Intermediate Chef

training@chef.io
Copyright (C) 2015 Chef Software, Inc.

v1.2.1

Introductions

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

Course Objectives

- Upon completion of this course you will be able to:
  - Extend Chef with custom resources and providers
  - Describe the internals of a Chef Client run
  - Create, debug, and distribute custom Ohai plugins
  - Configure report and exception handlers
  - Avoid common cookbook errors using Foodcritic and Rubocop
  - Write simple unit tests with 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

Topics

- Chef Fundamentals Refresher
- Building Custom Resources
- Writing Ohai Plugins
- Chef Client Run Internals
- Implementing Chef Handlers
- Cookbook Style and Correctness
- Foodcritic and Rubocop
- 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

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:fe1f:79a3%en0 prefixlen 64 scopeid 0x4
    inet 10.100.0.84 netmask 0xffffff00 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

Lab VM

- URL_TO_LIST_OF_IP_ADDRESSES

- User: chef
- Password: chef

Lab - Login

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

```
ssh chef@54.174.197.139
The authenticity of host '54.174.197.139 (54.174.197.139)' can't be
established.
RSA key fingerprint is c1:34:39:16:cf:91:c0:d8:ea:53:e9:59:72:1f:a8:5e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.174.197.139' (RSA) to the list of known hosts.
chef@54.174.197.139's password:
```

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

v1.2.1

CHEF

Lesson Objectives

- After completing this lesson, you should be able to:
    - Create a new Organization
    - Manually configure your certificate (.pem file) & 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 Enterprise 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:
  github.com/chef-training/chef-fundamentals-repo
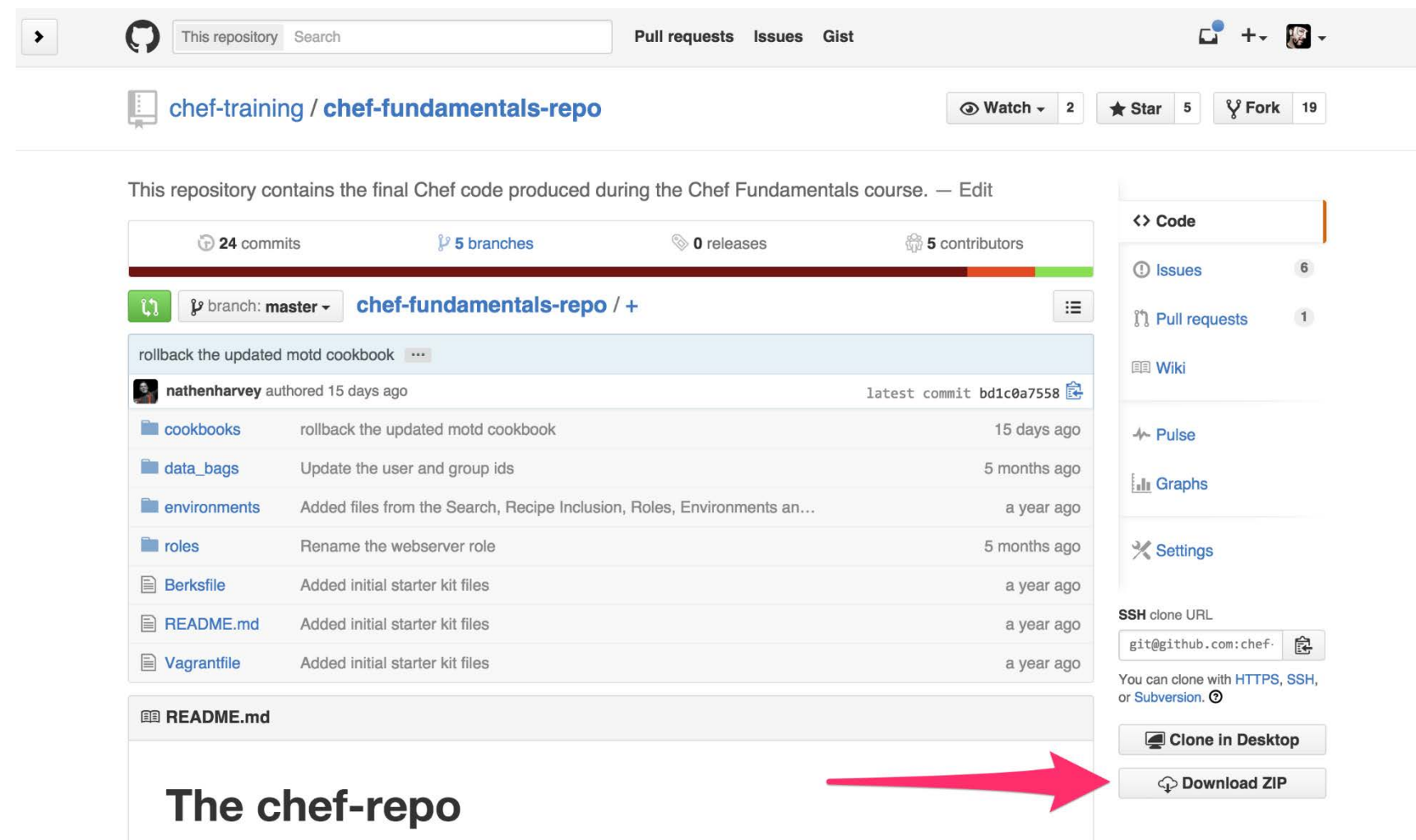
Exercise: Extract the repo to Your Working Directory

```
$ cp ~/Downloads/chef-fundamentals-repo-master.zip .
$ unzip chef-fundamentals-repo-master.zip
```

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

Exercise: View Your Working Directory

```
$ cd chef-fundamentals-repo-master
$ 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 and your 'client.pem' in it

So What's in Our Working Directory Now?

```
cookbooks
├── apache
├── chef-client
├── chef_handler
├── cron
├── logrotate
├── motd
├── ntp
├── pci
├── users
└── windows
10 directories, 1 file
```

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

0 directories, 2 files
```

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

0 directories, 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: Create a New Org

- Create a new account on Hosted Chef

- Configure your workstation to connect to Hosted Chef
  - What is required for this?

- Do NOT download a starter kit

- `knife client list` should show your validator client

Exercise: Create a New Org

- Visit Hosted Enterprise Chef (manage.chef.io)
- Sign in or create a new account
- Create a new Organization
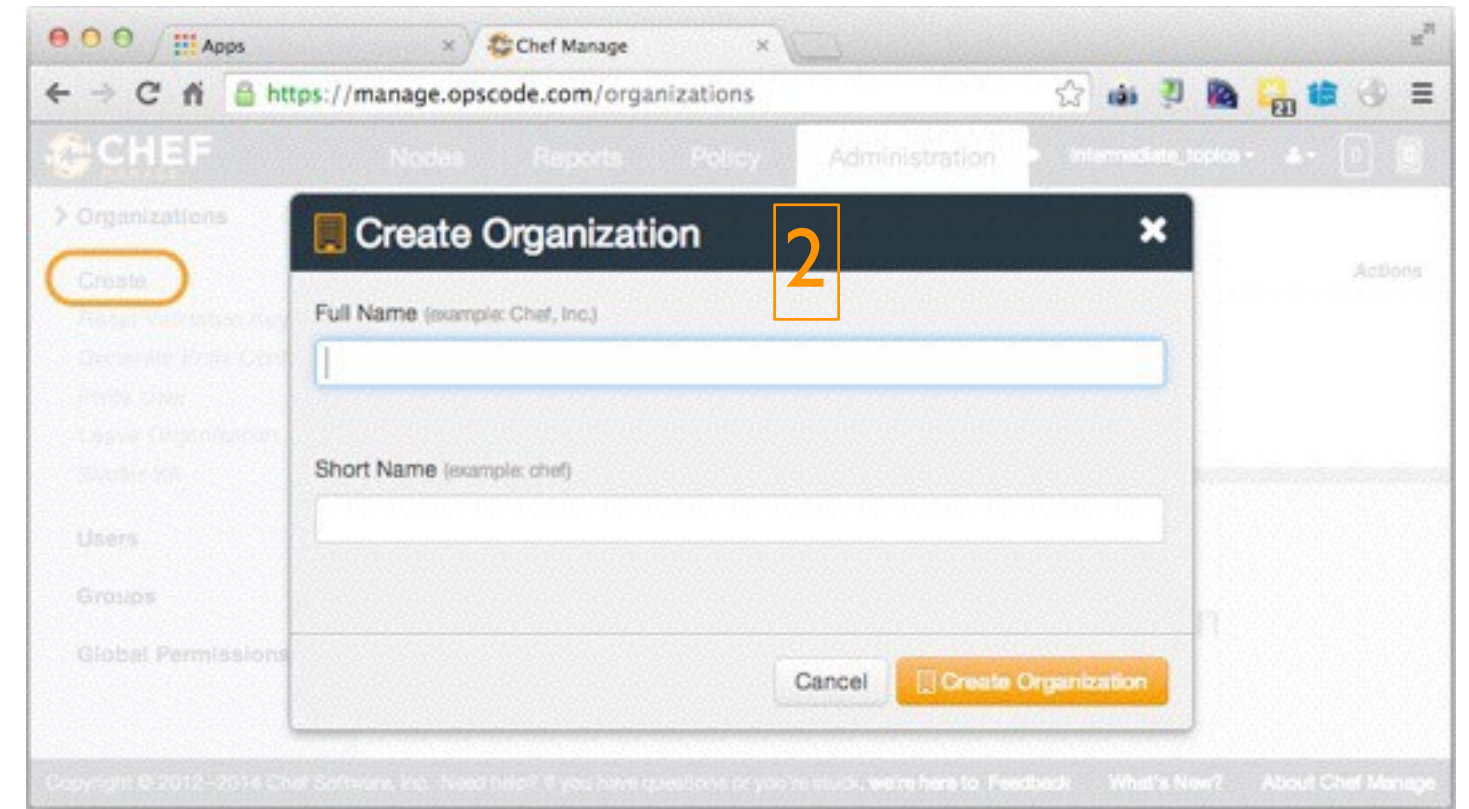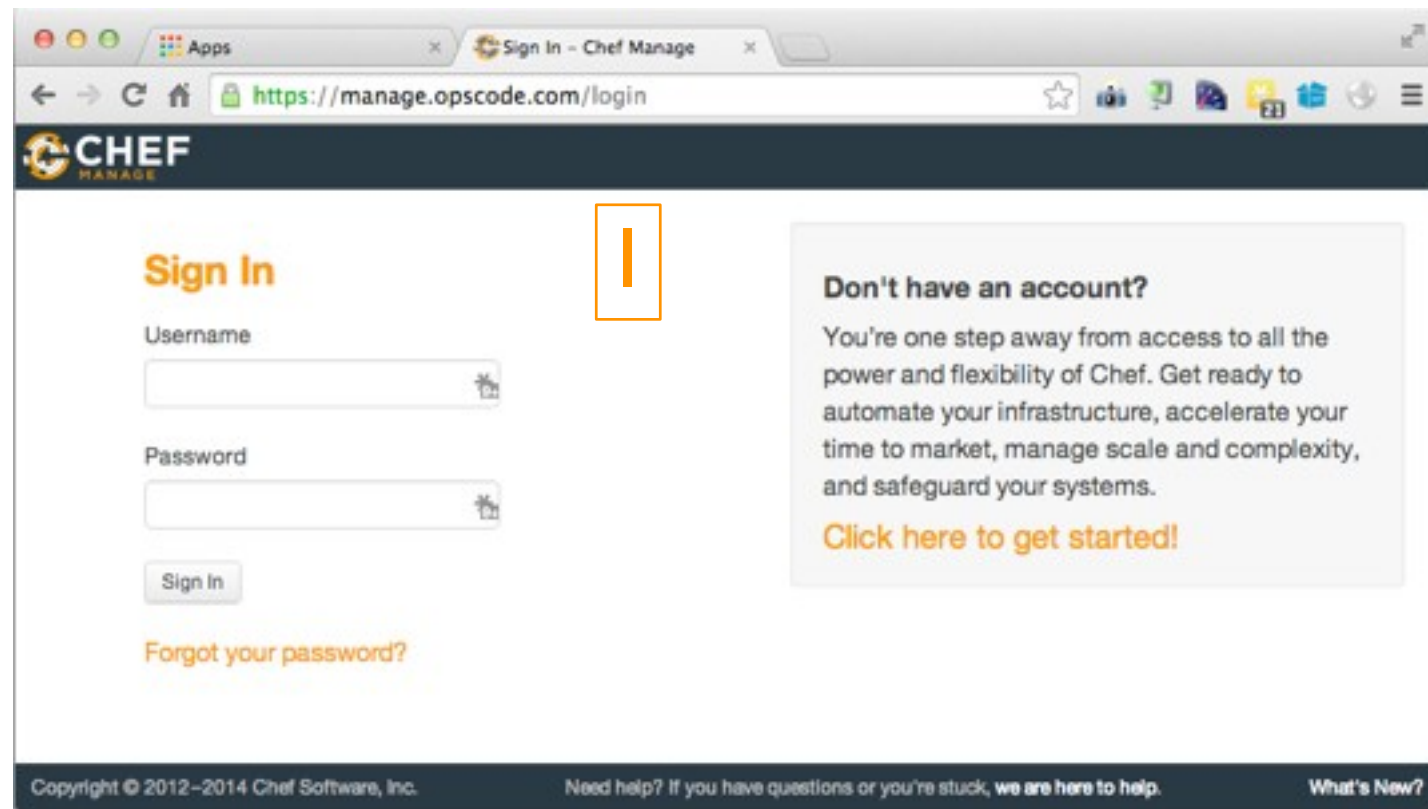
Configuring Your Workstation

- In Chef Fundamentals you set your workstation up the easy way using 'Starter Kit'

- In this class you will download your user .pem file and knife.rb and configure it manually

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
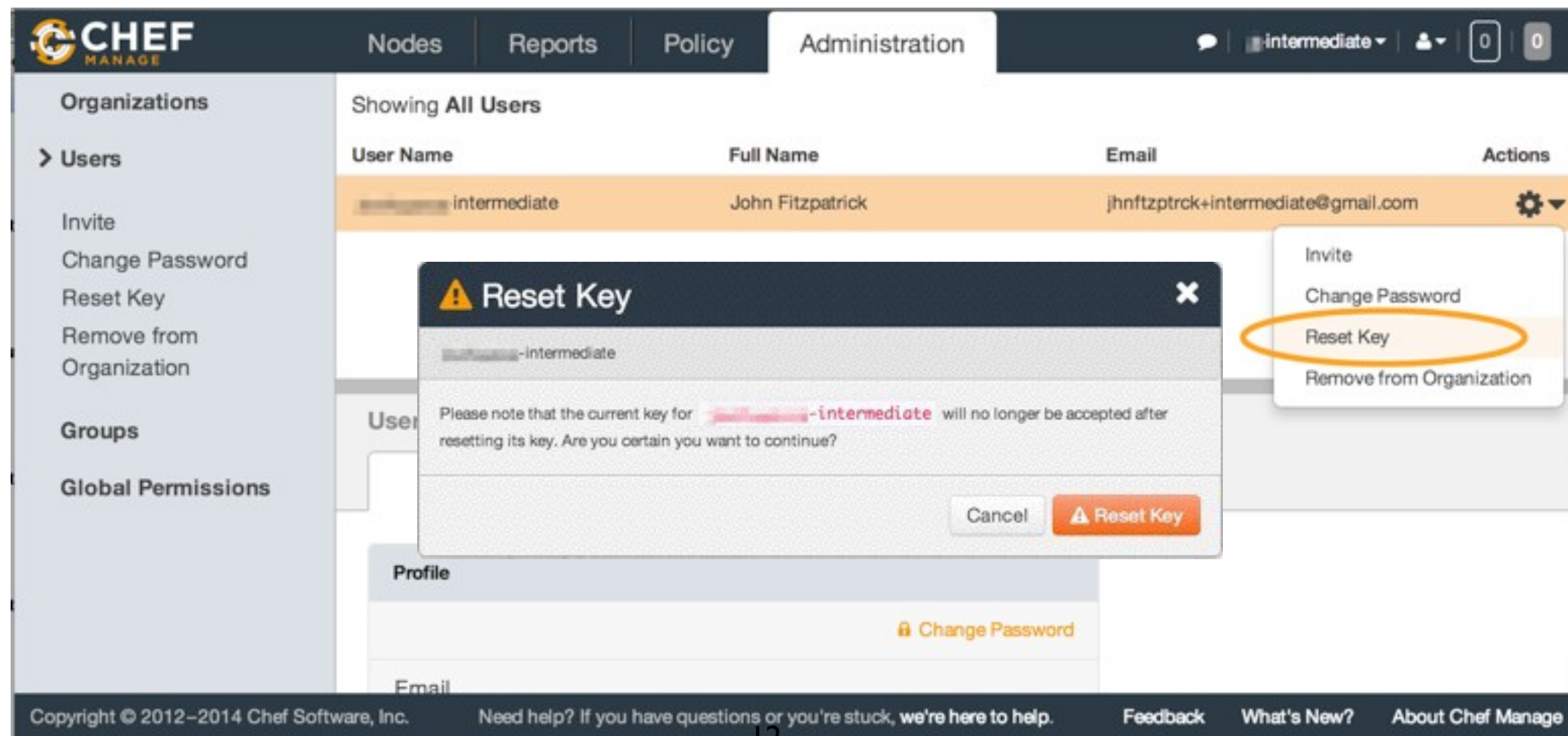
Exercise: Create and Populate a .chef Directory

```
$ cd ~/intermediate/chef-fundamentals-repo-master
$ mkdir .chef
$ cp ~/Downloads/<yourname>.pem .chef
$ cp ~/Downloads/knife.rb .chef
```

- knife.rb & .pem file reside in the `.chef` directory which can be in

  1. `<current-directory>/.chef`

  2. `/etc/.chef`

  3. `~/.chef`

Exercise: Test Your Workstation

```
$ knife client list
```

```
<your-org>-validator.pem
```

Exercise: Upload to Hosted Chef

- Upload the following to the Chef server
  - cookbooks
  - data bags
  - roles
  - environments

Exercise: Upload Cookbooks

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

```
Uploading apache          [0.2.0]
Uploading chef-client     [4.3.0]
Uploading chef_handler    [1.1.9]
Uploading cron            [1.6.1]
Uploading logrotate       [1.9.2]
Uploading motd            [0.1.0]
Uploading ntp             [1.8.6]
Uploading pci             [0.1.0]
Uploading users           [0.1.0]
Uploading windows         [1.37.0]
Uploaded all cookbooks.
```

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

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

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

Exercise: Upload Environments

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

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

"Bootstrap" the Target Instance

```
$ knife bootstrap <EXTERNAL_ADDRESS> \
    --sudo -x chef -P chef -N node1 -r 'role[web]' --bootstrap-version
    12.3.0
```

```
uvo164727i3mvh1jup2.vm.cld.sr --2014-05-13 04:31:10--  https://www.opscode.com/chef/install.sh
uvo164727i3mvh1jup2.vm.cld.sr Resolving www.opscode.com... 184.106.28.90
uvo164727i3mvh1jup2.vm.cld.sr Connecting to www.opscode.com|184.106.28.90|:443... connected.
uvo164727i3mvh1jup2.vm.cld.sr HTTP request sent, awaiting response... 200 OK
uvo164727i3mvh1jup2.vm.cld.sr Length: 15934 (16K) [application/x-sh]
uvo164727i3mvh1jup2.vm.cld.sr Saving to: `STDOUT'
uvo164727i3mvh1jup2.vm.cld.sr
100%[====================================>] 15,934      --.-K/s    in 0s
uvo164727i3mvh1jup2.vm.cld.sr
uvo164727i3mvh1jup2.vm.cld.sr 2014-05-13 04:31:10 (538 MB/s) - written to stdout [15934/15934]
uvo164727i3mvh1jup2.vm.cld.sr
uvo164727i3mvh1jup2.vm.cld.sr Downloading Chef 11.8.2 for el...
uvo164727i3mvh1jup2.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
uvo164727i3mvh1jup2.vm.cld.sr   to file /tmp/install.sh.41533/metadata.txt
uvo164727i3mvh1jup2.vm.cld.sr trying wget...
uvo164727i3mvh1jup2.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
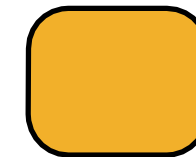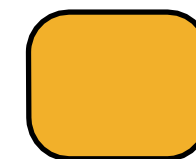
Chef Server

Workstation

Node

knife bootstrap

```
knife bootstrap HOSTNAME -x root -P PASSWORD -N node1
…
```

Chef Server

Workstation

Node

knife bootstrap

```
knife bootstrap HOSTNAME -x root -P PASSWORD -N node1
…
```

Chef Server

Workstation

ssh / scp

chef_server_url

Node

knife bootstrap

```
knife bootstrap HOSTNAME -x root -P PASSWORD -N node1
…
```
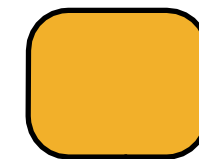
Chef Server

Workstation

ssh / scp

chef_server_url
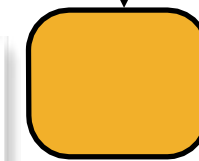
Node

install chef-client
configure chef-client
run chef-client

knife bootstrap

```
knife bootstrap HOSTNAME -x root -P PASSWORD -N node1
…
```


Chef Server

register node
save node details

install chef-client
configure chef-client
run chef-client

Workstation

ssh / scp

chef_server_url

Node

knife bootstrap

```
knife bootstrap HOSTNAME -x root -P PASSWORD -N node1
...
```

Chef Server

Workstation

register node
save node details

chef_server_url

ssh / scp

PostgreSQL

install chef-client
configure chef-client
run chef-client

Node

2-26

knife bootstrap

```
knife bootstrap HOSTNAME -x root -P PASSWORD -N node1
…
```

Chef Server

Workstation

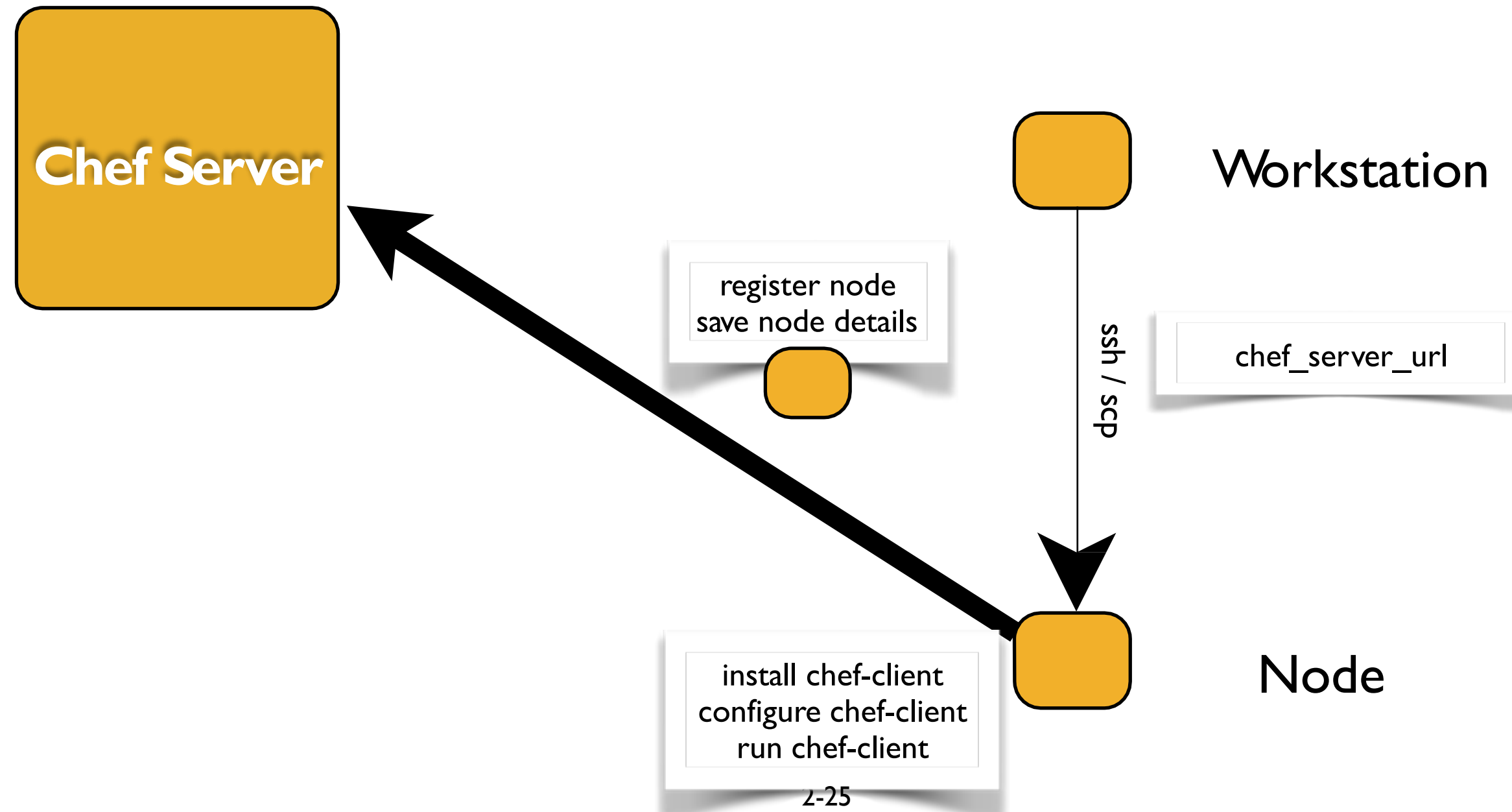register node
save node details

chef_server_url

ssh / scp

Search Index

PostgreSQL

install chef-client
configure chef-client
run chef-client

Node

2-27
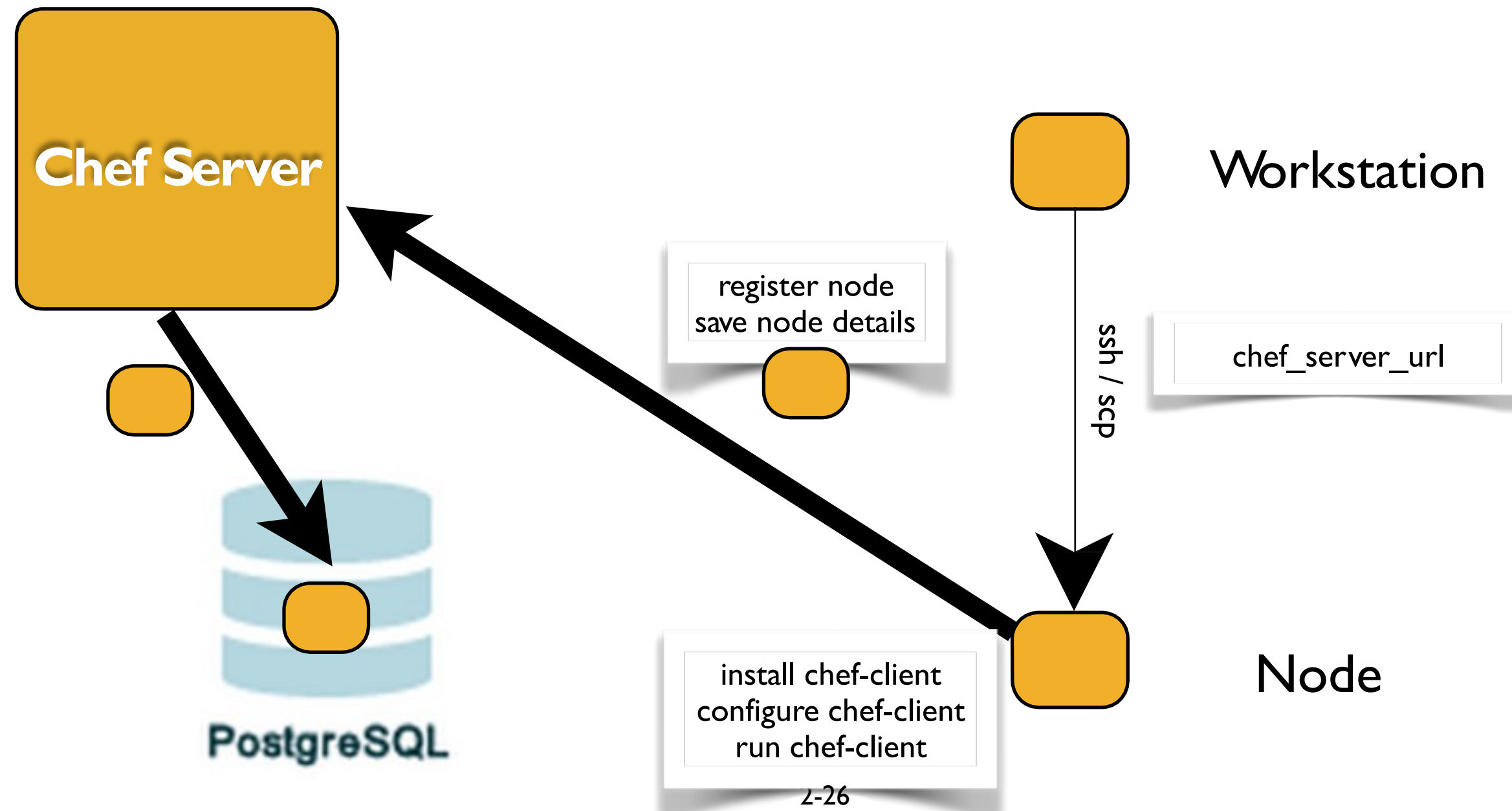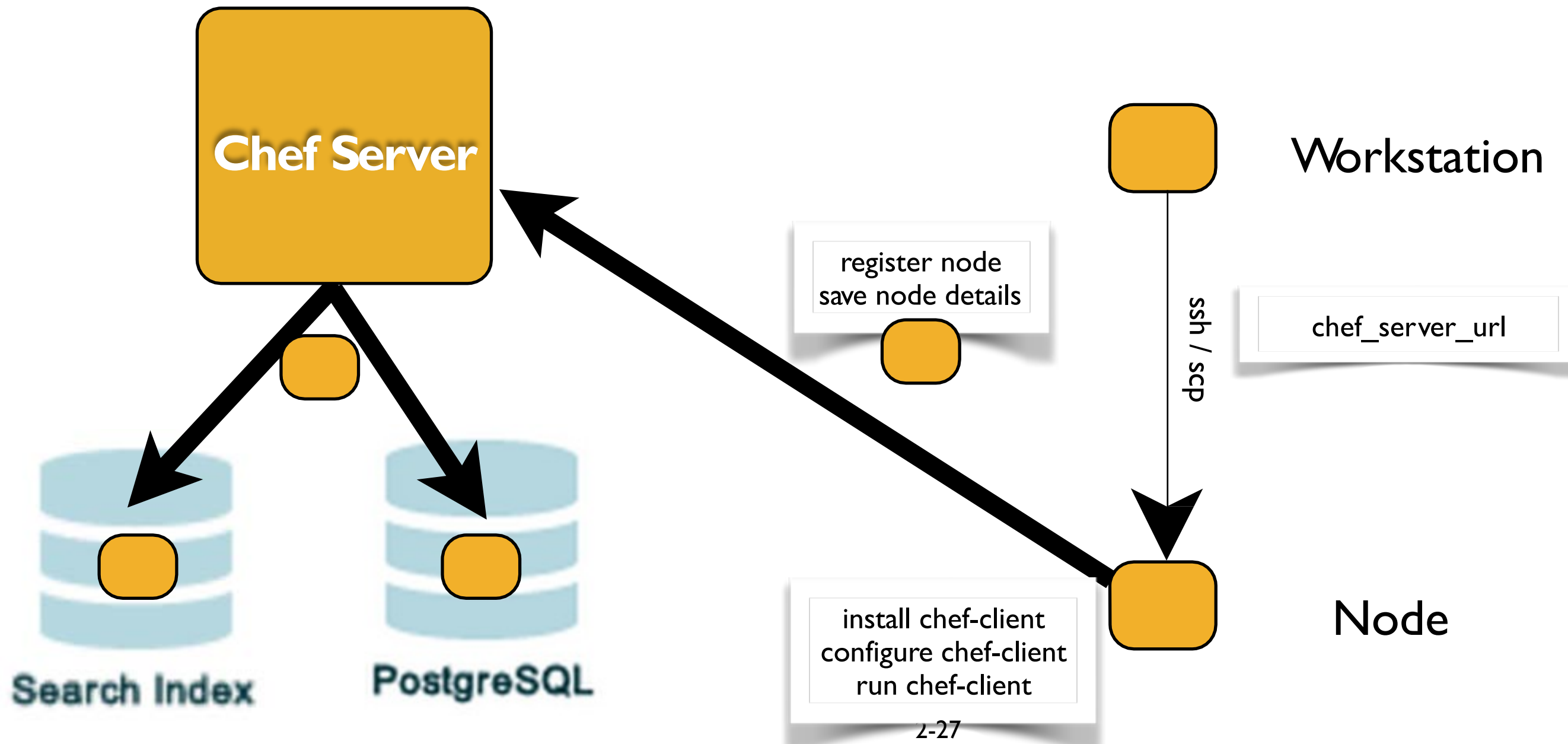
What Just Happened?

- 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 ohai

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

chef@node1:~$ chef-client -v
Chef: 12.3.0
```

Exercise: Re-run the Chef Client

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

```
[2015-06-23T05:06:02+00:00] INFO: Forking chef instance to converge...
Starting Chef Client, version 12.3.0
[2015-06-23T05:06:02+00:00] INFO: *** Chef 12.3.0 ***
[2015-06-23T05:06:02+00:00] INFO: Chef-client pid: 5704
[2015-06-23T05:06:03+00:00] INFO: Run List is [role[web]]
[2015-06-23T05:06:03+00:00] INFO: Run List expands to [chef-client::delete_validation, chef-client,
ntp, motd, users, apache]
[2015-06-23T05:06:03+00:00] INFO: Starting Chef Run for node1
[2015-06-23T05:06:03+00:00] INFO: Running start handlers
[2015-06-23T05:06:03+00:00] INFO: Start handlers complete.
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "ntp", "motd",
"users", "apache"]
[2015-06-23T05:06:04+00:00] INFO: Loading cookbooks [apache@0.2.0, chef-client@4.3.0, cron@1.6.1,
logrotate@1.9.2, motd@0.1.0, ntp@1.8.6, pci@0.1.0, users@0.1.0, chef_handler@1.1.9, windows@1.37.0]
Synchronizing Cookbooks:
  - apache
```

Exercise: Verify chef-client is Running

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

```
root      8933  0.3  2.2 130400 37816 ?      Sl
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 That the Two Sites Are Working
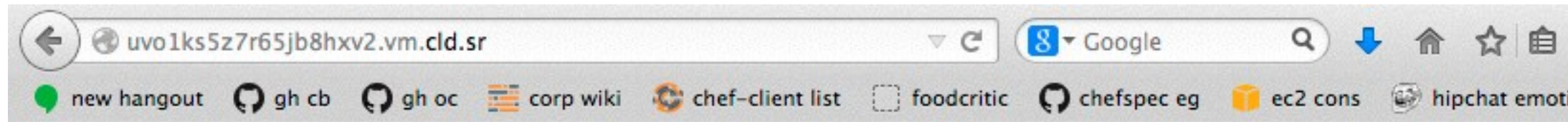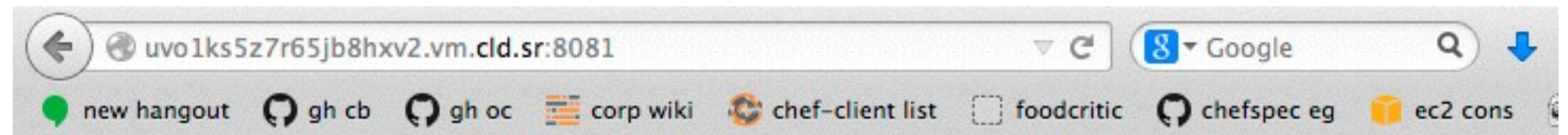
Review Questions

- 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?

Building Custom Resources

Expanding Functionality in the Chef Framework

CHEF

Lesson Objectives

- After completing the lesson, you will be able to
  - Describe the components of the Lightweight Resource/Provider (LWRP) framework
  - Explain the Resource Domain Specific Language (DSL)
  - Explain the Provider DSL
  - Build a resource and provider from scratch

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: Review the apache::default recipe

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

```ruby
# 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

```ruby
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

```ruby
# 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

```ruby
# Add 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: 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

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

```ruby
action :create
```

SAVE FILE!

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

```ruby
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

```ruby
# 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

```ruby
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]`

use_inline_resources

- `use_inline_resources` instructs the embedded resources ("`log`") to notify the parent ("`apache_vhost`") if their states change

- More info:

http://docs.chef.io/lwrp_custom_provider.html#use-inline-resources

LWRPs and the Resource Collection

R_0 R_1 R_2 R_3 R_4 ... R_n

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

LWRPs and the Resource Collection

$$R_0 \quad R_1 \quad R_2 \quad R_3 \quad R_4 \quad \ldots \quad R_n$$

$$L_0 \quad L_1 \quad L_2 \quad L_3$$
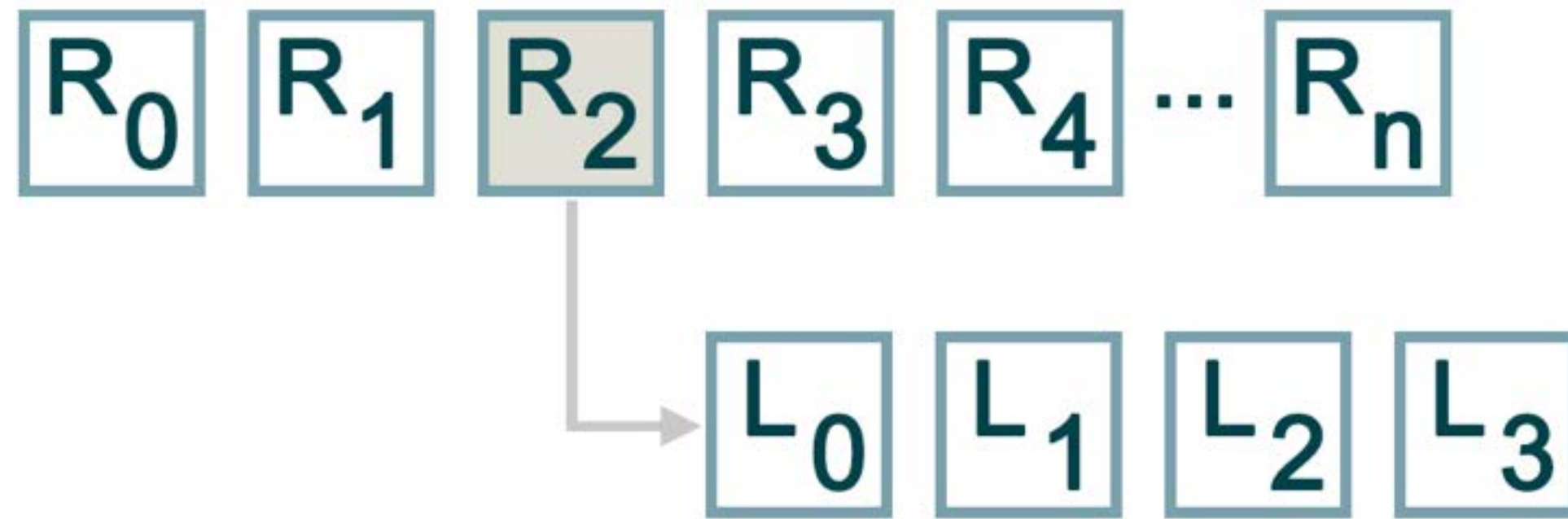
- `use_inline_resources` creates a mini resource collection (execution context)
- Notifications are rolled up to the master resource collection.

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
```

```
  * execute[mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/welcome.conf.disabled] action
run (skipped due to only if)
    * apache_vhost[lions] action create
      * log[My name is lions] action write


  * template[/etc/httpd/conf.d/clowns.conf] action create (up to date)
  * directory[/srv/apache/clowns] action create (up to date)
  * template[/srv/apache/clowns/index.html] action create (up to date)
  * template[/etc/httpd/conf.d/bears.conf] action create (up to date)
  * directory[/srv/apache/bears] action create (up to date)
  * template[/srv/apache/bears/index.html] action create (up to date)
  * template[/etc/httpd/conf.d/admin.conf] action create (up to date)
  * directory[/srv/apache/admin] action create (up to date)
  * template[/srv/apache/admin/index.html] action create (up to date)
  * service[httpd] action enable (up to date)
  * service[httpd] action start (up to date)

Running handlers:
Running handlers complete
Chef Client finished, 2/42 resources updated in 4.386652029 seconds
```

Exercise: Create Attribute Parameters for the apache_vhost Resource

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

```ruby
actions :create

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

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

```
actions :create

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

```ruby
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`

```ruby
# 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

```ruby
# 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
```

```
  * apache_vhost[lions] action create
    * template[/etc/httpd/conf.d/lions.conf] action create
        create new file /etc/httpd/conf.d/lions.conf
      - update content in file /etc/httpd/conf.d/lions.conf from none to 75b467
      --- /etc/httpd/conf.d/lions.conf  2015-06-23 06:09:51.291365440 +0000
      +++ /tmp/chef-rendered-template20150623-29275-1uqg8tg    2015-06-23
06:09:51.291365440 +0000
      @@ -1 +1,18 @@
      +  Listen 8080
      +
      +<VirtualHost *:8080>
      +  ServerAdmin webmaster@localhost
      +
      +  DocumentRoot /srv/apache/lions
      +  <Directory />
      +    Options FollowSymLinks
      +    AllowOverride None
```

Exercise: Verify New 'lions' Site

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

Exercise: Add the lions to the attributes

OPEN IN EDITOR: cookbooks/apache/attributes/default.rb

```
default["apache"]["sites"]["clowns"] = { "port" => 80 }
default["apache"]["sites"]["bears"] = { "port" => 81 }
default["apache"]["sites"]["lions"] = { "port" => 8080 }
```

Exercise: Refactor apache::default Recipe

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

```ruby
# Iterate over the apache sites
node["apache"]["sites"].each do |site_name, site_data|
  # Enable an Apache Virtualhost
  apache_vhost site_name do
    site_port site_data['port']
    action :create
    notifies :restart, "service[httpd]"
  end
end
```

Exercise: Remove the lions apache_vhost

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

```ruby
# 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|
  # Enable an Apache Virtualhost
  apache_vhost site_name do
    site_port site_data['port']
```

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
```

```
  * apache_vhost[lions] action create
    * template[/etc/httpd/conf.d/lions.conf] action create (up to date)
    * directory[/srv/apache/lions] action create (up to date)
    * template[/srv/apache/lions/index.html] action create (up to date)
     (up to date)
   * apache_vhost[lions] action create
    * template[/etc/httpd/conf.d/lions.conf] action create (up to date)
    * directory[/srv/apache/lions] action create (up to date)
    * template[/srv/apache/lions/index.html] action create (up to date)
     (up to date)
  * apache_vhost[clowns] action create
    * template[/etc/httpd/conf.d/clowns.conf] action create (up to date)
    * directory[/srv/apache/clowns] action create (up to date)
    * template[/srv/apache/clowns/index.html] action create (up to date)
     (up to date)
```

Exercise: Create an apache_vhost Resource with Two Allowed Actions

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

```ruby
action :create, :remove

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

SAVE FILE!

Exercise: The :remove Action

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

```ruby
    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

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


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

- Delete existing **execute** resource that disables the welcome site

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
node["apache"]["sites"].each do |site_name, site_data|
```

- 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
```

```
Recipe: apache::default
  * yum_package[httpd] action install (up to date)
  * apache_vhost[welcome] action remove
    * file[/etc/httpd/conf.d/welcome] action delete (up to date)
     (up to date)
  * apache_vhost[lions] action create
    * template[/etc/httpd/conf.d/lions.conf] action create (up to date)
    * directory[/srv/apache/lions] action create (up to date)
    * template[/srv/apache/lions/index.html] action create (up to date)
     (up to date)
```

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
- 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?
- How did `use_inline_resources` effect the resource collection?
- What were some of the parameters supported by LWRP attributes?

Writing Ohai Plugins

Getting More Data From Your Systems

Lesson Objectives

After completing the lesson, you will be able to:

- Explain the function Ohai and Ohai Plugins
- Write your own Ohai Plugins
- Force Ohai Plugins to run
- Disable Ohai Plugins

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...

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)

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

- Plugins can be disabled

Loading and Disabling Plugins

- The /etc/chef/client.rb file manages which additional plugins are loaded and the plugins to disable.

- This file is maintained by the chef-client cookbook's config recipe.

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`

- Let us look at the `chef-client::config` recipe.

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
```

Exercise: View the chef-client::config Recipe

```erb
<% if node.attribute?("ohai") && node["ohai"].attribute?("plugin_path") -%>

Ohai::Config[:plugin_path] << "<%= node["ohai"]["plugin_path"] %>"
<% end -%>
<% unless @ohai_disabled_plugins.empty? -%>
Ohai::Config[:disabled_plugins] = [<%= @ohai_disabled_plugins.map { |k| k.match(/...

<% end -%>
```

Exercise: Update the base role

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

Exercise: Upload the base role

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

```
Updated Role base!
```

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 2.0.0
```

```
Downloading ohai from the cookbooks site at version 2.0.0 to /Users/YOU/chef-
repo/ohai-2.0.0.tar.gz


Cookbook saved: /Users/YOU/chef-repo/cookbooks/ohai-2.0.0.tar.gz
```

Exercise: Download the Ohai Cookbook

```
$ tar -zxvf ohai-2.0.0.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
```

Exercise: Upload the Ohai Cookbook

```
$ knife cookbook upload ohai
```

```
Uploading ohai [2.0.0]

Uploaded 1 cookbook.
```

Exercise: Inspect the Ohai Cookbook

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

```ruby
# 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

```ruby
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

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

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

  collect_data(:platform) do
    attribute_a Mash.new
    # platform-specific Ruby code
  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
    attribute_a Mash.new
    # some Ruby code
  end

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

4-23

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

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

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

  collect_data(:platform) do
    attribute_a Mash.new
    # platform-specific Ruby code
  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

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

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

  collect_data(:platform) do
    attribute_a Mash.new
    # platform-specific Ruby code
  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

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

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

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

Exercise: Create modules.rb

```ruby
Ohai.plugin(:Apache) do
provides "apache/modules"

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

Mash.new

- It's essentially a Hash that allows you to use string keys and symbol keys interchangeably to return the same value.

- https://github.com/chef/chef/blob/master/lib/chef/mash.rb

shell_out (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 shell_out (Mixlib::Shellout)!**

Exercise: Write the ohai_plugin recipe

OPEN IN EDITOR: cookbooks/apache/recipes/ohai_plugin.rb

```ruby
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.
```

Exercise: Add apache::ohai_plugin to the web role

📄 **OPEN IN EDITOR:** `roles/web.rb`

```
name "web"
description "Web Server"
run_list "role[base]", "recipe[apache]", "recipe[apache::ohai_plugin]"
default_attributes({
  "apache" => {
    "sites" => {
      "admin" => {
        "port" => 8000
      },
      "bears" => {
        "port" => 8081
      }
    }
  } })
```

Exercise: 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 12.3.0
resolving cookbooks for run list: ["chef-client::config", "chef-
client::delete_validation", "chef-client", "ntp", "motd", "users", "apache",
"apache::ohai_plugin"]
Synchronizing Cookbooks:
  - ohai
  - cron
  - apache
  - chef-client
  - chef_handler
```

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

  provides "apache/modules"


  collect_data(:default) do

    apache Mash.new

    modules = shell_out("apachectl -t -D DUMP_MODULES")

    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`

```ruby
Ohai.plugin(:Apache) do
  provides "apache/modules"

  collect_data(:default) do
    apache Mash.new
    apache[:modules] = { :static => [], :shared => [] }
    modules = shell_out("apachectl -t -D DUMP_MODULES")
```

Exercise: Refactor modules.rb

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

```ruby
modules.stdout.each_line do |line|
  fullkey, value = line.split("(", 2).map { |token| token.strip }
  apache_module = fullkey.gsub("_module","")
  dso_type = value.to_s.chomp("\)")
  if dso_type.match(/shared/)
    apache[:modules][:shared] << apache_module
  elsif dso_type.match(/static/)
    apache[:modules][:static] << apache_module
  end
end

end
```

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
```

Exercise: Show apache.modules Attributes

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

```
node1:

  apache.modules:

    shared:

      auth_basic

      auth_digest

      authn_file

      ...

    static:

      core

      mpm_prefork
```

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
```

Exercise: Update the Base Role

📄 **OPEN IN EDITOR:** `roles/base.rb`

```
name "base"
description "Base Server Role"
run_list "recipe[chef-client::config]", "recipe[chef-client::delete_v...
default_attributes({
  "ohai" => {
    "disabled_plugins" => [ ":Passwd" ]
  }
})
```

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"

validations_client_name "intermediate050614-validator"

verify_api_cert true

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)
```

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

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"`?
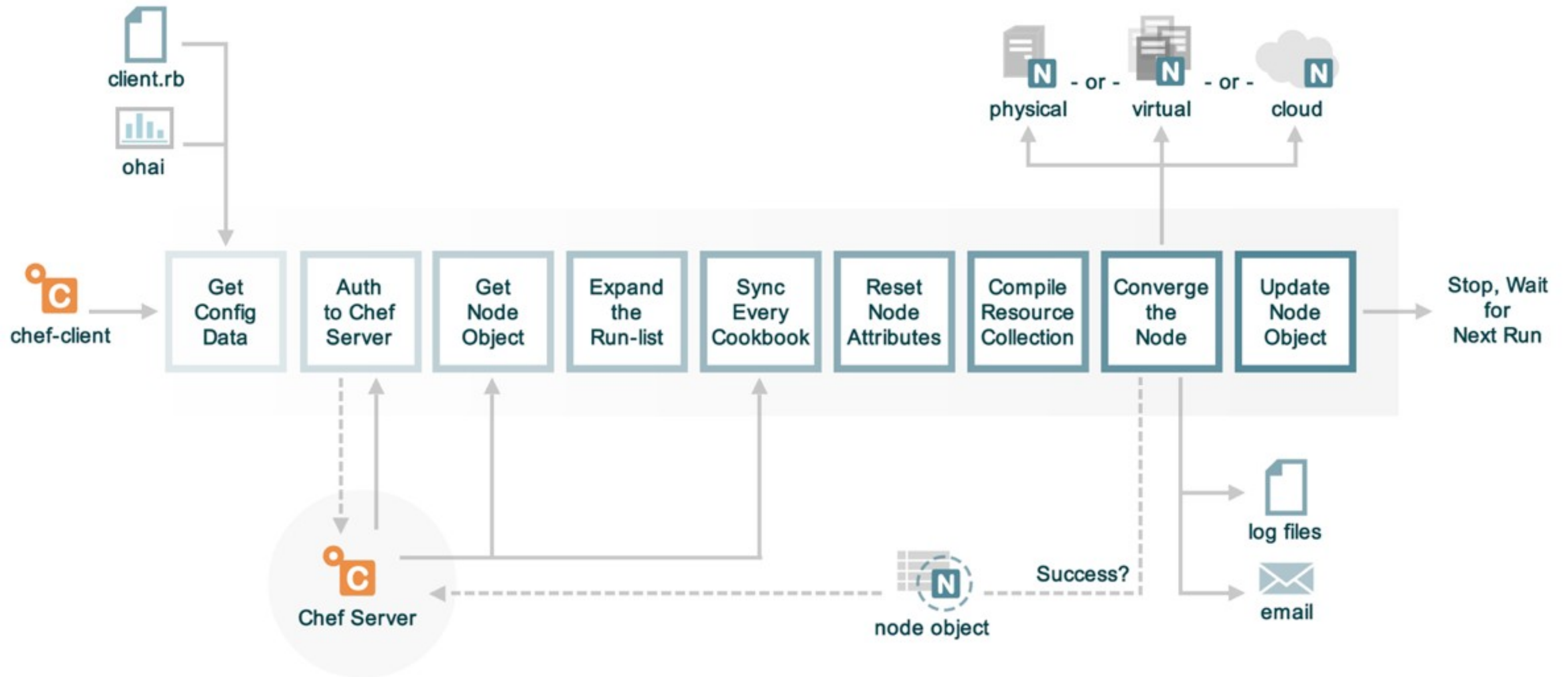
Chef Client Run Internals

What Really Happens When Chef Runs?

Lesson Objectives

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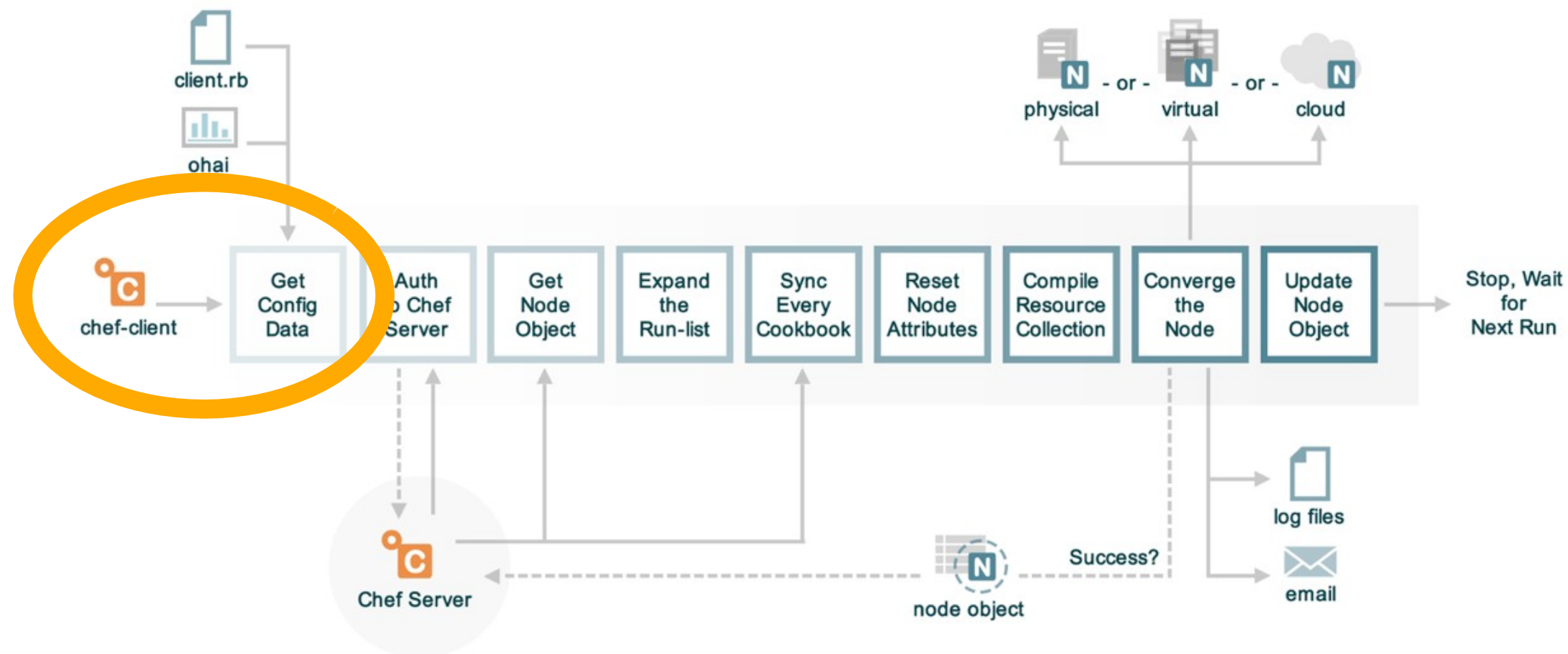 - Use `node.run_state` to pass information around the recipe at execution time

Remember This From Fundamentals?

Customizations

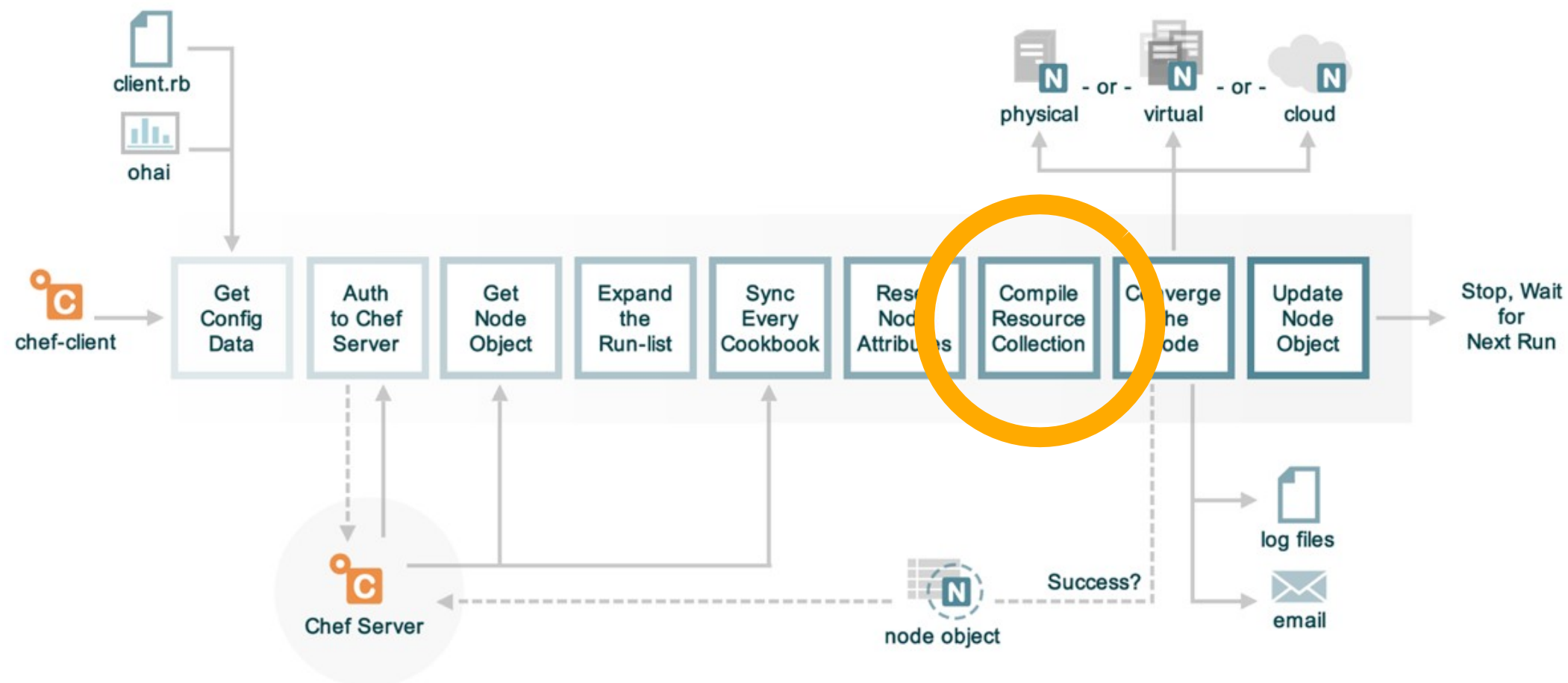- Ohai Plugins

Customizations

- LWRPs
- HWRPs
- Libraries

Customizations
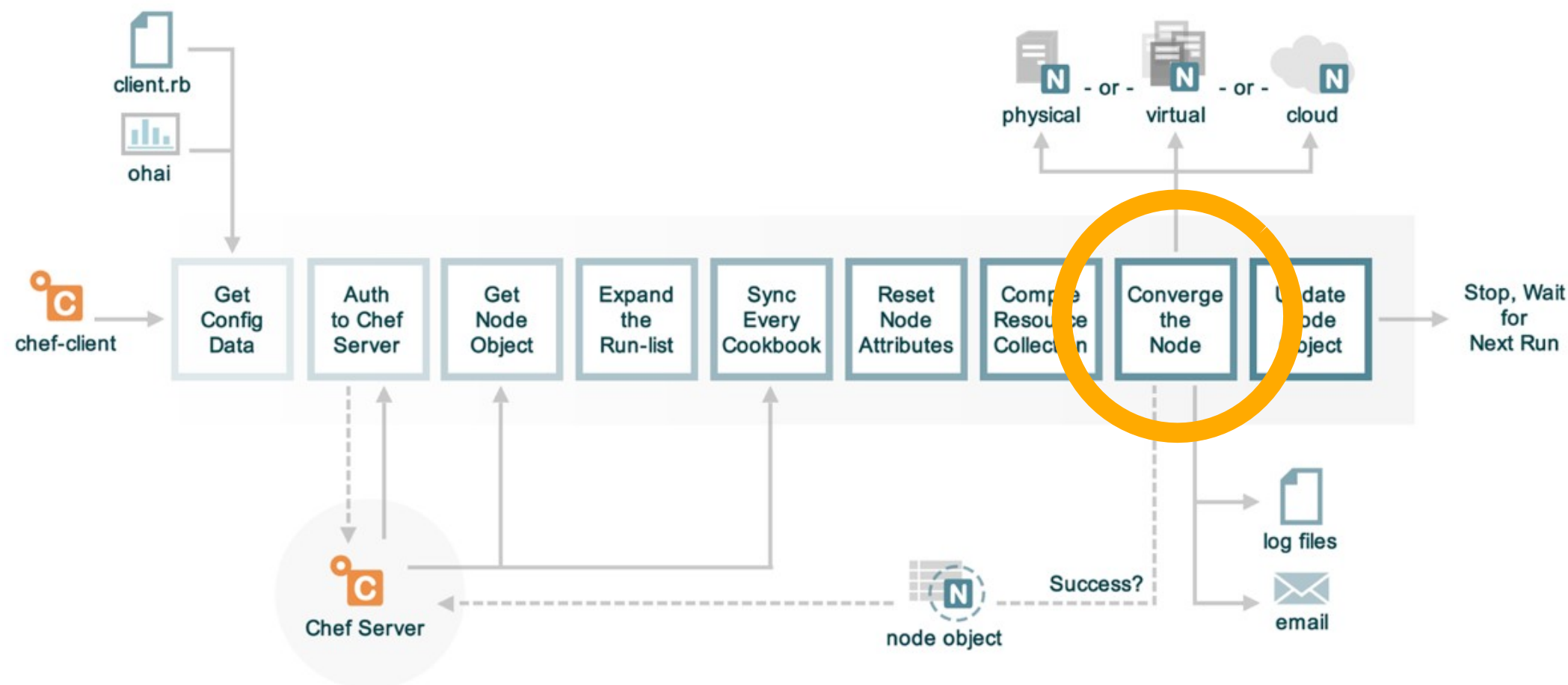
- Exception Handlers
- Report Handlers

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

- **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

```ruby
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
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
```

Review Questions

- Where would you stash transient data during the run?

- What object stores the current status of the run?

- What resource allows you to execute ruby code at execute time?

- What can be used to populate resource attributes at execute time?

Implementing Chef Handlers

Communicating the Status of a Chef Client Run

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://onddo.github.io/chef-handler-zookeeper/

  - 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/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 1.1.9
```

```
Downloading chef_handler from the cookbooks site
at version 1.1.9 to /Users/YOU/chef-repo/
chef_handler-1.1.9.tar.gz

Cookbook saved: /Users/YOU/chef-repo/
chef_handler-1.1.9.tar.gz
```

Exercise: Download the chef-client Cookbook

```
$ tar -zxvf chef_handler-1.1.9.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          [1.1.9]
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

Dear Ruby, How Do You Email?

- We could look at Ruby's Standard Library -

  http://www.rubydoc.info/stdlib/core/1.9.3 - but that won't get it

- Or we could look at Ruby's Extended Library -

  http://www.rubydoc.info/stdlib - but we won't find it there

- We could look at Ruby gems - https://www.rubygems.org/ - to no avail

- This will do it - 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'

```
$ knife cookbook create email_handler
```

```
** Creating cookbook email_handler
** Creating README for cookbook: email_handler
** Creating CHANGELOG for cookbook: email_handler
** Creating metadata for cookbook: email_handler
```

Exercise: Edit the default Recipe

OPEN IN EDITOR: cookbooks/email_handler/recipes/default.rb

```ruby
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

```ruby
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
```

Exercise: Setup the Handler

```ruby
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!

6-20

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

```ruby
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`

```ruby
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

```ruby
    @to_address = to_address
  end


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

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

6-24

The updated_resources Hash

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

```ruby
    body = ""
    # 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|
        body += "The resource #{r.name} was changed in cookbook
#{r.cookbook_name} at #{r.source_line}\n"
      end
    else
      body += "No resources changed by chef-client\n"
    end
```

- **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`

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

    Pony.mail(:to => @to_address,
              :from => @from_address,
              :subject => subject,
              :body => body)
   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-3.1.8.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        [3.1.8]
upload complete
```

Exercise: Create the mailx Cookbook

```
$ knife cookbook create mailx
```

```
** Creating cookbook mailx
** Creating README for cookbook: mailx
** Creating CHANGELOG for cookbook: mailx
** Creating metadata for cookbook: mailx
```

Exercise: Install the Package

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

SAVE FILE!

Exercise: Upload the email_handler Cookbook

```
$ knife cookbook upload mailx
```

```
Uploading mailx                          [0.1.0]
Uploaded 1 cookbook.
```

Exercise: Add the Mail Dependencies

OPEN IN EDITOR: cookbooks/email_handler/recipes/default.rb

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

include_recipe "chef_handler"
```

```ruby
include_recipe "postfix"
include_recipe "mailx"
```

```ruby
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

```ruby
name                "email_handler"
maintainer          "You"
maintainer_email    "you@somewhere.com"
license             "Apache 2.0"
description          "Email me on every Chef run"
long_description IO.read(File.join(File.dirname(__FILE__),
'README.md'))
version             "0.1.0"


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

SAVE FILE!

Exercise: Upload the email_handler Cookbook

```
$ knife cookbook upload email_handler
```

```
Uploading email_handler                [0.1.0]
Uploaded 1 cookbook.
```

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::config]",
"recipe[chef-client::delete_validation]",, "recipe[chef-client]",
"recipe[ntp]", "recipe[motd]", "recipe[users]"
```

SAVE FILE

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

CHEF

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

Devops is a Two-Way Street

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

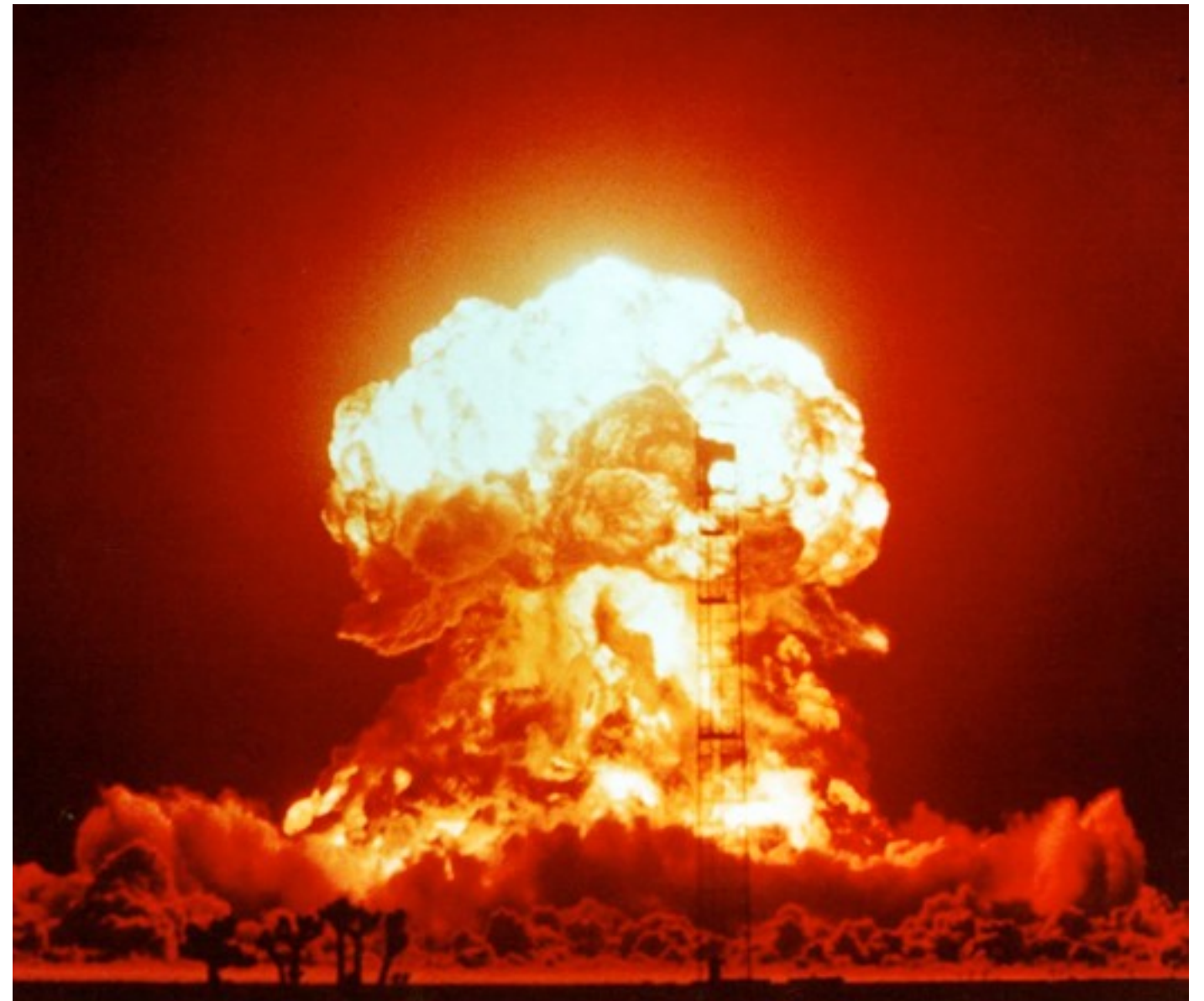# Devops is a Two-Way Street

- **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

CHEF™

Module Objectives

- After completing this module, you should be able to:

  - Use Foodcritic to avoid common cookbook errors

What is Foodcritic?

- A lint testing tool, or correctness checker

- Ensure you adhere to best practices for cookbook style

- Ability to write and use custom rules

- http://foodcritic.io/

FOODCRITIC

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
```

8- 5

Abide by Best Practices

- There are 50+ rules in Foodcritic; your cookbooks should pass them (or have a good reason if they do 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

Foodcritic

- If you are using the ChefDK, you already have foodcritic installed

- Located at: .\chefdk\embedded\bin\foodcritic

Exercise: Check Cookbook Correctness with Foodcritic

```
$ foodcritic cookbooks/apache
```

```
FC003: Check whether you are running with chef server before using server-
specific features: ./recipes/default.rb:43
FC003: Check whether you are running with chef server before using server-
specific features: ./recipes/iplogger.rb:1
FC008: Generated cookbook metadata needs updating: ./metadata.rb:2
FC008: Generated cookbook metadata needs updating: ./metadata.rb:3
FC009: Resource attribute not recognised: ./recipes/default.rb:27
FC016: LWRP does not declare a default action: ./resources/vhost.rb:1
```

Exercise: Narrow the Tests

```
$ foodcritic cookbooks/apache -t ~FC003 -t ~FC009
```

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

Exercise: Fix the Cookbook to Clear the Errors

```
$ foodcritic cookbooks/apache -t ~FC003 -t ~FC009
```

```
(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
```

Saving Foodcritic Rules

- When you have defined your set of tag options, you can save them to a file called **.foodcritic**
- Foodcritic will read that file upon execution, and use those as the default options

http://www.foodcritic.io/#FC003

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

Lesson Objectives

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

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

Rubocop

- New Ruby developers often ask for guidance on writing idiomatic Ruby

- Rubocop gives the same kind of feedback for your Ruby style that Foodcritic gives for your Chef Code

- Evaluates your Chef code for Ruby best practices, not for Chef style

- Documentation located here

  https://github.com/bbatsov/rubocop

Navigate to Apache Cookbook

```
$ cd cookbooks/apache
```

Exercise: Use Additional Rules

```
$ rubocop
```

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


4 files inspected, 50 offences detected
```

Fixing Rubocop Offenses

- We can mask offenses using the rubocop configuration file `.rubocop.yml`

- Rubocop can automatically generate a configuration called file named `.rubocop_todo.yml` that contains entries to mask all of your offenses

- We can use `.rubocop_todo.yml` to iterate through the offenses

- `.rubocop_todo.yml` settings are documented at

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

Exercise: Generate .rubocop_todo.yml

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

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


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

Exercise: Review .rubocop_todo.yml

📄 **OPEN IN EDITOR:** `.rubocop_todo.yml`

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


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


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

Exercise: Review .rubocop_todo.yml

OPEN IN EDITOR: `.rubocop_todo.yml`

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


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


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

Exercise: Update .rubocop.yml

📄 **OPEN IN EDITOR:** .rubocop.yml

```
inherit from: .rubocop_todo.yml
```

SAVE FILE!

Exercise: Re-run rubocop

```
$ rubocop
```

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

4 files inspected, no offences detected
```

Work Through the Issues

- We now have a configuration file that masks all of our offenses

- We want to fix some of the offenses and mask the others

- Move the configuration rules for the offenses we want to continue masking from
  `.rubocop_todo.yml` to `.rubocop.yml`

Exercise: Update Rubocop Configuration

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

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

Encoding:
  Enabled: false


LineLength:
  Max: 200


HashSyntax:
  EnforcedStyle: hash_rockets


StringLiterals:
  Enabled: false
```

SAVE FILE!

Exercise: Re-generate .rubocop_todo.yml

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

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


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

Fixing Rubocop Offenses

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

Exercise: Update .rubocop.yml

📄 **OPEN IN EDITOR:** `.rubocop.yml`

```
inherit from: .rubocop_todo.yml      <<- Un-comment this line


Encoding:
  Enabled: false


LineLength:
  Max: 200


HashSyntax:
  EnforcedStyle: hash_rockets


StringLiterals:
  Enabled: false
```

SAVE FILE!

Exercise: Delete Entry From .rubocop_todo.yml

OPEN IN EDITOR: .rubocop_todo.yml

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


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


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

SAVE FILE!

Exercise: Re-run rubocop

```
$ rubocop
```

```
Offences:

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


4 files inspected, 1 offence detected
```

Exercise: Fix Default Recipe

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

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


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


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

Remove trailing whitespace
from line 19

SAVE FILE!

Exercise: Re-run rubocop

```
$ rubocop
```

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

7 files inspected, no offenses detected
```

WIN!

Review Questions

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

An Introduction to ChefSpec

Unit Testing Your Cookbooks to Prevent Regressions

Lesson Objectives

- After completing the lesson, you will be able to:
  - Explain what unit testing means for Chef cookbooks and recipes
  - Explain 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 do not 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

Installing ChefSpec

- ChefSpec is already included in the ChefDK

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

```ruby
require 'spec_helper'

describe 'cookbook_name::recipe_name' do
  let(:chef_run) { ChefSpec::SoloRunner.converge(described_recipe) }
  it 'does something' do
    expect(chef_run).to action_resourcetype('RESOURCE_NAME')
    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: Move into cookbook

```
$ cd cookbooks/motd
```

```
(No output)
```

Exercise: Make a 'spec' Directory

```
$ rspec --init
```

```
(No output)
```

Exercise: Make a 'spec/unit' Directory

```
$ mkdir spec/unit
```

```
(No output)
```

Exercise: Create a Spec Helper

OPEN IN EDITOR: cookbooks/motd/spec/spec_helper.rb

```ruby
require 'chefspec'


ChefSpec::Coverage.start!


# This file was generated by the `rspec --init`
```

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

```ruby
require 'spec_helper'

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

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

- 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

```
$ rspec
```

```
*

Pending:
  motd::default does something
    # need to write this test
    # ./cookbooks/motd/spec/unit/default_spec.rb:7


Finished in 0.00033 seconds
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

```ruby
require 'spec_helper'

describe 'motd::default' do
  let(:chef_run) { ChefSpec::SoloRunner.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

10-15

Exercise: Run rspec

```
$ rspec
```

```
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

```ruby
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

```
$ rspec
```

```
.

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

```
$ knife cookbook upload motd
```

```
Uploading motd              [0.1.0]
Uploaded 1 cookbook.
```

Using Fauxhai to Mock Platforms

- ChefSpec is great for unit testing cross-platform cookbooks

- Let us add tests to our `mailx` cookbook that supports both CentOS and Ubuntu variants.

- On CentOS mailx is installed through the mailx package.

- On Ubuntu mailx is installed through the mailutils package.

Exercise: Move into cookbook

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

```
(No output)
```

Exercise: Make a 'spec' Directory

```
$ rspec --init
```

```
(No output)
```

Exercise: Make a 'spec/unit' Directory

```
$ mkdir spec/unit
```

```
(No output)
```

Exercise: Create a Spec Helper

OPEN IN EDITOR: `cookbooks/mailx/spec/spec_helper.rb`

```ruby
require 'chefspec'

ChefSpec::Coverage.start!
```

SAVE FILE!

- Same as before, only now we are in the `mailx` cookbook

Exercise: Write a Real Test

📄 **OPEN IN EDITOR:** cookbooks/mailx/spec/unit/default_spec.rb

```ruby
require 'spec_helper'

describe 'mailx::default' do
  context 'on Ubuntu' do
    let(:chef_run) do
      ChefSpec::SoloRunner.new({ :platform => 'ubuntu',
                                 :version => '14.04'}).converge(described_recipe)
    end

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

- Set up a test context for ubuntu

SAVE FILE!

Exercise: Write a Real Test

📄 **OPEN IN EDITOR:** `cookbooks/mailx/spec/unit/default_spec.rb`

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

  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!

Exercise: Run rspec

```
$ 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: Add Cross-platform Attributes

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

```ruby
case node['platform']
  when "ubuntu"
    default['mailx']['mailx-package'] = "mailutils"
  when "centos"
    default['mailx']['mailx-package'] = "mailx"
end
```

SAVE FILE!

Exercise: Install the Package

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

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

SAVE FILE!

Exercise: Run rspec

```
$ 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!
```

Review Questions

- What is ChefSpec used for?

- What tool is ChefSpec based on?

- What directory does do your tests go into?

- Given a recipe named 'backup', what will the ChefSpec test filename be?

Further Resources

What Do I Do Now?

- To learn more you are going to need more practice and more resources. It takes a village and a couple of web servers.

- In this module, we will:

  - Provide you with resources that you can read
  - Provide you with resources that you can watch
  - Provide you with resources that you can listen
  - Provide you with resources that you can attend

The Slides From This Course

- https://goo.gl/cExAi1

- Provided in PDF format

Documentation

- http://docs.chef.io
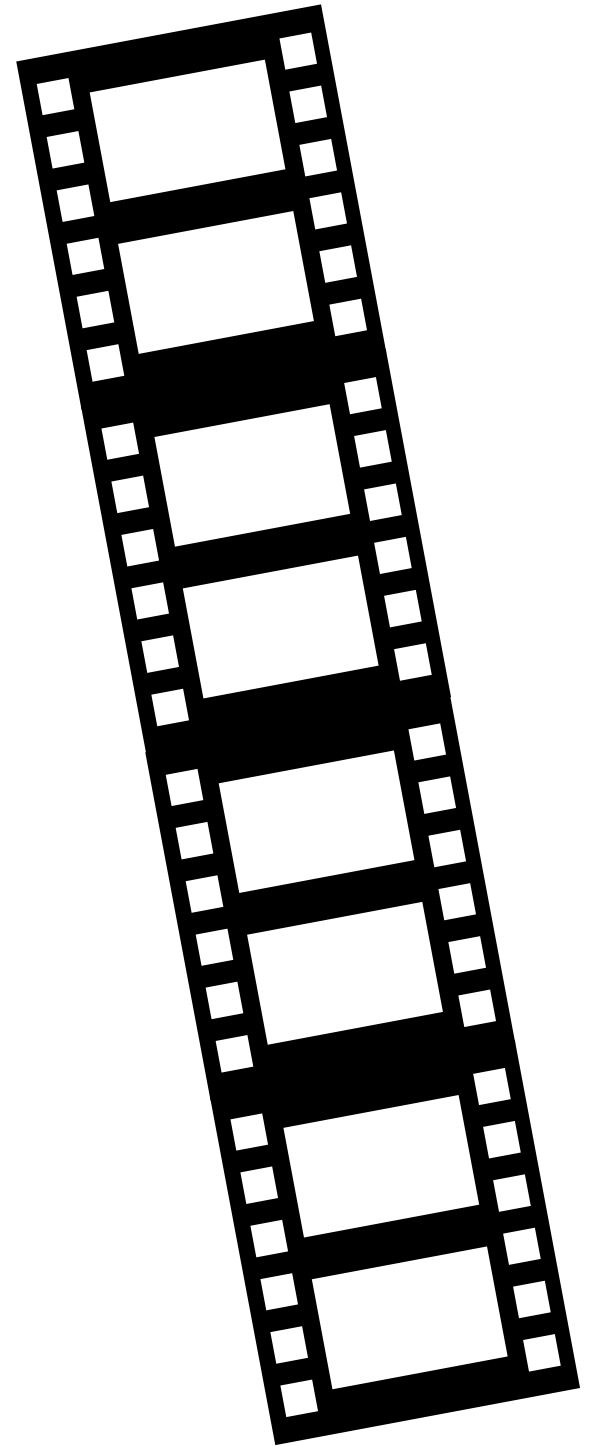
Learning Chef

A Guide to Configuration Management and Automation
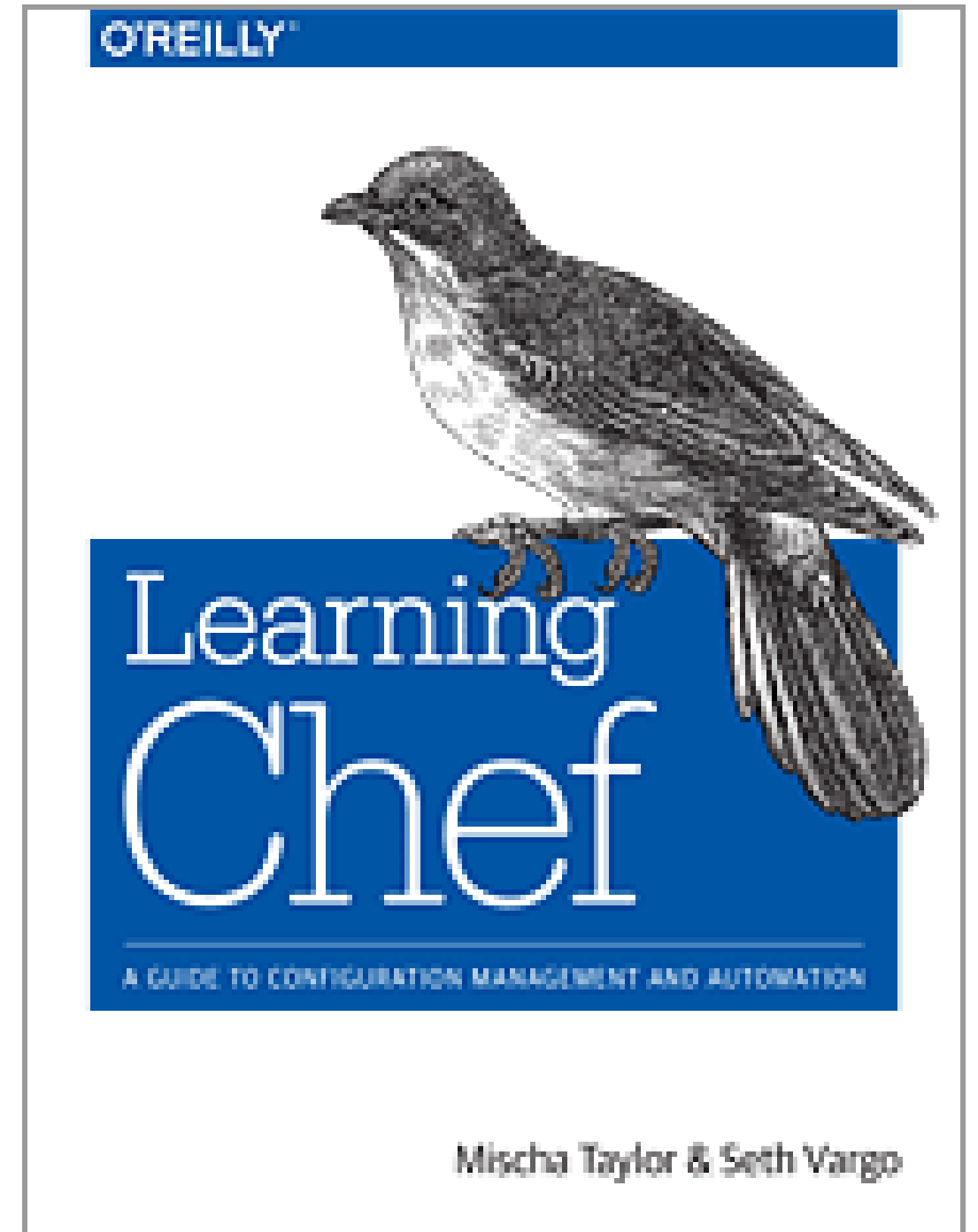
Customizing Chef

Getting the Most Out of Your
Infrastructure Automation
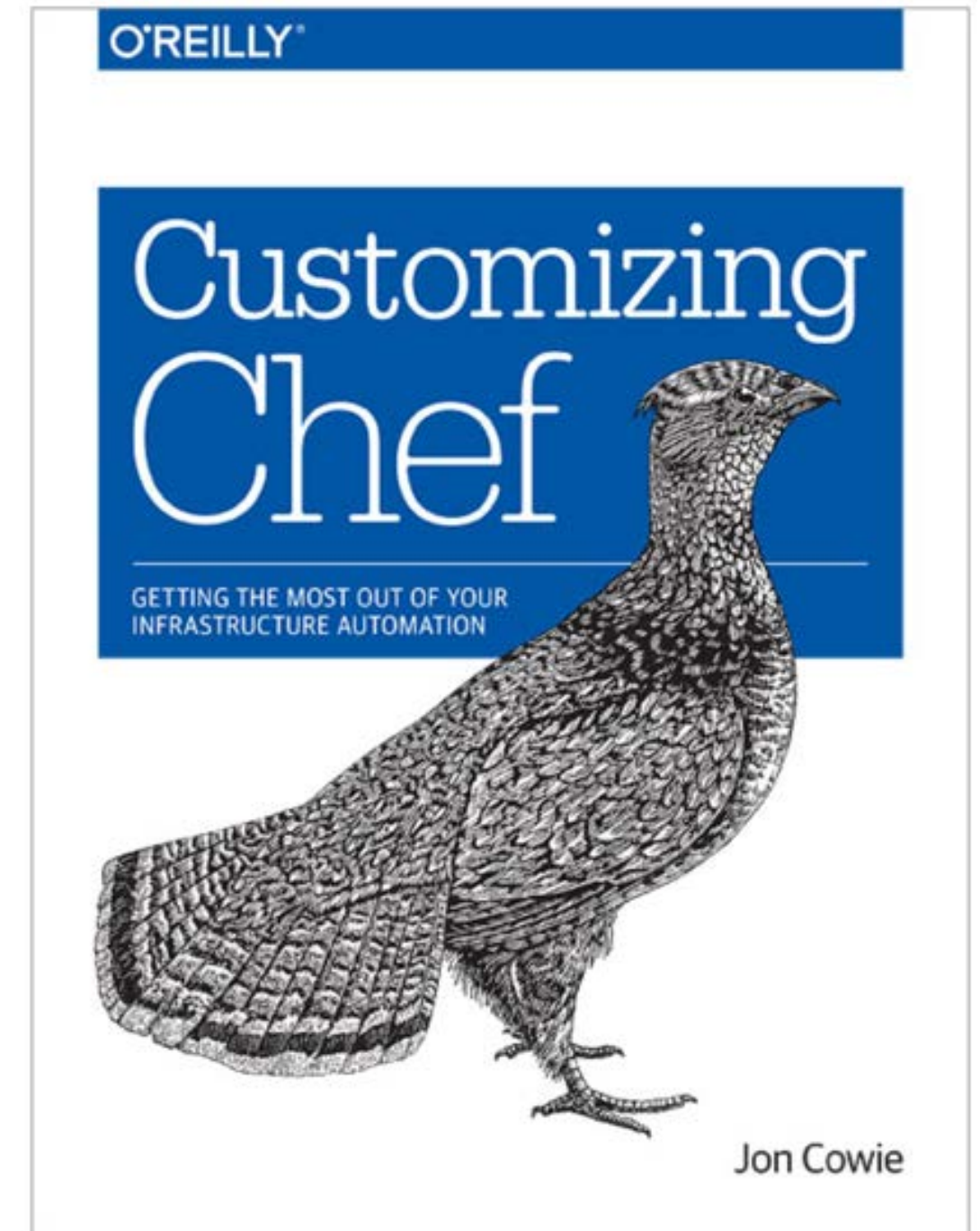
Youtube Channel

- ChefConf Talks

- Training Videos

- https://www.youtube.com/user/getchef/playlists

foodfightshow.org

- http://foodfightshow.org

- Food Fight is a bi-weekly podcast for the Chef community. We bring together the smartest people in the Chef community and the broader DevOps world to discuss the thorniest issues in system administration.

theshipshow.com

- http://theshipshow.com

  The Ship Show is a twice-monthly podcast, featuring discussion on everything from build engineering to devops to release management, plus interviews, new tools and techniques, and reviews.

Chef Developers' IRC Meeting

- Join members of the Chef Community in a meeting for Chef Developers where we'll discuss the future of the Chef project and other things pertinent to the community.

- irc.freenode.net#chef-hacking

- https://github.com/chef/chef-community-irc-meetings
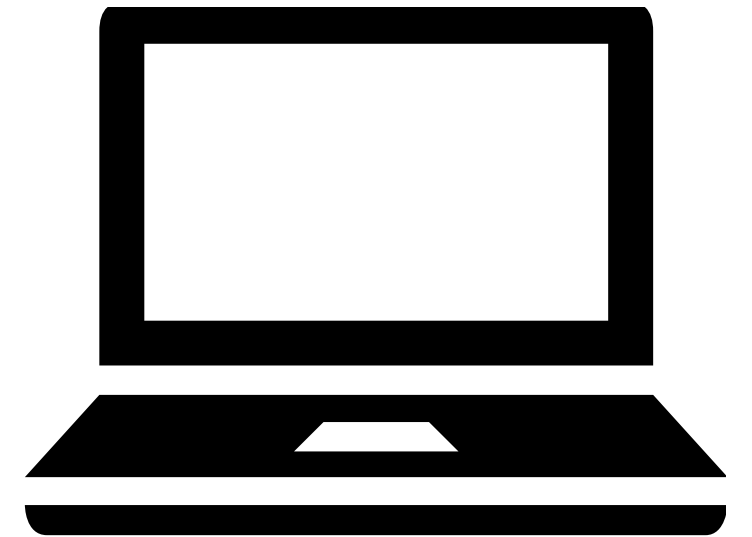
Chef Community Summit

- The Chef Community will gather for two days of open space sessions and brainstorm on Chef best practices.

- The Chef Community Summit is a facilitated Open Space event. The participants of the summit propose topics, organize an agenda, and discuss and work on the ideas that are most important to the community.

- https://www.chef.io/summit
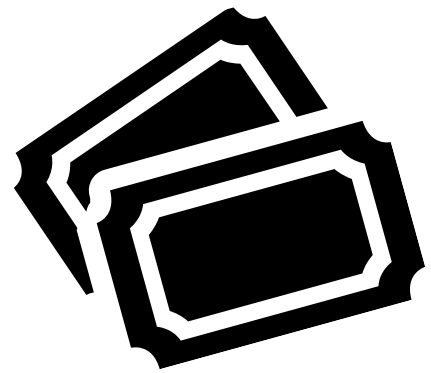
ChefConf

- It's the gathering of hundreds of Chef community members

- We get together to learn about the latest and greatest in the industry (both the hows and the whys), as well as exchange ideas, brainstorm solutions, and give hugs, which has become the calling card of the DevOps community, and the Chef community in particular
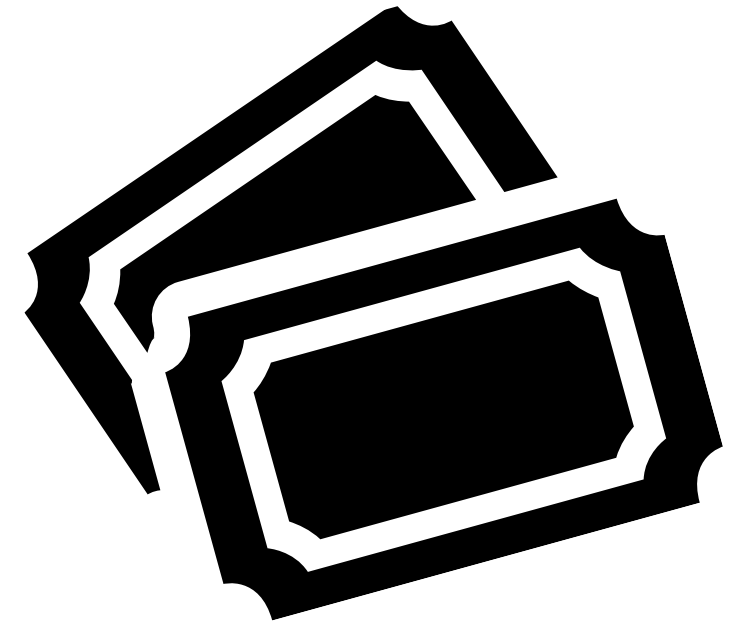
Discussion

What questions can we answer for you?