **mailx** package

# Exercise: Create the mailx cookbook

```
$ chef generate cookbook cookbooks/mailx
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
* directory[/Users/nathenharvey/chef-repo/cookbooks/mailx] action create
  - create new directory /Users/nathenharvey/chef-repo/cookbooks/mailx
* template[/Users/nathenharvey/chef-repo/cookbooks/mailx/metadata.rb] action create_if_missing
  - create new file /Users/nathenharvey/chef-repo/cookbooks/mailx/metadata.rb
  - update content in file /Users/nathenharvey/chef-repo/cookbooks/mailx/metadata.rb from none to
6bf99d
  (diff output suppressed by config)
* template[/Users/nathenharvey/chef-repo/cookbooks/mailx/README.md] action create_if_missing
  - create new file /Users/nathenharvey/chef-repo/cookbooks/mailx/README.md
  - update content in file /Users/nathenharvey/chef-repo/cookbooks/mailx/README.md from none to
c721a1
  (diff output suppressed by config)
* cookbook_file[/Users/nathenharvey/chef-repo/cookbooks/mailx/chefignore] action create
  - create new file /Users/nathenharvey/chef-repo/cookbooks/mailx/chefignore
```

# Exercise: Install the package



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

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

**SAVE FILE!**

# Exercise: Add the mail dependencies



**OPEN IN EDITOR:** cookbooks/email\_handler/recipes/default.rb

```
chef_gem "pony" do
  action :install
end

include_recipe "chef handler"
include_recipe "postfix"
include_recipe "mailx"

cookbook_file "#{node['chef_handler']['handler_path']}/
email_handler.rb" do
  source "handlers/email_handler.rb"
  owner "root"
```

**SAVE FILE!**

# Exercise: Update the metadata.rb



**OPEN IN EDITOR:** cookbooks/email\_handler/metadata.rb

```
name          "email handler"
maintainer    "You"
maintainer_email "you@somewhere.com"
license        "Apache 2.0"
description    "Email me on every Chef run"
long_description "Email me on every Chef run"
version        "0.1.0"

depends "chef_handler"
depends "postfix"
depends "mailx"
```

**SAVE FILE!**

# Exercise: Upload the email\_handler cookbook

```
$ knife cookbook upload email_handler  
postfix mailx
```

```
Uploading email_handler [0.1.0]  
Uploading postfix [3.6.2]  
Uploading mailx [0.1.0]  
Uploaded 3 cookbooks.
```

# Exercise: Add email\_handler to base role



**OPEN IN EDITOR:** roles/base.rb

```
name "base"
description "Base Server Role"
run_list "recipe[email_handler]", "recipe[chef-client::delete_validation]",
"recipe[chef-client::config]", "recipe[chef-client]", "recipe[ntp]",
"recipe[motd]", "recipe[users]"
```

**SAVE FILE!**

252

## Best Practice: Add handlers at beginning of run

- Set up handlers (especially exception handlers) at the beginning of the `run_list`
- That way, if the run fails, the handler will still execute

# Exercise: Upload the base role

```
$ knife role from file base.rb
```

Updated Role base!

# Exercise: Run chef-client

```
chef@node1$ sudo chef-client
```

```
...
* chef_handler[MyCompany::EmailMe] action enable
  - load /var/chef/handlers/email_handler.rb
  - enable chef_handler[MyCompany::EmailMe] as a
report handler
    - enable chef_handler[MyCompany::EmailMe] as a
exception handler
...
...
```

# Read the Message Using "mail"

```
chef@node1$ mail
```

```
Heirloom Mail version 12.4 7/29/08. Type ? for help.  
"/var/spool/mail/chef": 1 message 1 new  
>N 1 pony@unknown Wed May 14 09:14 30/2412 "Successful Chef run report from node1"  
& r  
To: chef@localhost pony@unknown  
Subject: Re: Successful Chef run report from node1  
  
pony@unknown wrote:  
  
> The resource pony was changed in cookbook email_handler at /var/chef/cache/cookbooks/  
email_handler/recipes/default.rb:9:in `from_file'  
> The resource /var/chef/handlers was changed in cookbook chef_handler at /var/chef/cache/  
cookbooks/chef_handler/recipes/default.rb:23:in `from_file'  
> The resource /etc/postfix/main.cf was changed in cookbook postfix at /var/chef/cache/  
cookbooks/postfix/recipes/default.rb:95:in `block in from_file'  
> The resource /etc/postfix/master.cf was changed in cookbook postfix at /var/chef/cache/  
cookbooks/postfix/recipes/default.rb:95:in `block in from_file'
```

# Review Questions

- What cookbook is used to configure Chef Handlers?
- What are the three types of Chef Handler?
- When are Start Handlers run?
- When are Report Handlers run?
- When are Exception Handlers run?

# Cookbook Style & Correctness

The real benefits of Infrastructure As Code

v1.1.6

# Lesson Objectives

- After completing this lesson, you will be able to:
  - Explain the benefits of correctness checking your cookbooks
  - Explain how and why you should test your cookbooks
  - Use Foodcritic to check the validity of your cookbooks
  - Verify your code adheres to the community Ruby style guide by using Rubocop

# Devops is a Two-Way Street

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



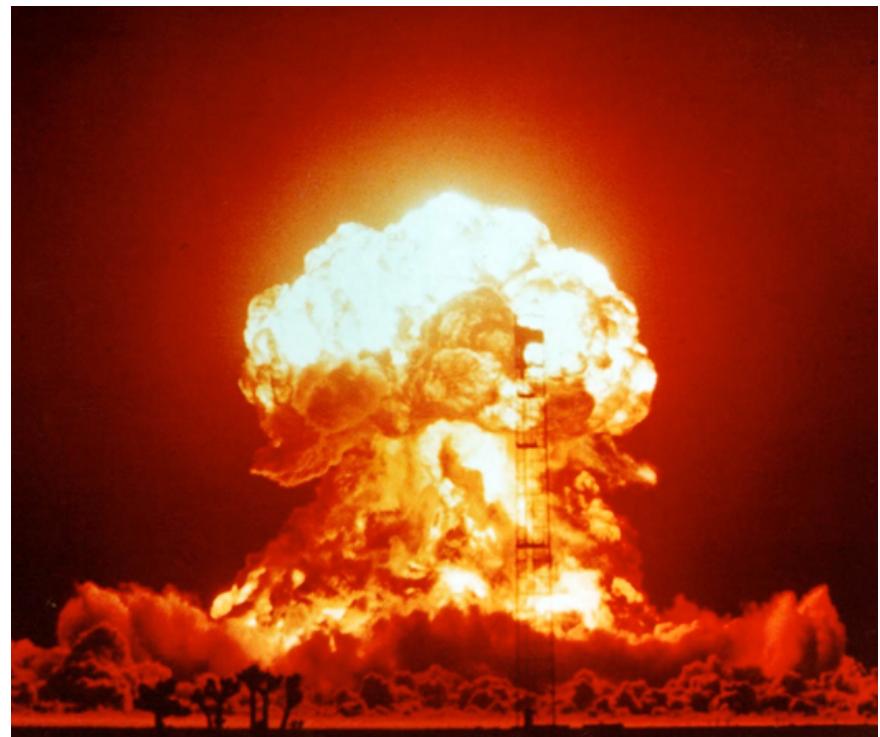
# Devops is a Two-Way Street



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

# Old Software Development Workflow

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



# New Software Development Workflow

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



# Old Chef Cookbook Workflow

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



# New Chef Cookbook Workflow

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



# Tools of the Trade

- **Code correctness:** Foodcritic, Rubocop
- **Unit tests:** ChefSpec
- **Integration tests** (not covered in this class): Test Kitchen, ServerSpec, BATS

# Foodcritic

v1.0.0

# What is Foodcritic?

- A correctness-checking tool for cookbooks
- Community best practices for cookbook style
- Ability to write and use custom rules
- <http://foodcritic.io/>



# Foodcritic

- A **code linting** tool for Chef
  - Checks **code correctness**
  - **Catches common mistakes** before they cause problems
  - Extensible & Configurable
    - Create new **rulesets**
    - Ignore existing rulesets
  - Integrates nicely into **CI pipelines**

# Rules and Tags

- All rules are numbered, e.g.
  - *FC002 - "Avoid string interpolation where not required"*
  - *FC034 - "Unused template variables"*
- Rules can be categorized and tagged, e.g.
  - 'style'
  - 'portability'
  - ...

## FC002: Avoid string interpolation where not required

### style strings

When you declare a resource in your recipes you frequently want to reference dynamic values such as node attributes. This warning will be shown if you are unnecessarily wrapping an attribute reference in a string.

#### Unnecessary string interpolation

This example would match the FC002 rule because the `version` attribute has been unnecessarily quoted.

```
# Don't do this
package "mysql-server" do
  version "#{node['mysql']['version']}"
  action :install
end
```

#### Modified version

This modified example would not match the FC002 rule:

```
package "mysql-server" do
  version node['mysql']['version']
  action :install
end
```

# Abide by best practices

- There are 50+ rules in Foodcritic; your cookbooks should pass them (or have a good reason for not)
- You can write your own rules
- Extra community-contributed rules:  
<http://www.foodcritic.io/#extra-rules>

# The Problem and the Success Criteria

- **The Problem:** We want to make sure our Apache cookbook follows best practice for Chef code.
- **Success Criteria:** We use Foodcritic to check our cookbook before committing it.

## Exercise: Check cookbook correctness with Foodcritic

```
$ foodcritic cookbooks/apache
```

```
FC003: Check whether you are running with chef server before using server-specific  
features: cookbooks/apache/recipes/default.rb:22  
FC003: Check whether you are running with chef server before using server-specific  
features: cookbooks/apache/recipes/ip-logger.rb:1  
FC008: Generated cookbook metadata needs updating: cookbooks/apache/metadata.rb:2  
FC008: Generated cookbook metadata needs updating: cookbooks/apache/metadata.rb:3  
FC016: LWRP does not declare a default action: cookbooks/apache/resources/vhost.rb:1
```

# Foodcritic: Ignore some rules



**OPEN IN EDITOR:** cookbooks/apache/.foodcritic

~FC003

~FC009

**SAVE FILE!**

274

# Exercise: Run Foodcritic

```
$ foodcritic cookbooks/apache
```

```
FC008: Generated cookbook metadata needs updating: cookbooks/apache/metadata.rb:2
FC008: Generated cookbook metadata needs updating: cookbooks/apache/metadata.rb:3
FC016: LWRP does not declare a default action: cookbooks/apache/resources/vhost.rb:1
```

# Exercise: Fix the cookbook to clear the errors

```
$ foodcritic cookbooks/apache
```

```
(no output)
```

- Edit your cookbook to fix the **FC008 & FC016** errors, then rerun your Foodcritic
- Refer to <http://www.foodcritic.io/> to find the errors
- No output means no errors!

# Running certain tests

- Run only certain categories of test

```
$ foodcritic <path> -t style
```

- Run specific tests

```
$ foodcritic <path> -t FC034
```

- Exclude specific tests

```
$ foodcritic <path> -t ~FC034
```

## Best Practice: Run Foodcritic Before Each Commit

- Always **check** in correct code!
- Make **Foodcritic** a part of your **build pipeline** with commit hooks or other methods
- The Foodcritic web page has extensive documentation & examples showing how to build Foodcritic rules into Jenkins or Travis-CI

# Rubocop

v1.0.0

# Rubocop

- A robust Ruby static **code analyzer**, based on the community Ruby **style guide**
- Looks at cookbooks for **Ruby** best practices, not the **Chef DSL** - that's for Foodcritic!
- Similar to Foodcritic there are already lots of rules defined: <https://github.com/bbatsov/rubocop/blob/master/config/enabled.yml>

# The Problem and the Success Criteria

- **The Problem:** We want to make sure our Apache cookbook follows best practice for Ruby code.
- **Success Criteria:** We use Rubocop to check our cookbook before committing it.

# Exercise: Run Rubocop on Parent Directory

```
$ rubocop cookbooks
```

```
warning: parser/current is loading parser/ruby21, which recognizes
warning: 2.1.5-compliant syntax, but you are running 2.1.4.
Inspecting 144 files
CCCCCCCCWCCC...CCCCWCC.WCCCCCCCCWCCCWCCCC.CCCCCCCC.cc.cc....cc.cccccccc.ccc.cccc.cc.ccc.cc.cccc
CCC.cc.cc..c.WCWCCCCCCCCC.C.CCCCWCCC.CCCC.CCCCC

Offenses:

cookbooks/apache/attributes/default.rb:1:9: C: Prefer single-quoted strings when you don't need
string interpolation or special symbols.
default["apache"]["indexfile"] = "index1.html"
^~~~~~
cookbooks/apache/attributes/default.rb:1:19: C: Prefer single-quoted strings when you don't need
string interpolation or special symbols.
default["apache"]["indexfile"] = "index1.html"
```

# Exercise: Run Rubocop on Apache

```
$ rubocop cookbooks/apache
```

```
warning: parser/current is loading parser/ruby21, which recognizes
warning: 2.1.5-compliant syntax, but you are running 2.1.4.
```

```
Inspecting 10 files
CCCCCCCCCW
```

```
Offenses:
```

```
cookbooks/apache/Berksfile:1:8: C: Prefer single-quoted strings when you don't need
string interpolation or special symbols.
```

```
source "https://supermarket.getchef.com"
^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
```

```
cookbooks/apache/metadata.rb:1:5: C: Put one space between the method name and the
first argument.
```

```
name          'apache'
```

# Understanding the Output

```
Inspecting 10 files
CCCCCCCCCCW
```

Offenses:

- Each character represents a file checked
  - A dot represents a clean file
  - A capital letter means an offense - could be **C**onvention, **W**arning, **E**rror or **F**atal
- All errors are then listed in detail beneath this summary

# Create a boilerplate .rubocop.yml



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

```
AlignParameters:
```

```
  Enabled: false
```

```
Encoding:
```

```
  Enabled: false
```

```
LineLength:
```

```
  Max: 200
```

```
StringLiterals:
```

```
  Enabled: false
```

- Turn off common, annoying warnings

**SAVE FILE!**

# Exercise: Run Rubocop

```
$ rubocop cookbooks/apache
```

```
Inspecting 10 files
.CCCCCCW
```

```
Offenses:
```

```
cookbooks/apache/metadata.rb:1:5: C: Put one space between the
method name and the first argument.
```

```
name      'apache'
^^^^^^^^^
```

```
10 files inspected, 32 offenses detected
```

# Exercise: Make corrections in offending file



**OPEN IN EDITOR:** apache/files/default/plugins/modules.rb

```
...
modules.stdout.each_line do |line|
  fullkey, value = line.split(/\(/, 2).map { |i| i.strip }
  apache_mod = fullkey.gsub(/_module/, "")
...
...
```

**SAVE FILE!**

# Exercise: Run Rubocop

```
$ rubocop cookbooks/apache
```

```
Inspecting 10 files
```

```
.C.CCCCCCW
```

```
Offenses:
```

```
10 files inspected, 30 offenses detected
```

# Exercise: Rubocop workflow

```
$ cd cookbooks/apache
```

# Exercise: Rubocop workflow

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

```
...
cookbooks/motd/recipes/default.rb:11:9: C: Prefer single-quoted strings when you don't need string
interpolation or special symbols.
  mode "0644"
    ^^^^^^
cookbooks/motd/recipes/default.rb:12:2: W: end at 12, 1 is not aligned with template "/etc/motd" do at
9, 0
end
^^^

8 files inspected, 74 offences detected
Created rubocop-todo.yml.
Run rubocop with --config rubocop-todo.yml, or
add inherit_from: rubocop-todo.yml in a .rubocop.yml file.
```

# Exercise: Rubocop Workflow

```
$ echo "inherit_from: .rubocop_todo.yml" >> .rubocop.yml
```

# Review Questions

- Why should you check for correctness in your cookbooks?
- What is the difference between Foodcritic and Rubocop?
- At what point in the development lifecycle would you use Foodcritic and Rubocop?

# An Introduction to ChefSpec

Unit testing your cookbooks to prevent regressions

v1.1.6

# Lesson Objectives

- After completing the lesson, you will be able to:
  - Understand what unit testing means for Chef cookbooks and recipes
  - Understand why to write unit tests for Chef recipes
  - Use ChefSpec to create and manage a test suite for your cookbooks

# Why Write Unit Tests?

- Fixing bugs before deploying code is cheap
- Fixing them afterwards is expensive
  - Programmer cost
  - Operational cost (bugs cause outages)
- Unit tests assert your intended behavior
- Unit tests run quickly
- If you need to refactor your cookbook, tests ensure you don't break anything unless you meant to

# Problem Statement

- **Problem:** We broke our motd cookbook one too many times
- **Proposed Solution:** Use ChefSpec to write tests to ensure the code is valid

# Introduction to ChefSpec Syntax

- ChefSpec is built on-top of RSpec
  - The standard Ruby testing tool
- RSpec has a familiar, English-like syntax, and so does ChefSpec!
  - `context` - group related tests together
  - `describe` - a test
  - `it` - a block describing some behavior
  - `expect` - expectations to assert

# General Test Approach

- Set up the test
  - Make a Chef run in memory
  - Set up test harness if necessary
- Make some assertions (expectations)

# General Test Format for ChefSpec

```
require 'spec_helper'

describe 'the_cookbook::default' do
  chef_run = ChefSpec::Runner.new.converge(described_recipe)
  it 'does something' do
    expect(chef_run).to some_condition
  end
end
```

- `require 'chefspec'` - load ChefSpec
- `describe` - the thing under test
- `chef_run` - create a Chef run in memory & converge it
  - `described_recipe` is syntactic sugar for '`the_cookbook::default`'
- `it` block - make some assertions (expectations)
  - `some_condition` - known as a "matcher"

# Exercise: Make directory for unit tests

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

( No output )

# Exercise: Create a spec helper



**OPEN IN EDITOR:** cookbooks/motd/spec/spec\_helper.rb

```
require 'chefspec'

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

**SAVE FILE!**

- By convention, test suites have a “helper”
- Avoid restating `require 'chefspec'` over and over
- Can configure RSpec in here (formatting, enforced style, etc.)

# Exercise: Create a skeleton test



**OPEN IN EDITOR:** cookbooks/motd/spec/unit/default\_spec.rb

```
require 'spec_helper.rb'

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

  it 'does something' do
    skip 'need to write this test'
  end
end
```

**SAVE FILE!**

- Test file name should match recipe name (`default_`), and end in `_spec.rb`
- `describe` is always the cookbook name and recipe name
- `skip` - special syntax to tell RSpec that you know you need to do some work yet

# Exercise: Run rspec from the cookbook

```
$ cd cookbooks/motd  
$ rspec spec
```

```
*
```

```
Pending:
```

```
motd::default does something  
# need to write this test  
# ./cookbooks/motd/spec/unit/default_spec.rb:6
```

```
Finished in 0.00053 seconds (files took 3.23 seconds to load)  
1 example, 0 failures, 1 pending
```

```
No Chef resources found, skipping coverage calculation...
```

# Exercise: Write a real test



**OPEN IN EDITOR:** cookbooks/motd/spec/unit/default\_spec.rb

```
require 'spec_helper.rb'

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

  it 'creates an motd correctly' do
    expect(chef_run).to create_template('/etc/motd').with(
      :user => 'root',
      :group => 'root',
      :mode => '0644'
    )
  end
end
```

**SAVE FILE!**

# Exercise: Run rspec

```
$ rspec spec
```

```
Failures:
```

```
1) motd::default creates an motd correctly
Failure/Error: expect(chef_run).to create_template('/etc/motd').with(
  expected "template[/etc/motd]" to have parameters:

    user "root", was nil
    group "root", was nil
# ./spec/unit/default_spec.rb:7:in `block (2 levels) in <top (required)>'
```

```
Finished in 0.03019 seconds
```

```
1 example, 1 failure
```

```
...
```

# Exercise: Fix original recipe



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

```
template "/etc/motd" do
  source "motd.erb"
  mode "0644"
  owner "root"
  group "root"
end
```

- Add owner and group so the test passes.

**SAVE FILE!**

# Exercise: Run rspec again

```
.
```

```
Finished in 0.02726 seconds
```

```
1 example, 0 failures
```

```
ChefSpec Coverage report generated...
```

```
Total Resources: 1
```

```
Touched Resources: 1
```

```
Touch Coverage: 100.0%
```

```
You are awesome and so is your test coverage! Have a fantastic day!
```

# Exercise: Run rspec again

```
$ rspec spec
```

```
.
```

```
Finished in 0.02726 seconds
1 example, 0 failures
```

```
ChefSpec Coverage report generated...
```

```
Total Resources:    1
Touched Resources: 1
Touch Coverage:     100.0%
```

```
You are awesome and so is your test coverage! Have a fantastic day!
```

## Exercise: Upload all recently changed cookbooks

```
Uploading apache [ 0.4.0 ]
Uploading motd [ 0.1.0 ]
Uploaded 2 cookbooks.
```

## Exercise: Upload all recently changed cookbooks

```
$ knife cookbook upload apache motd
```

```
Uploading apache [ 0.4.0 ]
Uploading motd [ 0.1.0 ]
Uploaded 2 cookbooks.
```

# Using Fauxhai to mock platforms

- ChefSpec is great for unit testing cross-platform cookbooks
- Let's add tests to our `mailx` cookbook that supports both Debian and RedHat variants.

# Exercise: Create a spec helper



**OPEN IN EDITOR:** cookbooks/mailx/spec/spec\_helper.rb

```
require 'chefspec'  
require 'chefspec/berkshelf'
```

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

**SAVE FILE!**

- Same as before, only now we're in the mailx cookbook

# Exercise: Write a real test



**OPEN IN EDITOR:** mailx/spec/unit/recipes/default\_spec.rb

```
it 'converges successfully' do
  chef_run # This should not raise an error
end
end

context 'on Debian' do
  let(:chef_run) { ChefSpec::Runner.new({:platform => 'ubuntu', :version => '14.04'}) .converge(described_recipe) }

  it 'should install the correct packages' do
    expect(chef_run).to install_package 'mailutils'
  end
end
```

- Set up different test contexts for different platforms

**SAVE FILE!**

311

# Exercise: Write a real test



**OPEN IN EDITOR:** cookbooks/mailx/spec/unit/default\_spec.rb

```
context 'on CentOS' do
  let(:chef_run) { ChefSpec::Runner.new({:platform => 'centos', :version => '6.5'}) .converge(described_recipe) }

  it 'should install the correct packages' do
    expect(chef_run).to install_package 'mailx'
  end
end
end
```

- Set up test context for CentOS

**SAVE FILE!**

312

# Exercise: Run rspec

```
..  
  
Finished in 0.18828 seconds (files took 5.17 seconds to load)  
2 examples, 1 failures  
  
ChefSpec Coverage report generated...  
  
Total Resources:    1  
Touched Resources: 1  
Touch Coverage:    100.0%  
  
You are awesome and so is your test coverage! Have a fantastic day!
```

# Exercise: Run rspec

```
$ rspec spec
```

```
..
```

```
Finished in 0.18828 seconds (files took 5.17 seconds to load)
2 examples, 1 failures
```

```
ChefSpec Coverage report generated...
```

```
Total Resources:    1
Touched Resources: 1
Touch Coverage:     100.0%
```

```
You are awesome and so is your test coverage! Have a fantastic day!
```

# Exercise: Add cross-platform attributes



**OPEN IN EDITOR:** cookbooks/mailx/attributes/default.rb

```
case node['platform_family']
when "debian"
  default['mailutils']['mailx-package'] = "mailutils"
when "rhel"
  default['mailutils']['mailx-package'] = "mailx"
end
```

**SAVE FILE!**

# Exercise: Install the package



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

```
package node[ 'mailutils' ][ 'mailx-package' ] do
  action :install
end
```

**SAVE FILE!**

# Exercise: Run rspec

```
..  
  
Finished in 0.18828 seconds (files took 5.17 seconds to load)  
2 examples, 0 failures  
  
ChefSpec Coverage report generated...  
  
Total Resources: 2  
Touched Resources: 2  
Touch Coverage: 100.0%  
  
You are awesome and so is your test coverage! Have a fantastic day!
```

# Exercise: Run rspec

```
$ rspec spec
```

```
..
```

```
Finished in 0.18828 seconds (files took 5.17 seconds to load)
2 examples, 0 failures
```

```
ChefSpec Coverage report generated...
```

```
Total Resources: 2
Touched Resources: 2
Touch Coverage: 100.0%
```

```
You are awesome and so is your test coverage! Have a fantastic day!
```

# Review Questions

- What is ChefSpec used for?
- What tool is ChefSpec based on?
- What directory does your tests go into?
- Given a recipe named 'backup', what will the ChefSpec test filename be?

# Further Resources

v1.1.6

# Chef Fundamentals - Q&A Forum

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

# Custom Resources

- **Documentation**
  - [http://docs.chef.io/chef/lwrps\\_custom.html](http://docs.chef.io/chef/lwrps_custom.html)
- **Examples**
  - [http://docs.chef.io/lwp\\_yum.html](http://docs.chef.io/lwp_yum.html)
  - <https://github.com/opscode-cookbooks/powershell>
  - <https://github.com/opscode-cookbooks/aws>

# Ohai Plugins

- **Documentation**
  - <http://docs.chef.io/ohai.html>
- **Examples**
  - <https://github.com/chef/ohai/tree/master/lib/ohai/plugins>
  - [https://github.com/SPSCommerce/chef-etchosts/blob/master/templates/default/plugins/dhcpinfo\\_v7.rb](https://github.com/SPSCommerce/chef-etchosts/blob/master/templates/default/plugins/dhcpinfo_v7.rb)

# Style and Correctness

- **Documentation**
  - <http://docs.chef.io/foodcritic.html>
  - <http://batsov.com/rubocop/>
- **Style and Correctness Rules**
  - <http://foodcritic.io/>
  - <https://github.com/bbatsov/ruby-style-guide>

# Unit Testing

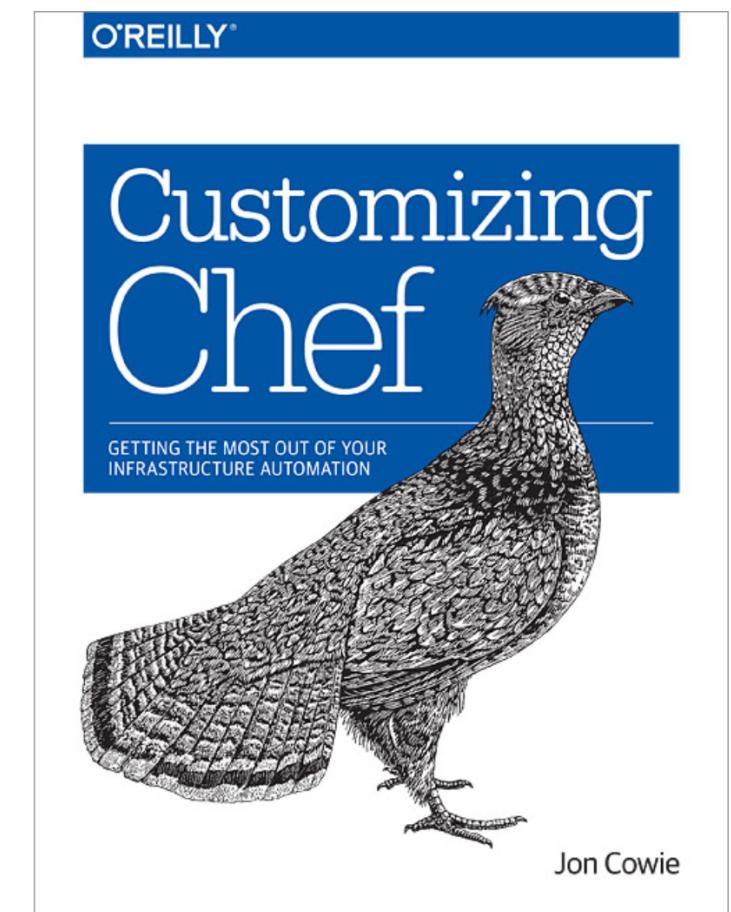
- **Documentation**
  - <http://docs.chef.io/chefspec.html>
  - <https://sethvargo.github.io/chefspec/>
- **Examples**
  - <https://github.com/sethvargo/chefspec/tree/master/examples>

# Further Resources

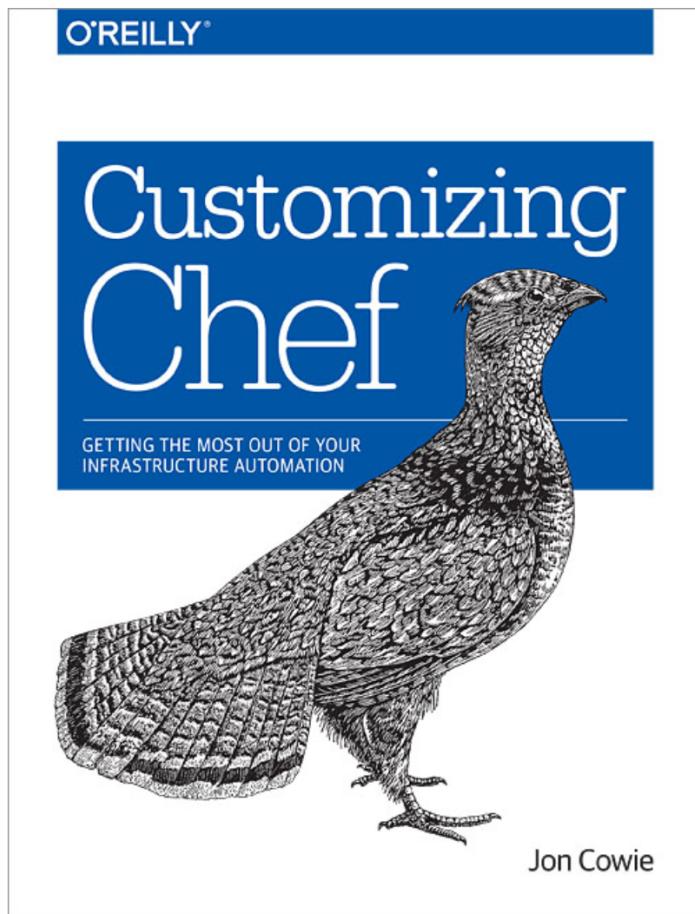
- <http://chef.io/>
- <http://supermarket.chef.io/>
- <http://docs.chef.io/>
- <http://learnchef.com>
- <http://lists.chef.io>
- <http://youtube.com/user/getchef>
- irc.freenode.net #chef, #chef-hacking
- Twitter @chef #getchef

# Learning Chef

- by Mischa Taylor & Seth Vargo of Chef
- The basics



# Customizing Chef



- by Jon Cowie of Etsy
- More detailed treatment of handlers, Ohai plugins, event subsystem

# Test-Driven Infrastructure with Chef



- by Stephen Nelson-Smith
- Introduction to testing your Chef code
- Be sure to get the 2nd edition

# 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



# ChefConf 2015



## ChefConf

ChefConf 2015 is the largest, most vibrant gathering of web-scale IT and DevOps leaders, practitioners, and innovators. Featuring three days of inspired discussions, collaborative presentations, technical training, and hands-on labs focused on automating infrastructure and the continuous delivery of applications.