

# Chef Fundamentals

[training@getchef.com](mailto:training@getchef.com)

Copyright (C) 2013 Chef Software, Inc.

v1.2.3

Monday, 30 June 14

# Introductions

v1.2.3



Monday, 30 June 14

# Instructor Introduction

# Introduce Yourselves

- Name
- Current job role
- Previous job roles/background
- Experience with Chef and/or config management
- Favorite Text Editor

Monday, 30 June 14

Student introduction

\* Name  
\* Current job role  
\* Previous job roles/background  
\* Experience with Chef  
\* Favorite Text Editor

# Course Objectives and Style

v1.2.3



Monday, 30 June 14

# Course Objectives

- Upon completion of this course you will be able to
  - Automate common infrastructure tasks with Chef
  - Describe Chef's architecture
  - Describe Chef's various tools
  - Apply Chef's primitives to solve your problems

# 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

Monday, 30 June 14

Hammer home that the students know how to run their infrastructure

YOU are the best person capable of automating your environment

Chef training is about understanding the framework Chef can provide to solving those problems

\* STUDENT KNOWLEDGE + CHEF TRAINING = VICTORY

# Chef is a Language

- Learning Chef is like learning the basics of a language
- 80% fluency will be reached very quickly
- The remaining 20% just takes practice
- The best way to **learn** Chef is to **use** Chef

Monday, 30 June 14

We will get you fluent

We will do it in a way that maps to what experts do

We will teach you how to find and use the resources that will make you an expert in time

# 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 **after** class

Monday, 30 June 14

I get asked at every class, "will we get the slides" so I thought it prudent to address it here. Why not hand them out at the beginning? To make people type things and learn by muscle memory.

# Agenda

v1.2.3



Monday, 30 June 14

# Topics

- Overview of Chef
- Workstation Setup
- Test Node Setup
- Dissecting your first Chef run
- Introducing the Node object
- Writing your first cookbook

# Topics

- Setting attributes, cookbook metadata, templates
- Idempotency, notifications, template variables
- Data bags, search
- Roles
- Environments
- Using community cookbooks

# Breaks!

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

Monday, 30 June 14

Take 5 minute breaks frequently

# Overview of Chef

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Describe how Chef thinks about Infrastructure Automation
  - Define the following terms:
    - Node
    - Resource
    - Recipe
    - Cookbook
    - Run List
    - Roles
    - Search

# Complexity



<http://www.flickr.com/photos/michaelheiss/3090102907/>

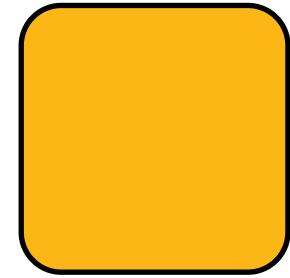
Monday, 30 June 14

Sysadmins, developers, service providers, we have a lot of complexity to manage.

# Items of Manipulation (Resources)

- Networking
- Files
- Directories
- Symlinks
- Mounts
- Registry Key
- Powershell Script
- Users
- Groups
- Packages
- Services
- Filesystems

# A tale of growth...

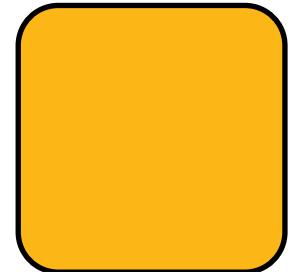


Application

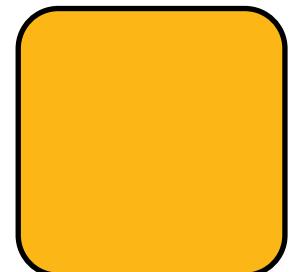
Monday, 30 June 14

You finally got the app running by managing a bunch of those resources

# Add a database

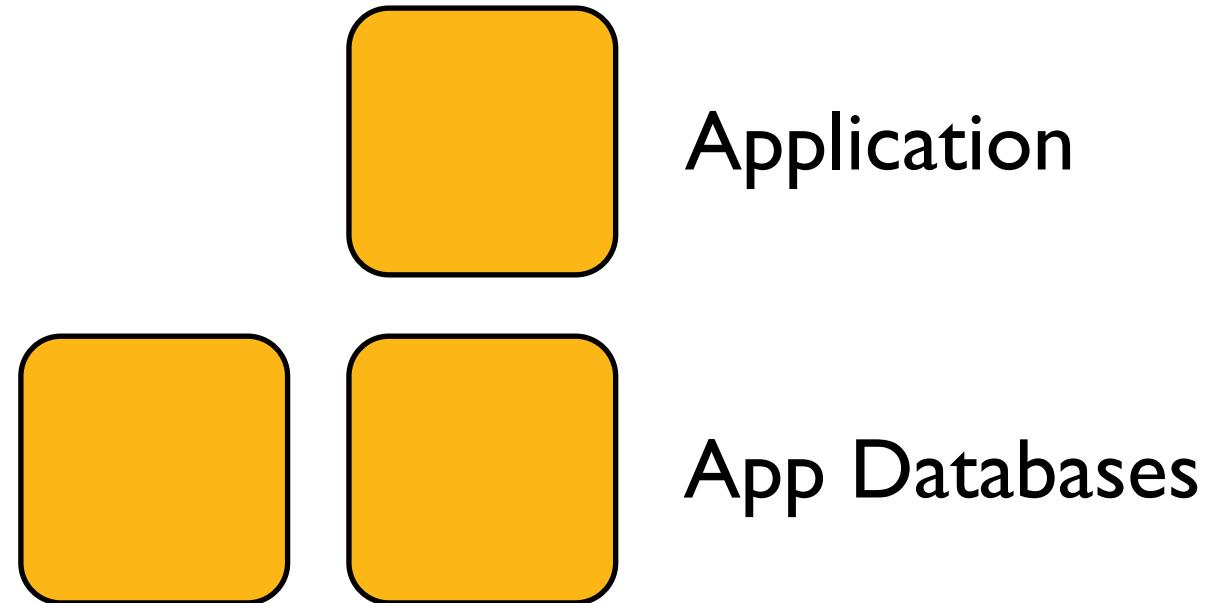


Application



Application Database

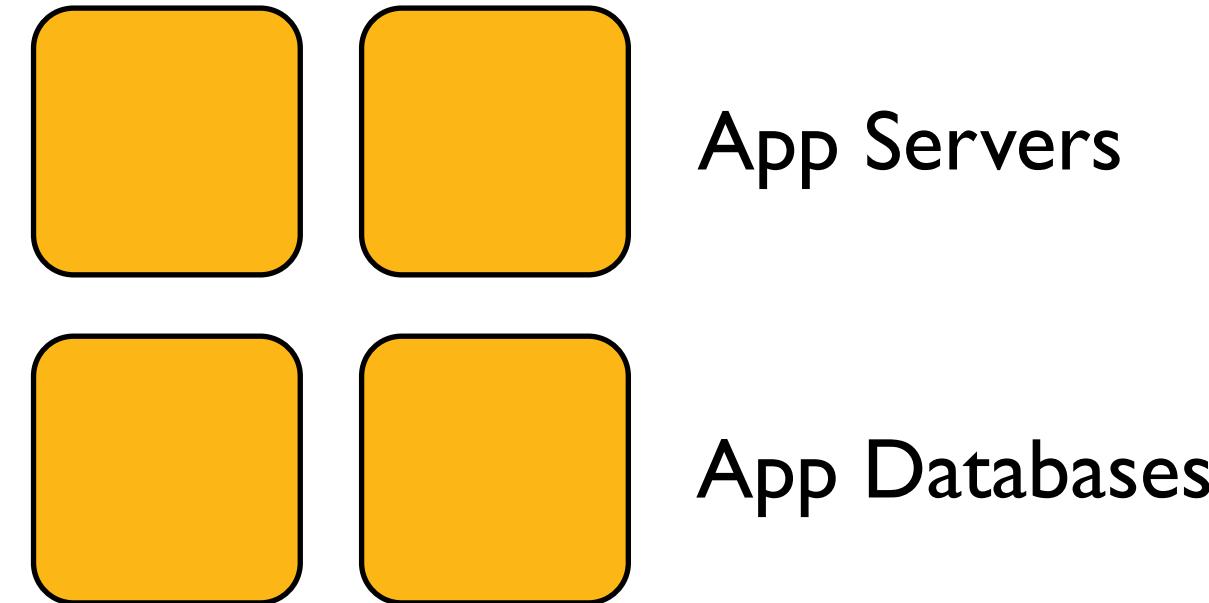
# Make database redundant



Monday, 30 June 14

Of course, we want to have redundant HA database servers

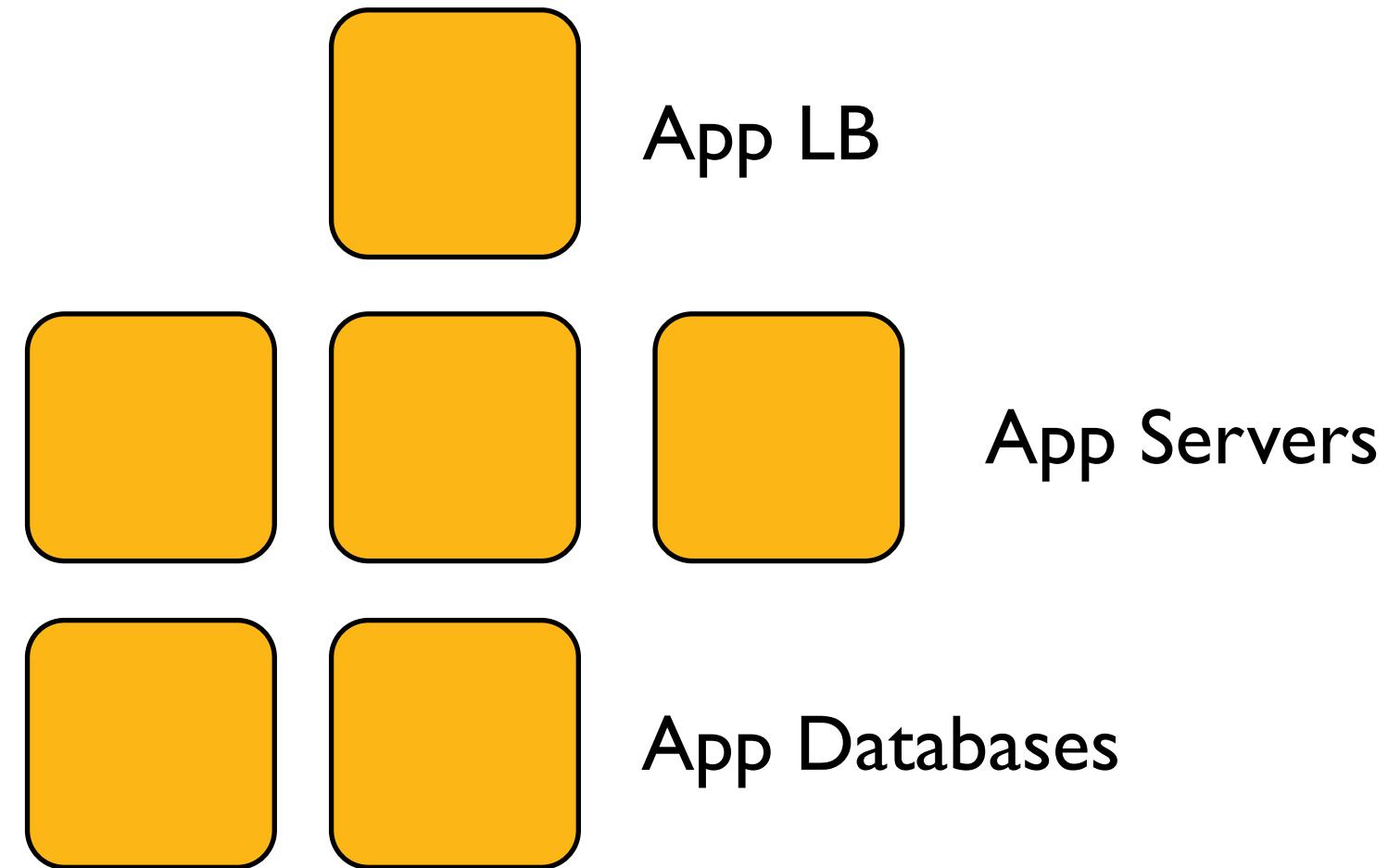
# Application server redundancy



Monday, 30 June 14

We need more app servers, redundancy right?

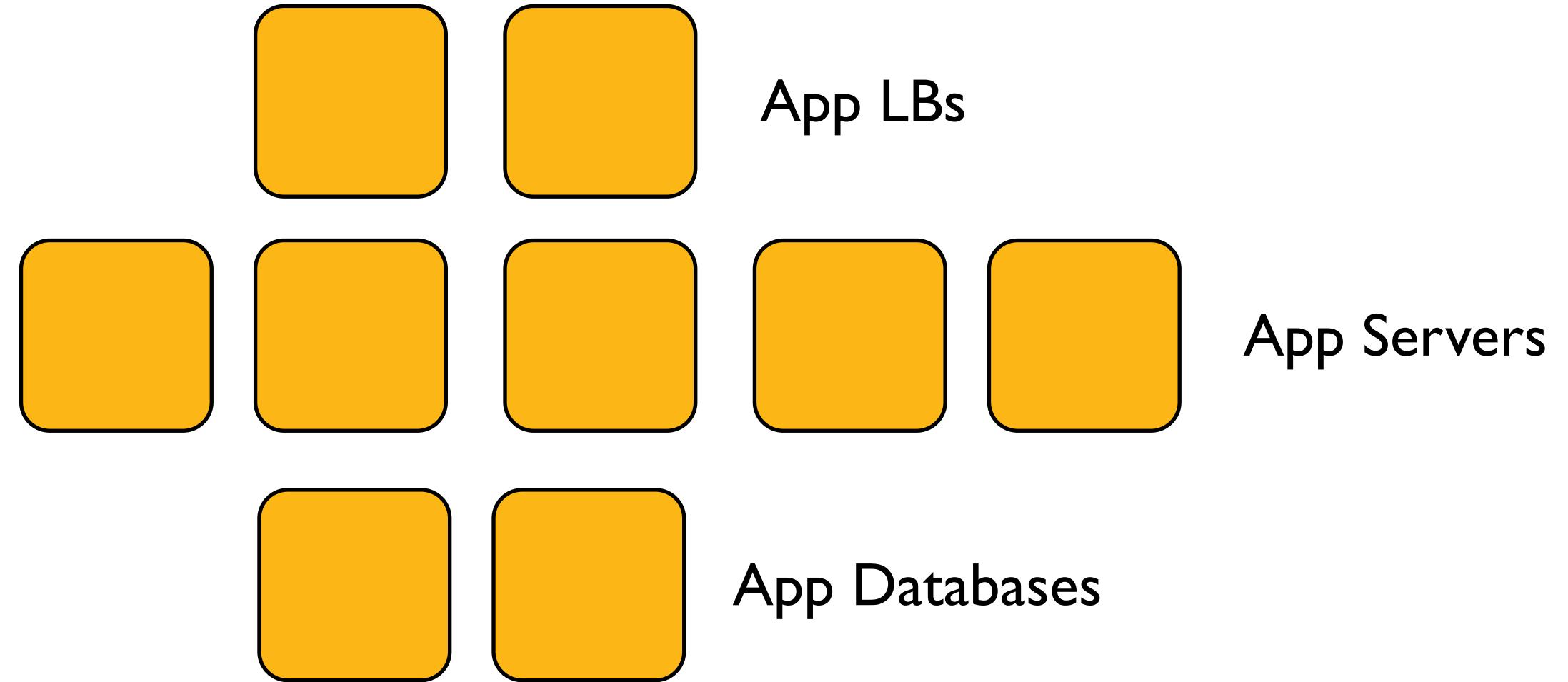
# Add a load balancer



Monday, 30 June 14

We need a load balancer in front of all these app servers

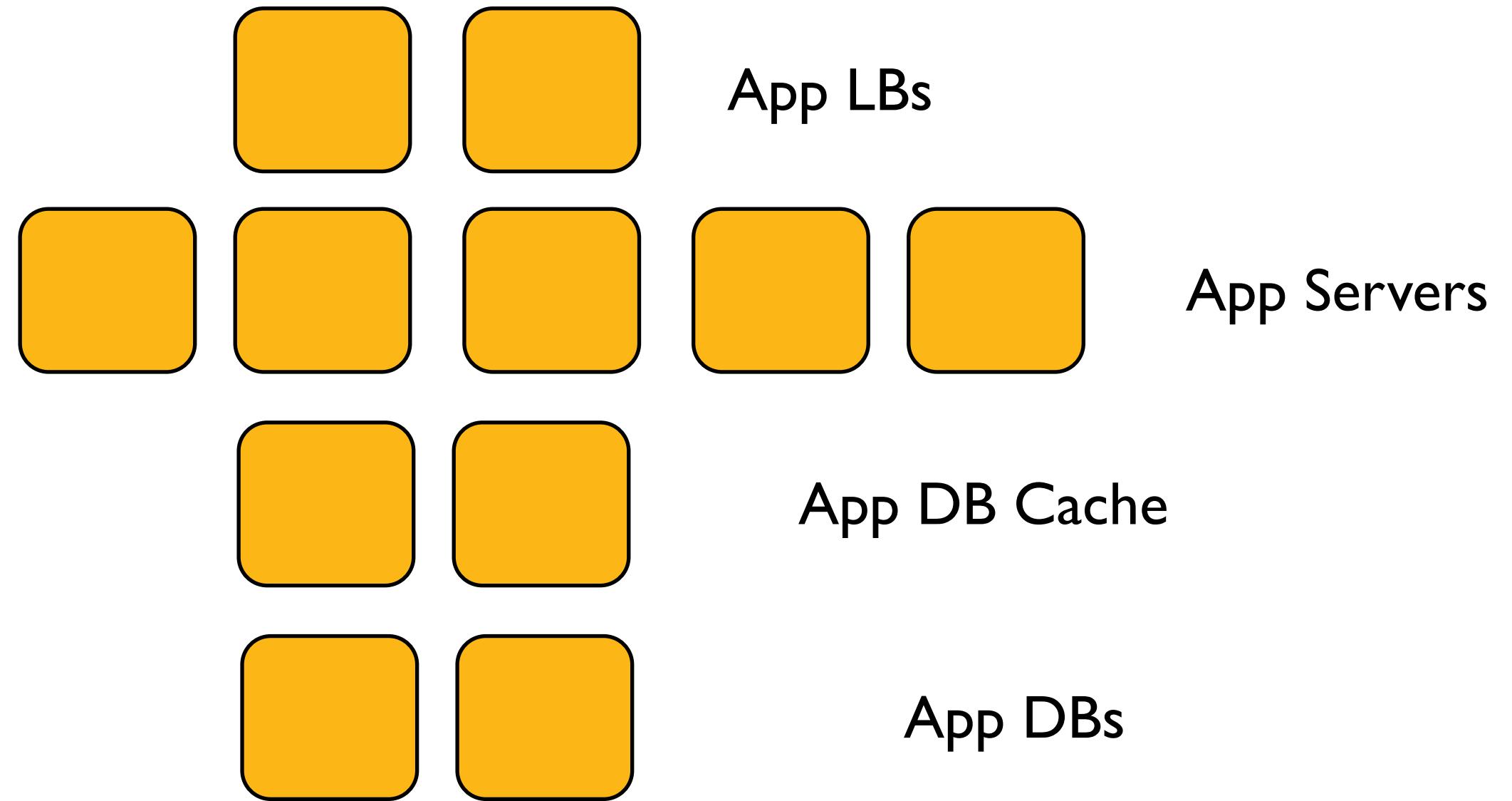
# Webscale!



Monday, 30 June 14

Scaling out the app layer, redundant LB

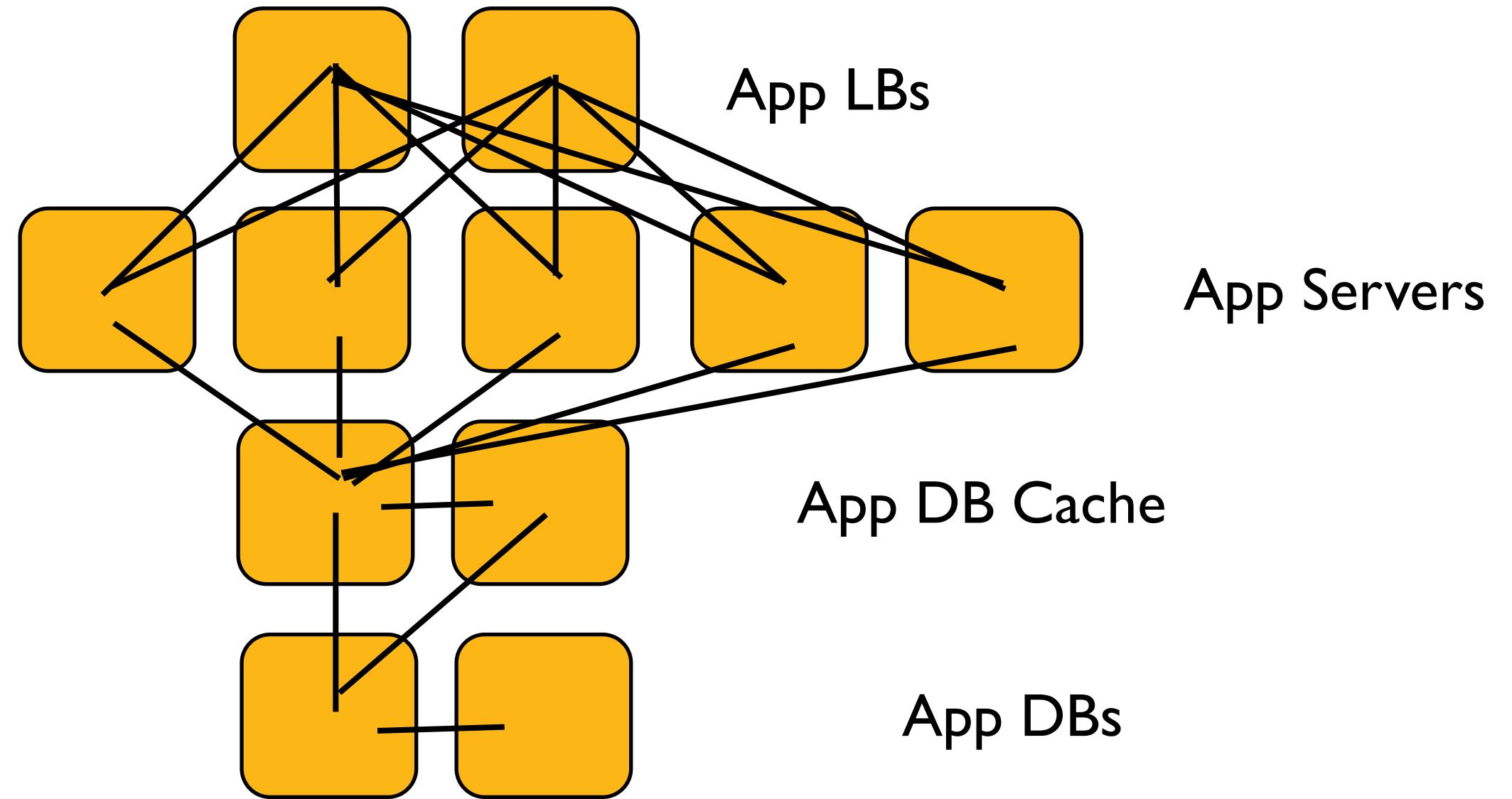
# Now we need a caching layer



Monday, 30 June 14

Posted on slashdot, hacker news or techcrunch! We must construct additional pylons

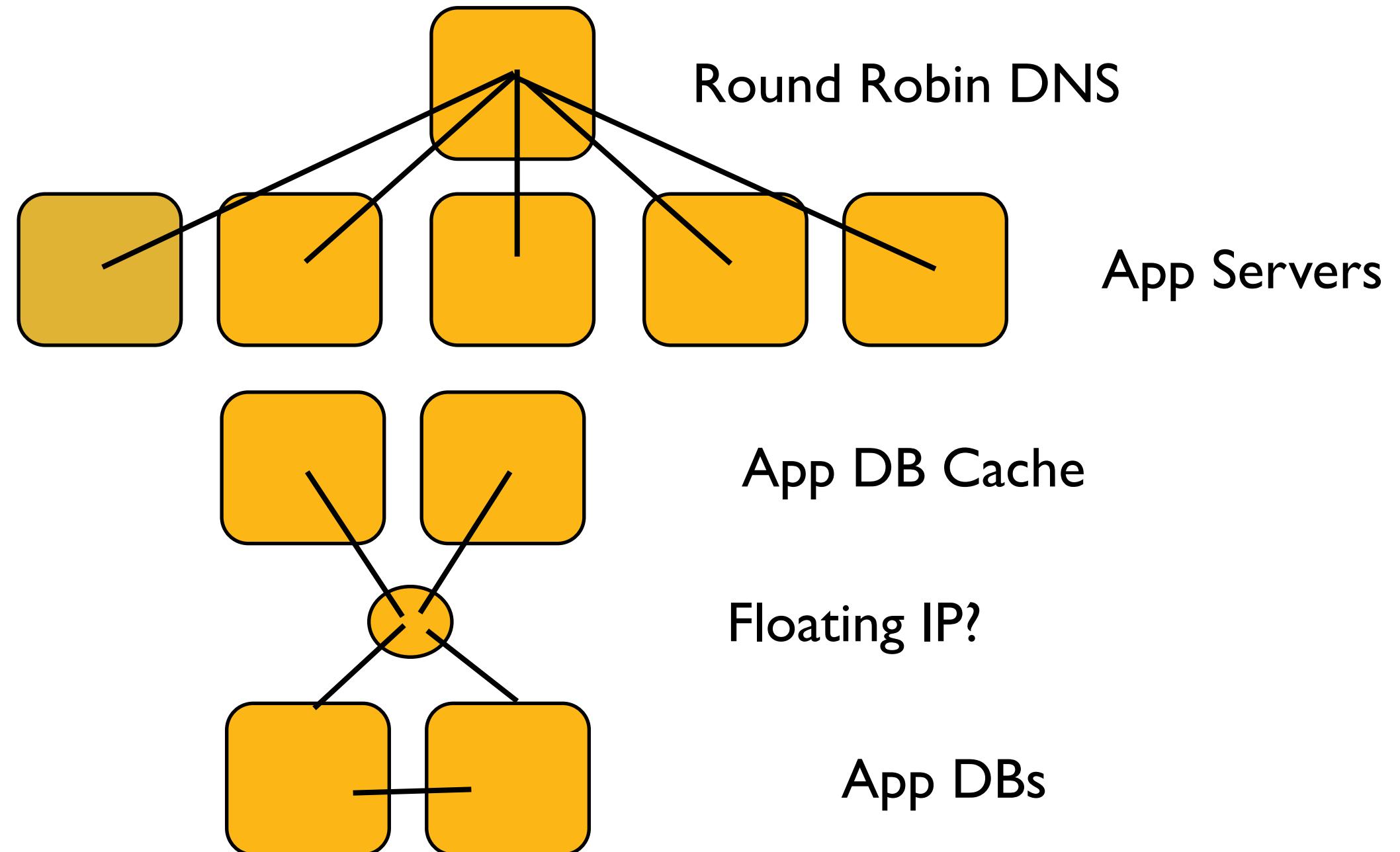
# Infrastructure has a Topology



Monday, 30 June 14

an Infrastructure has a Topology. Maybe you don't want to do it that way.

# Your Infrastructure is a Snowflake

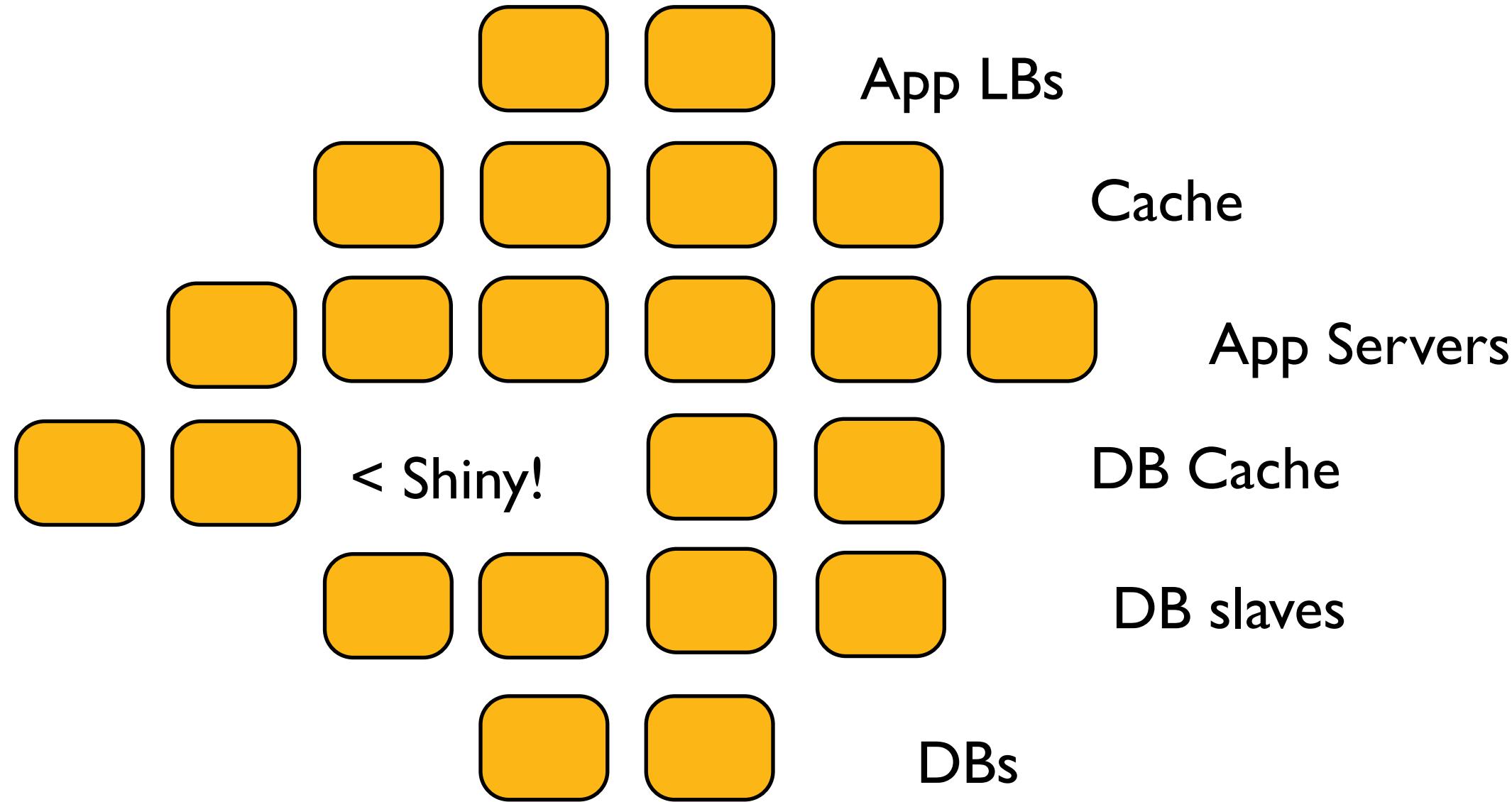


Monday, 30 June 14

How should I know. It's your application.

Your application is unique, and so is your infrastructure.  
They evolve symbiotically.

# Complexity Increases Quickly

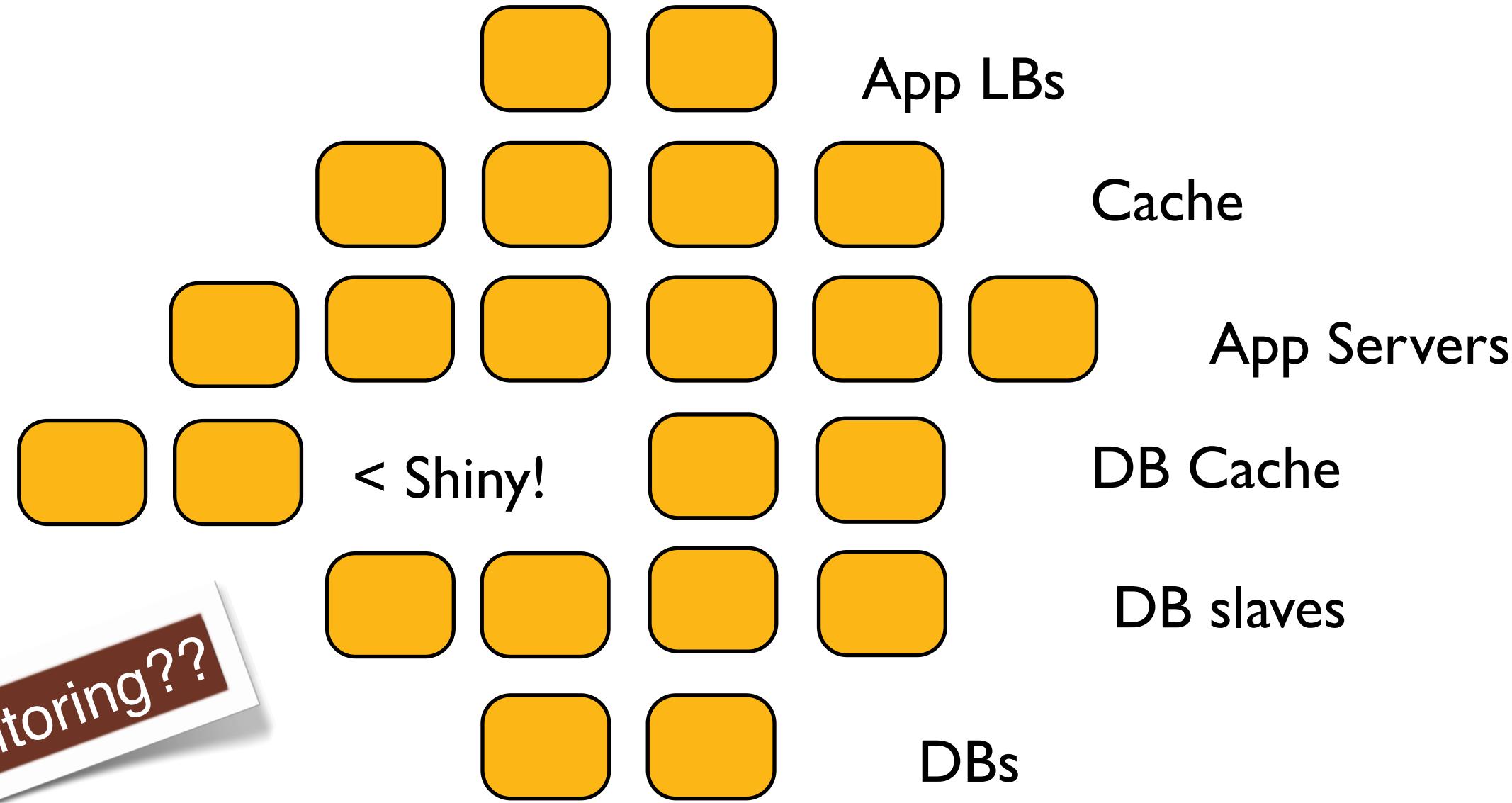


Monday, 30 June 14

This complexity comes in the form of:

- Resolving technical debt
- Business requirements
- Shiny object syndrome

# Complexity Increases Quickly

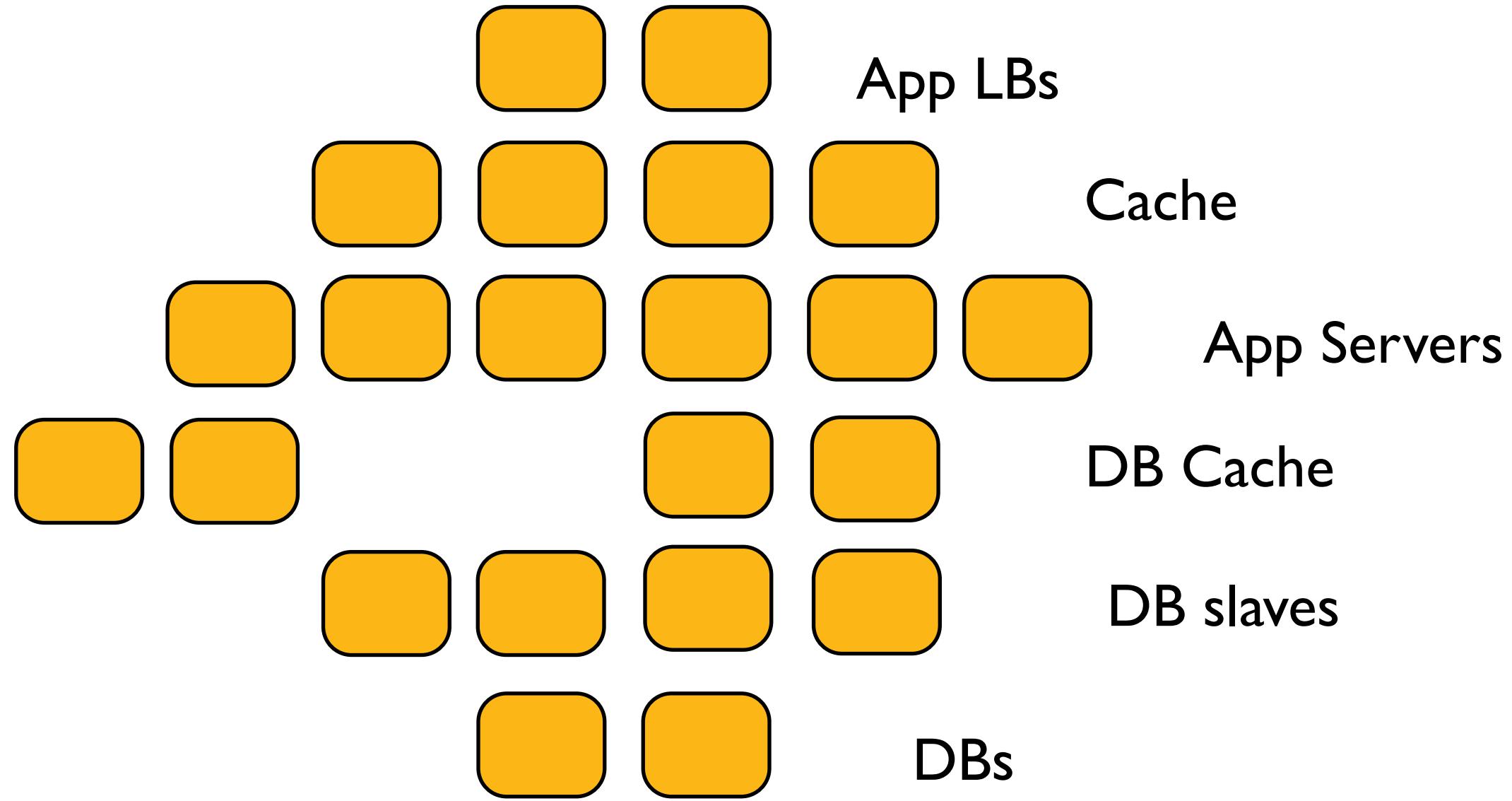


Monday, 30 June 14

This complexity comes in the form of:

- Resolving technical debt
- Business requirements
- Shiny object syndrome

# ...and change happens!

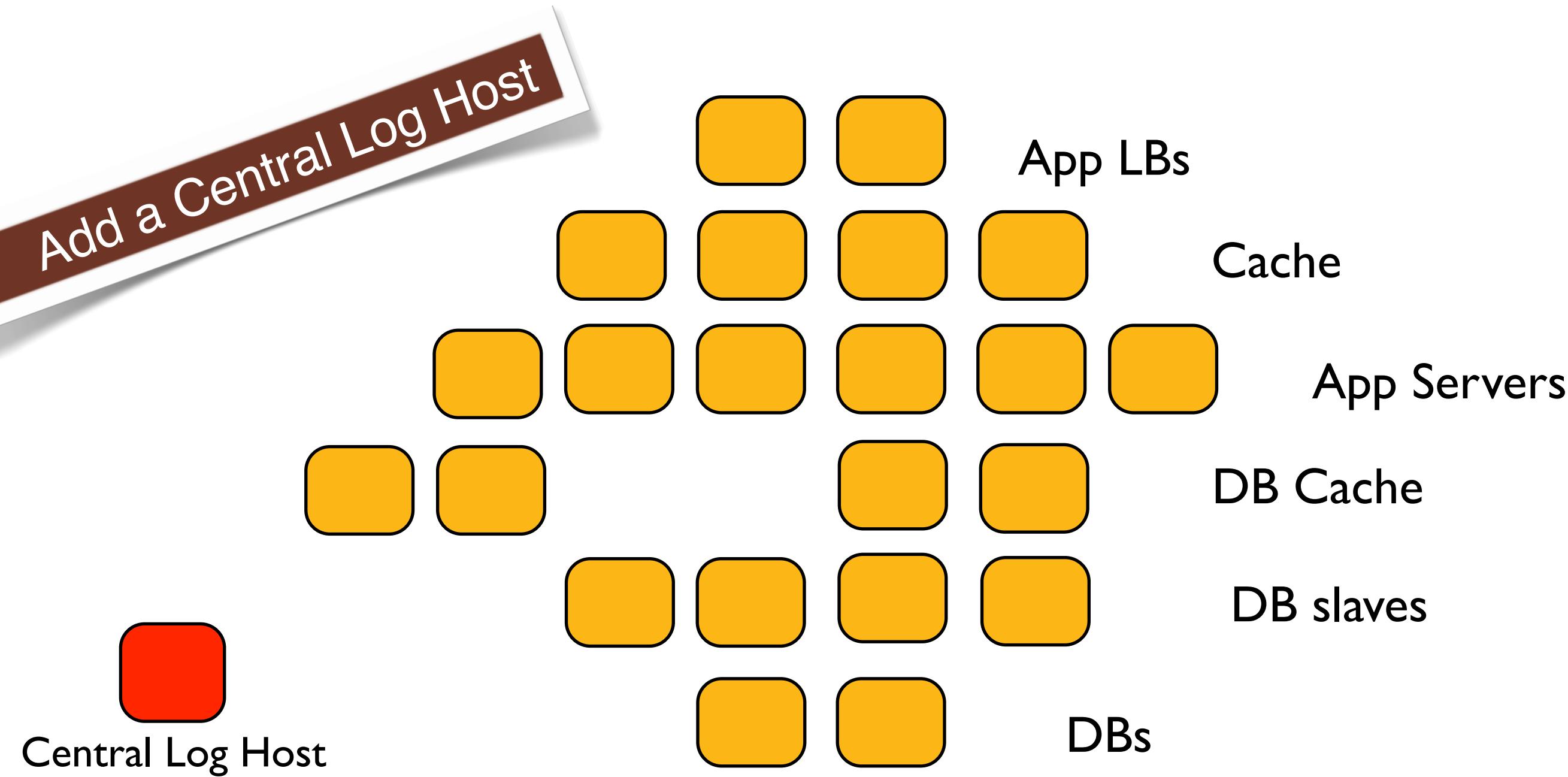


Monday, 30 June 14

It's not just about growing your infrastructure.

Your infrastructure will also need to change over time.

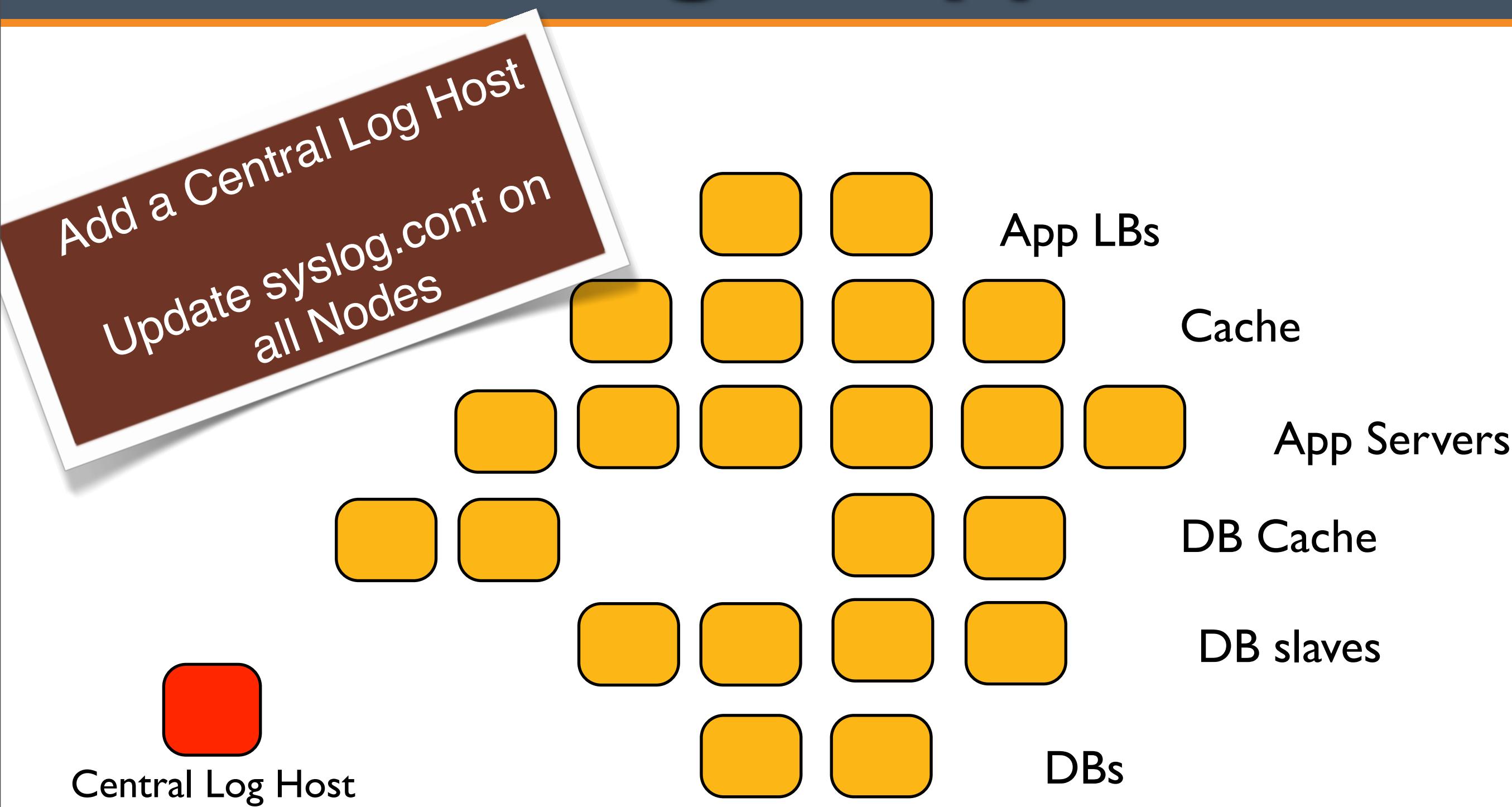
# ...and change happens!



Monday, 30 June 14

Add a central log host

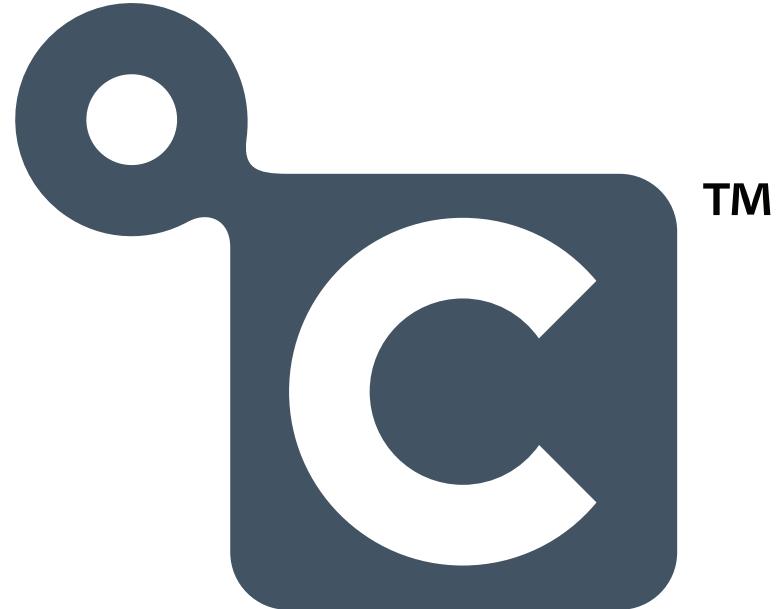
# ...and change happens!



Monday, 30 June 14

Now you need to change syslog.conf on all nodes in your infrastructure to log all kernel messages to the central log host

# Chef Solves This Problem



# Chef

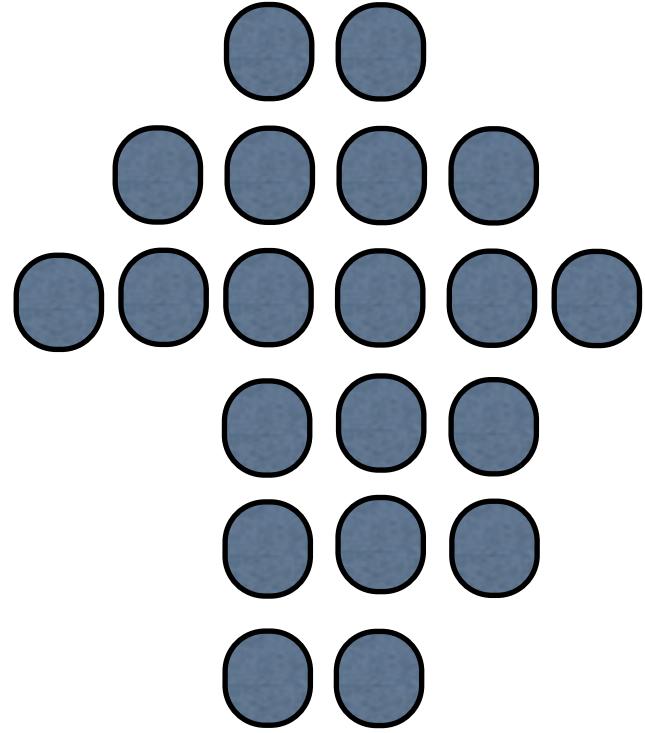
- But you already guessed that, didn't you?

# Managing Complexity

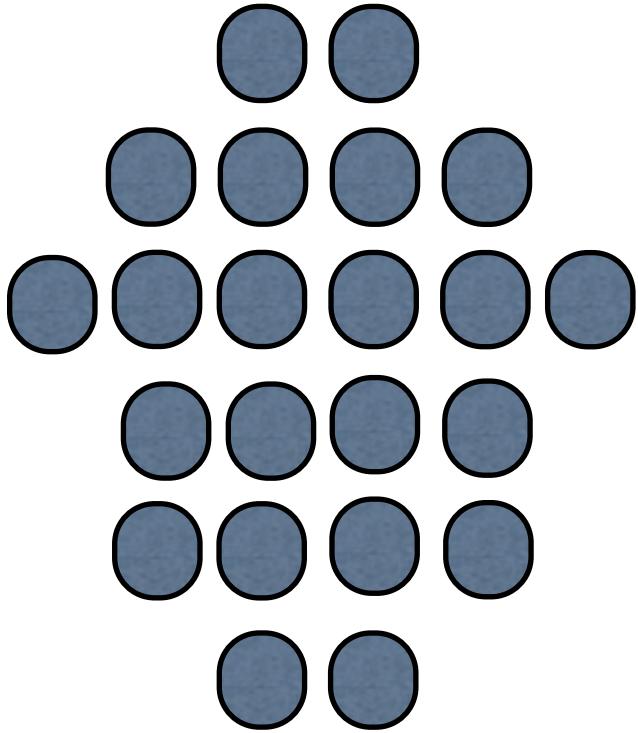
- Organizations
- Environments
- Roles
- Nodes
- Recipes
- Cookbooks
- Search

# Organizations

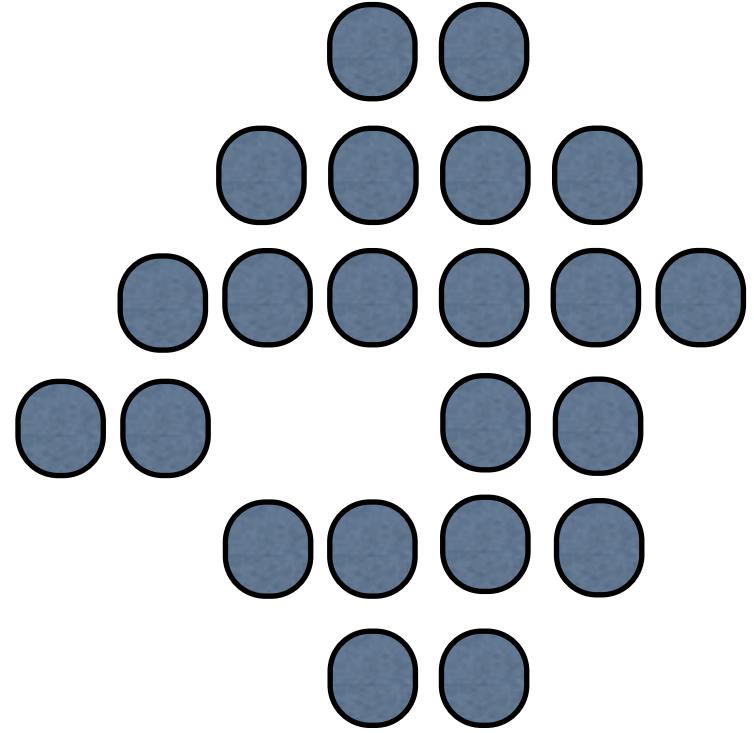
My Infrastructure



Your Infrastructure



Their Infrastructure

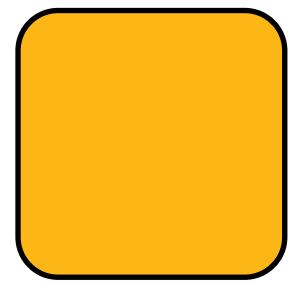


# Organizations

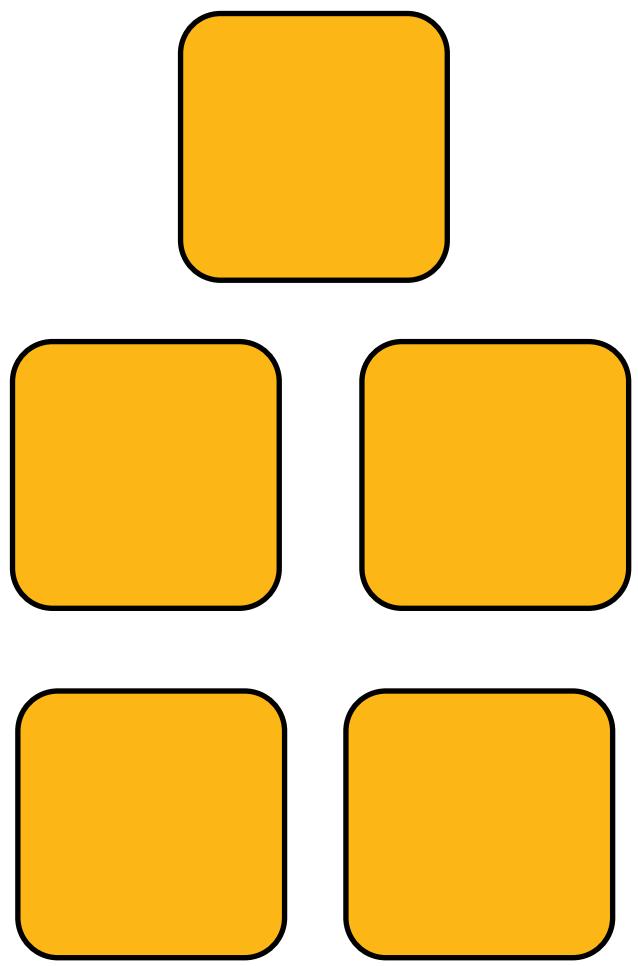
- Completely independent tenants of Enterprise Chef
- Share nothing with other organizations
- May represent different
  - Companies
  - Business Units
  - Departments

# Environments

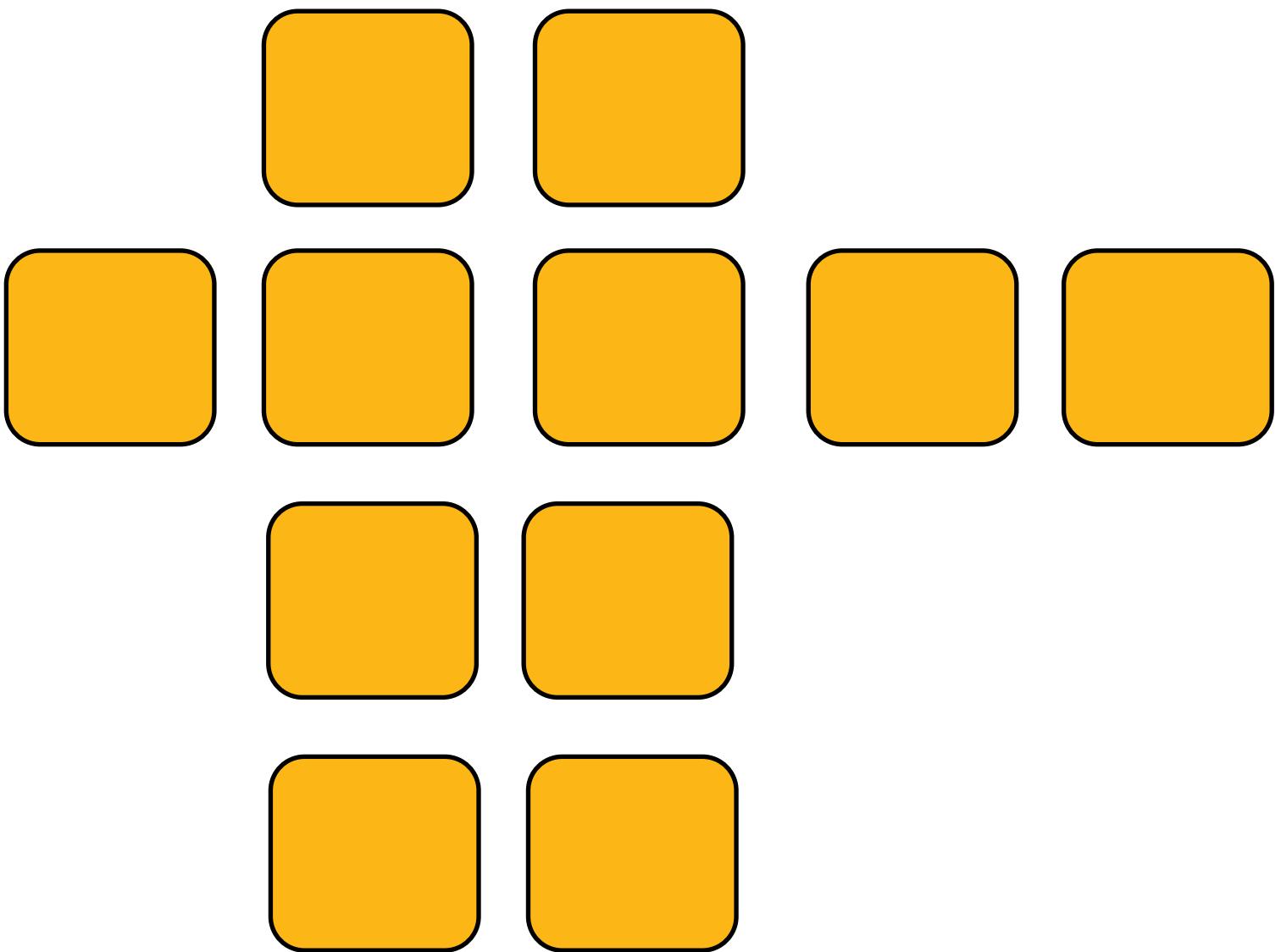
Development



Staging



Production



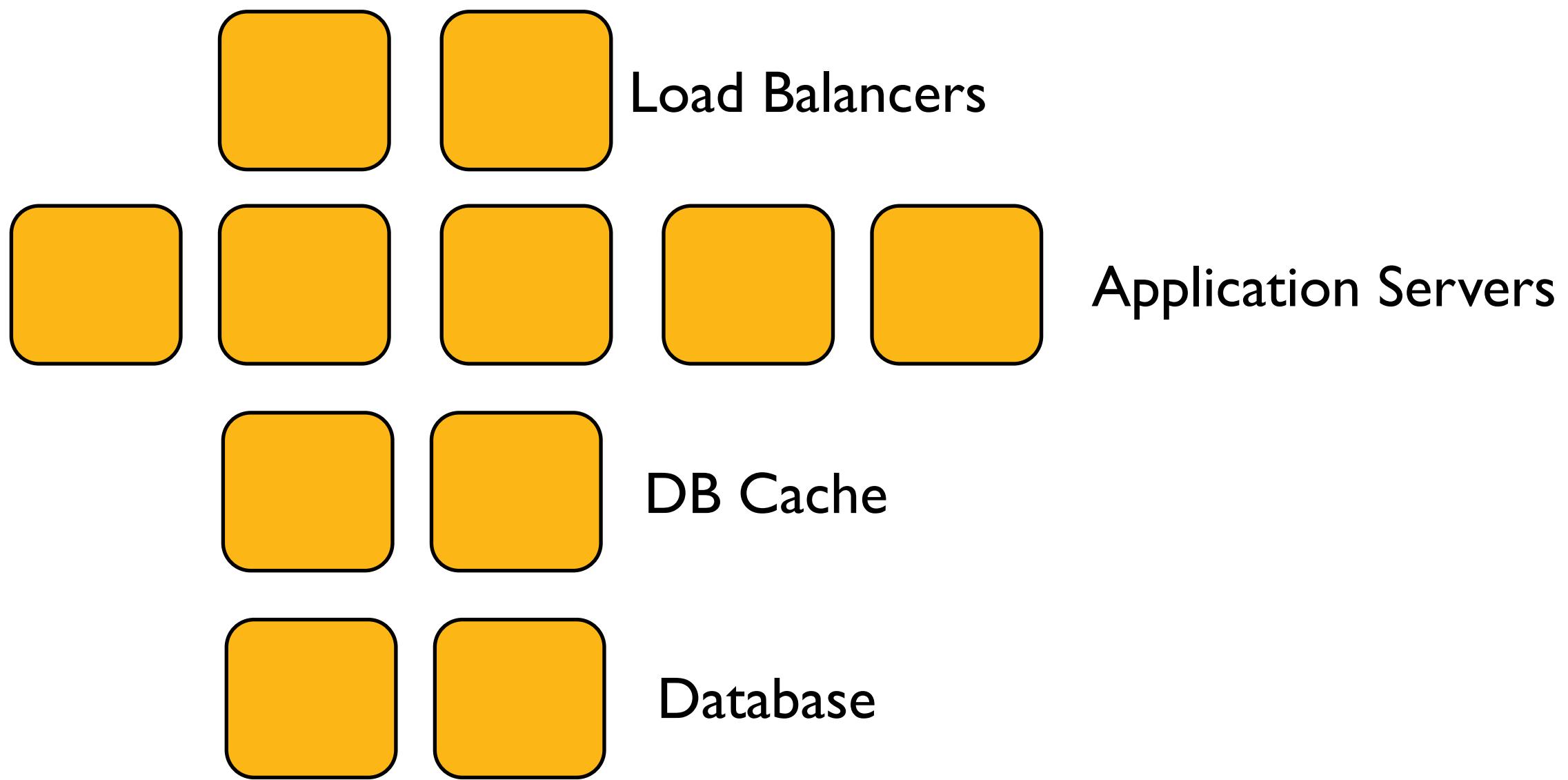
# Environments

- Model the life-stages of your applications
- Every Organization starts with a single environment
- Environments to reflect your patterns and workflow
  - Development
  - Test
  - Staging
  - Production
  - etc.

# Environments Define Policy

- Environments may include data attributes necessary for configuring your infrastructure
  - The URL of your payment service's API
  - The location of your package repository
  - The version of the Chef configuration files that should be used

# Roles



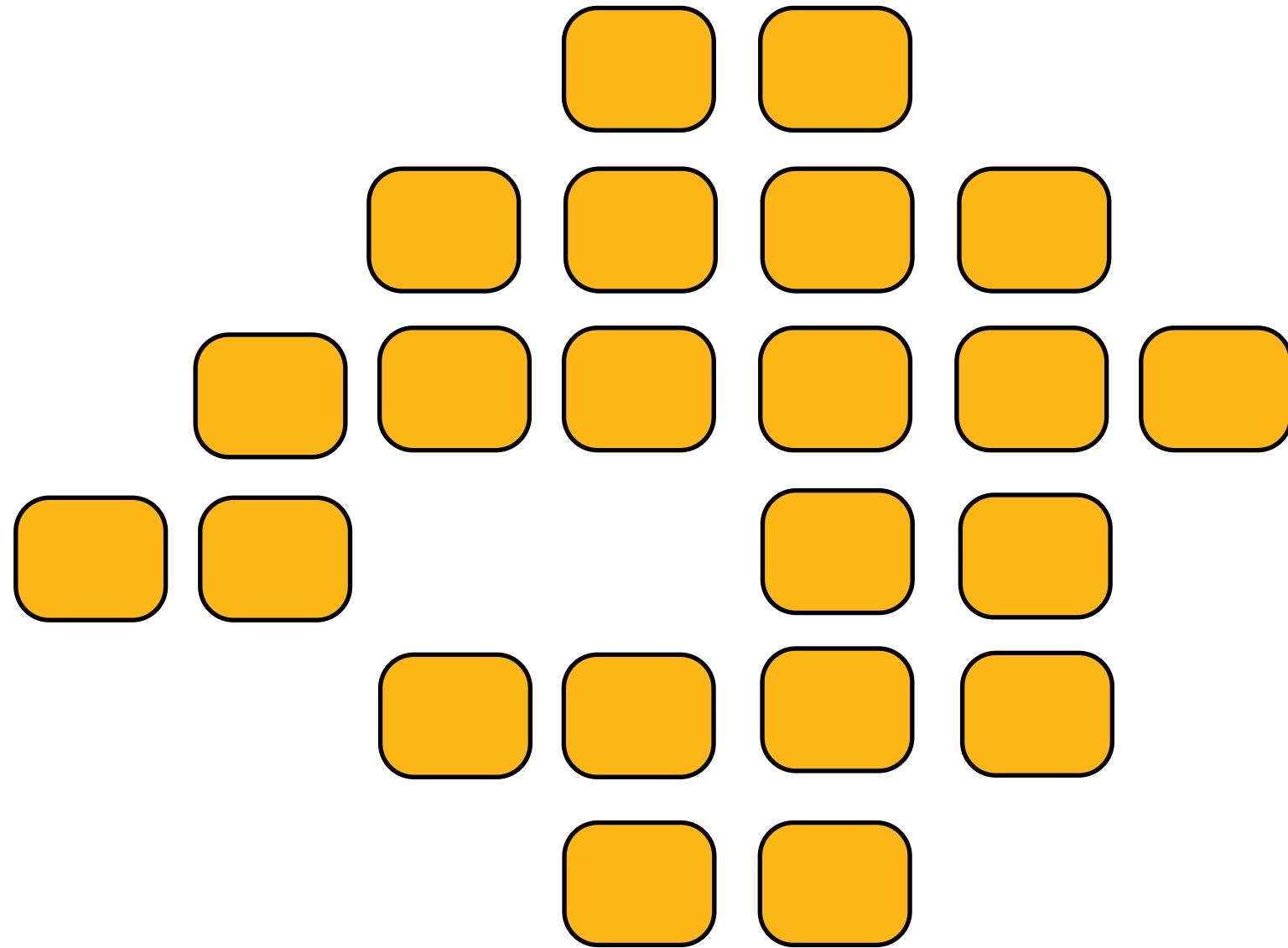
# Roles

- Roles represent the types of servers in your infrastructure
  - Load Balancer
  - Application Server
  - Database Cache
  - Database
  - Monitoring

# Roles Define Policy

- Roles may include a list of Chef configuration files that should be applied.
  - We call this list a Run List
- Roles may include data attributes necessary for configuring your infrastructure
  - The port that the application server listens on
  - A list of applications that should be deployed

# Nodes



# Nodes

- Nodes represent the servers in your infrastructure
- Nodes may represent physical servers or virtual servers
- Nodes may represent hardware that you own or may represent compute instances in a public or private cloud

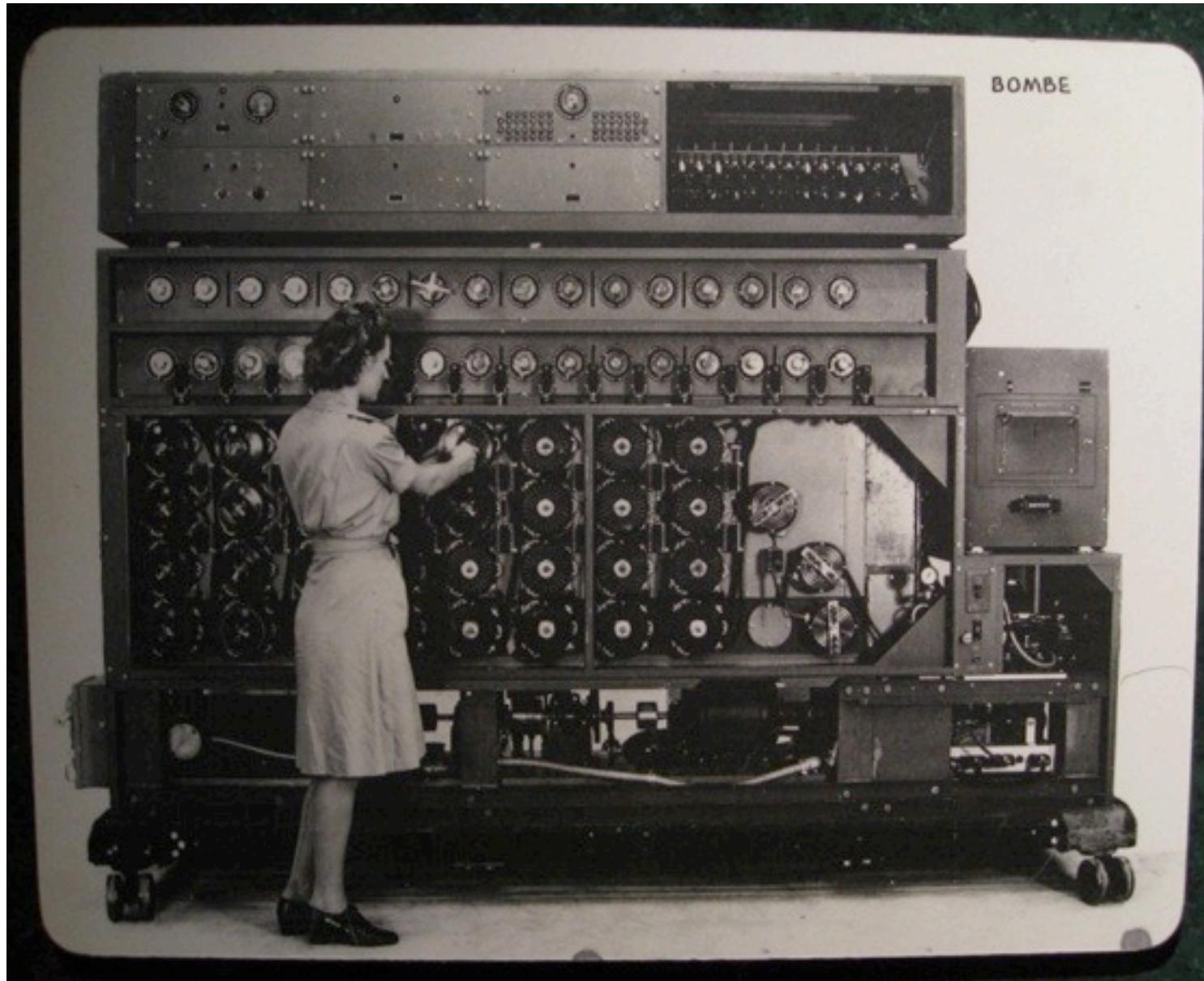
# Node

- Each Node will
  - belong to one Organization
  - belong to one Environment
  - have zero or more Roles

# Nodes Adhere to Policy

- An application, the chef-client, runs on each node
- chef-client will
  - gather current system configuration
  - download the desired system configuration from the Chef server
  - configure the node such that it adheres to the policy

# Chef is Infrastructure as Code



<http://www.flickr.com/photos/louisb/4555295187/>

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

# Configuration Code

- Chef ensures each Node complies with the policy
- Policy is determined by the configurations included in each Node's run list
- Reduce management complexity through abstraction
- Store the configuration of your infrastructure in version control

# Declarative Interface to Resources

- You define the policy in your Chef configuration
- Your policy states what state each resource should be in, but not how to get there
- Chef-client will pull the policy from the Chef Server and enforce the policy on the Node

# Resources

- A Resource represents a piece of the system and its desired state
  - A package that should be installed
  - A service that should be running
  - A file that should be generated
  - A cron job that should be configured
  - A user that should be managed
  - and more

# Resources in Recipes

- Resources are the fundamental building blocks of Chef configuration
- Resources are gathered into Recipes
- Recipes ensure the system is in the desired state

# Recipes

- Configuration files that describe resources and their desired state
- Recipes can:
  - Install and configure software components
  - Manage files
  - Deploy applications
  - Execute other recipes
  - and more

# Example Recipe

```
package "apache2"
```

```
  template "/etc/apache2/apache2.conf" do
    source "apache2.conf.erb"
    owner "root"
    group "root"
    mode "0644"
    variables(:allow_override => "All")
    notifies :reload, "service[apache2]"
  end
```

```
  service "apache2" do
    action [ :enable, :start ]
    supports :reload => true
  end
```

# Cookbooks

- Recipes are stored in Cookbooks
- Cookbooks contain recipes, templates, files, custom resources, etc
- Code re-use and modularity

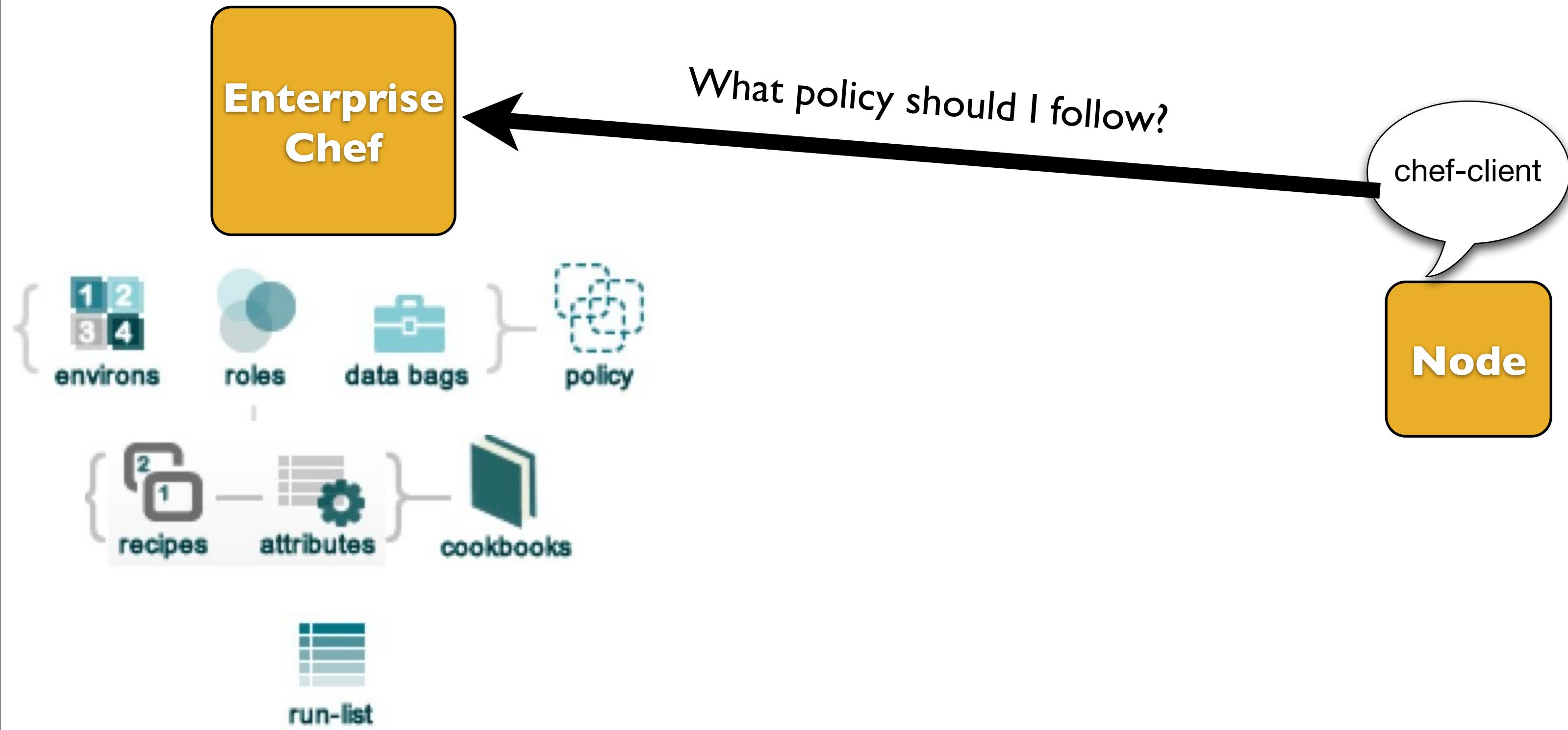


<http://www.flickr.com/photos/shutterhacks/4474421855/>

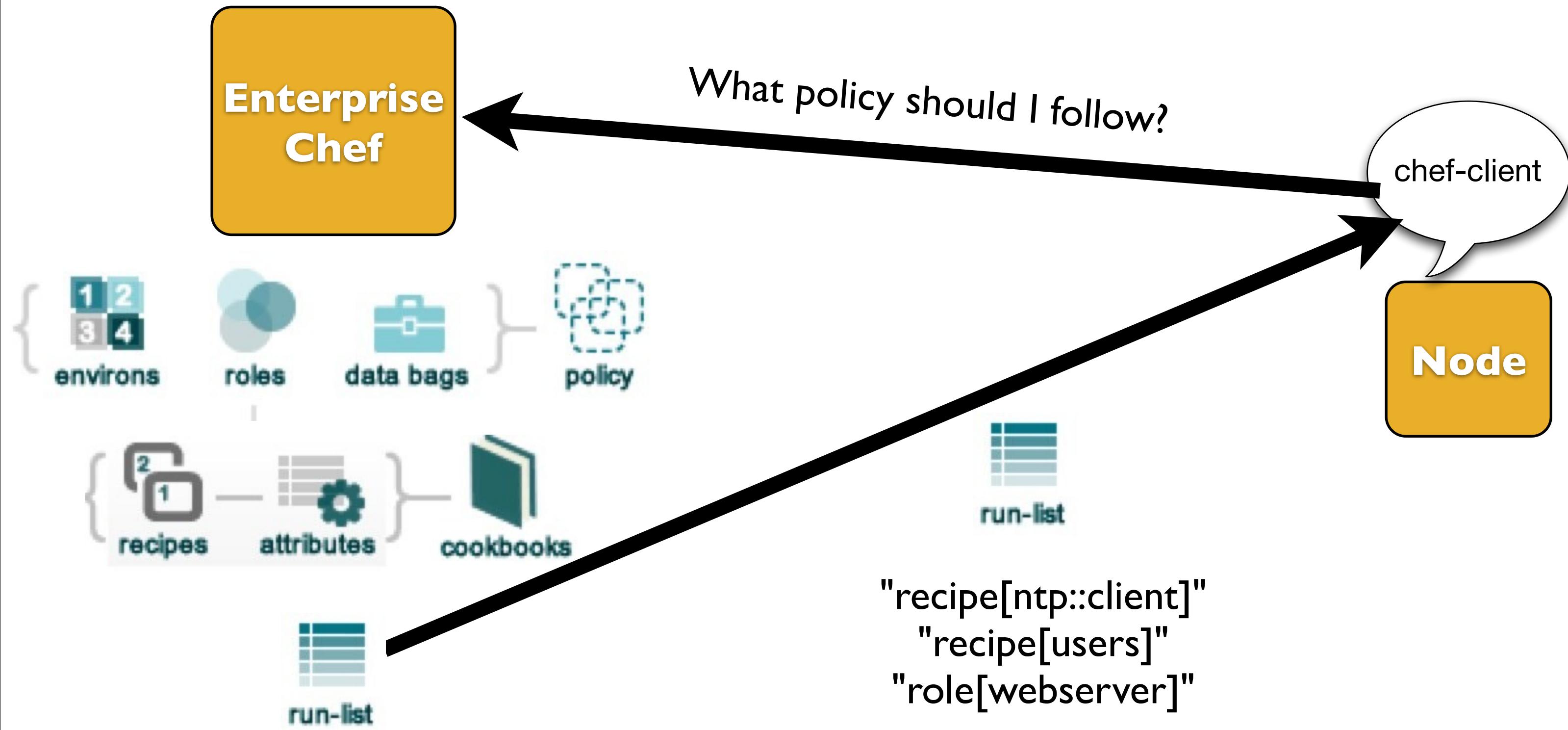
Monday, 30 June 14

LEAD IN: How are these recipes applied to a node? (RUN LIST!)

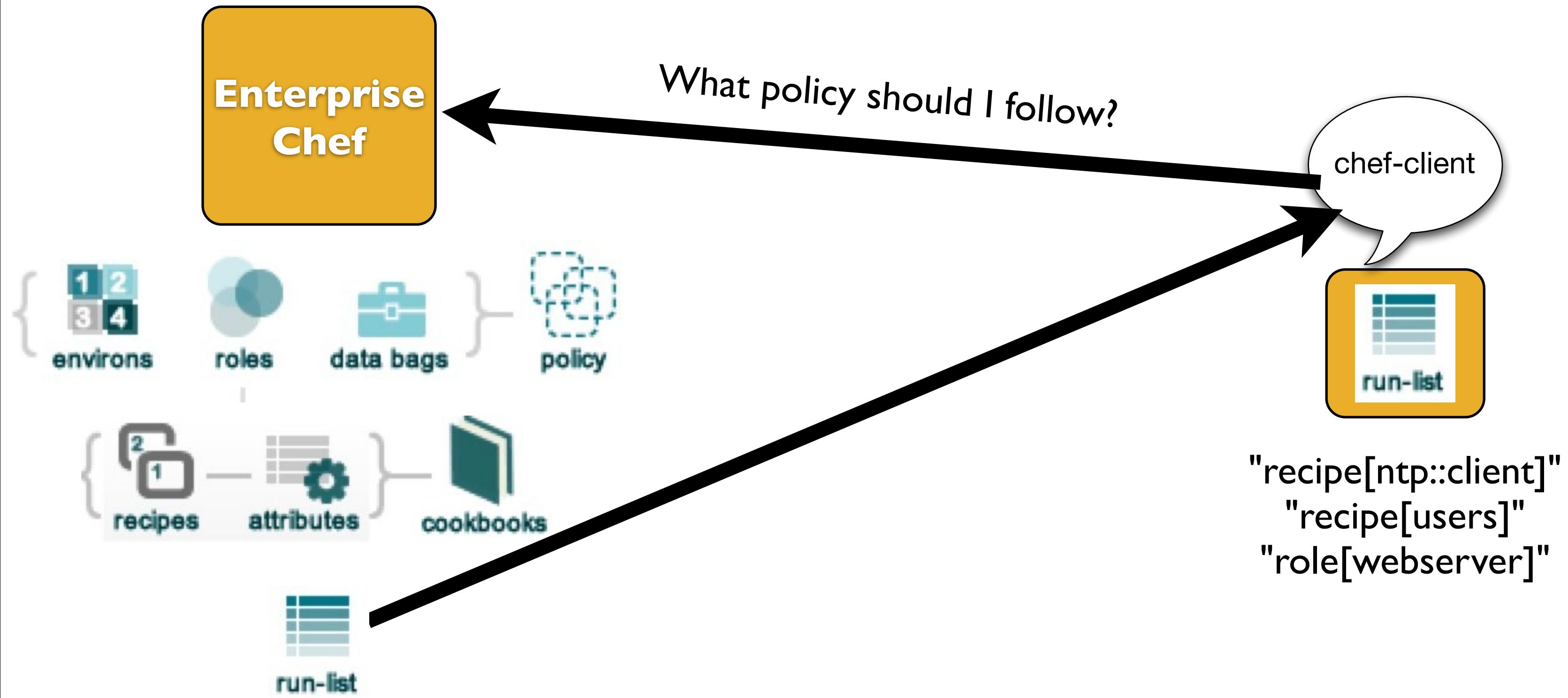
# Run List



# Run List



# Run List

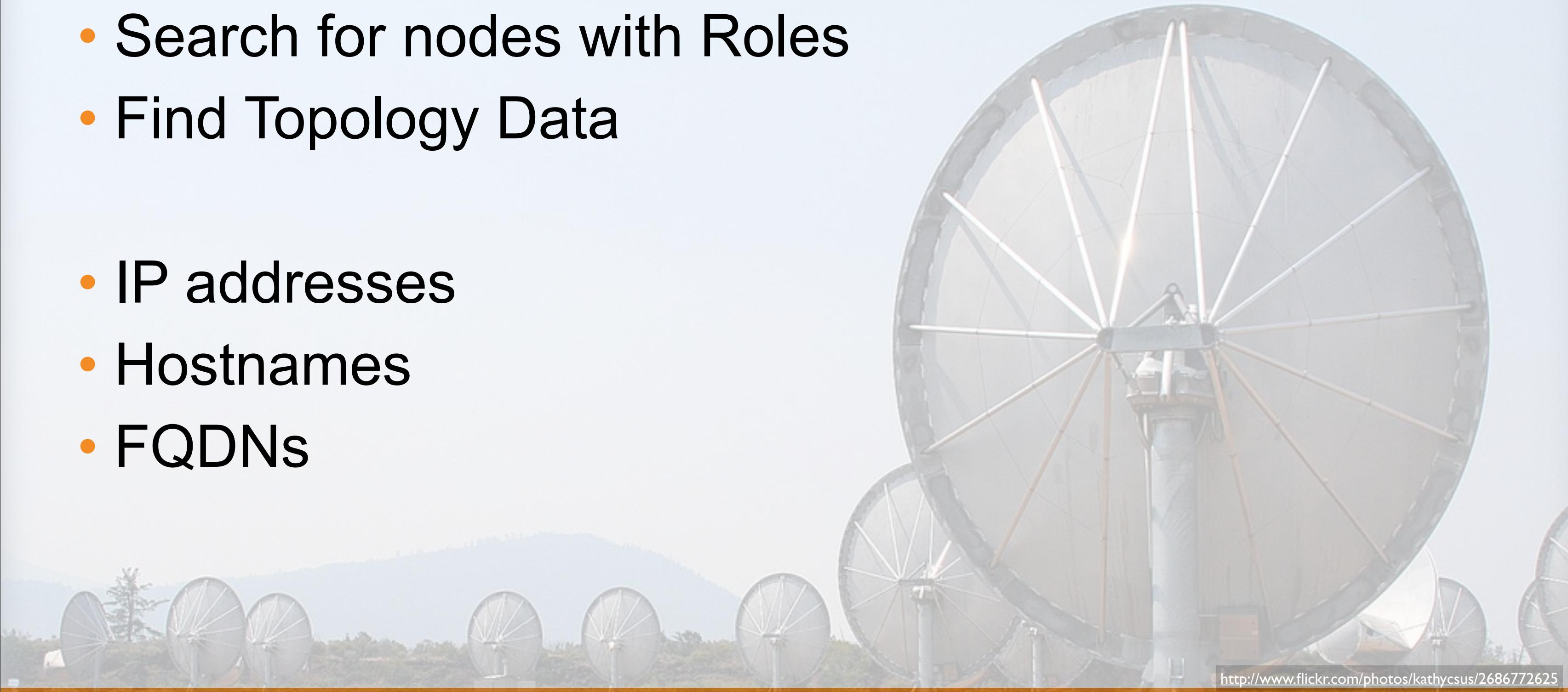


# Run List Specifies Policy

- The Run List is a collection of policies that the Node should follow.
- Chef-client obtains the Run List from the Chef Server
- Chef-client ensures the Node complies with the policy in the Run List

# Search

- Search for nodes with Roles
  - Find Topology Data
- 
- IP addresses
  - Hostnames
  - FQDNs



<http://www.flickr.com/photos/kathycsus/2686772625>

Monday, 30 June 14

Chef Search is one of its killer features. It enables you to do a lot of heavy lifting easily.

Search for nodes by their roles, or other attributes (you assign, or assume via cookbooks).

# Search for Nodes

```
pool_members = search("node", "role:webserver")

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy-app_lb.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  variables :pool_members => pool_members.uniq
  notifies :restart, "service[haproxy]"
end
```

# Search for Nodes

```
pool_members = search("node", "role:webserver")

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy-app_lb.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  variables :pool_members => pool_members.uniq
  notifies :restart, "service[haproxy]"
end
```

# Pass results into Templates

```
# Set up application listeners here.

listen application 0.0.0.0:80
  balance roundrobin
    <% @pool_members.each do |member| -%>
      server <%= member[:hostname] %> <%= member[:ipaddress] %>:>
weight 1 maxconn 1 check
  <% end -%>
<% if node["haproxy"]["enable_admin"] -%>
listen admin 0.0.0.0:22002
  mode http
  stats uri /
<% end -%>
```

Monday, 30 June 14

Note there is no space before the [ in attributes. Its  
node["haproxy"]["enable\_admin"]  
NOT this  
node ["haproxy"]["enable\_admin"]

# Pass results into Templates

```
# Set up application listeners here.

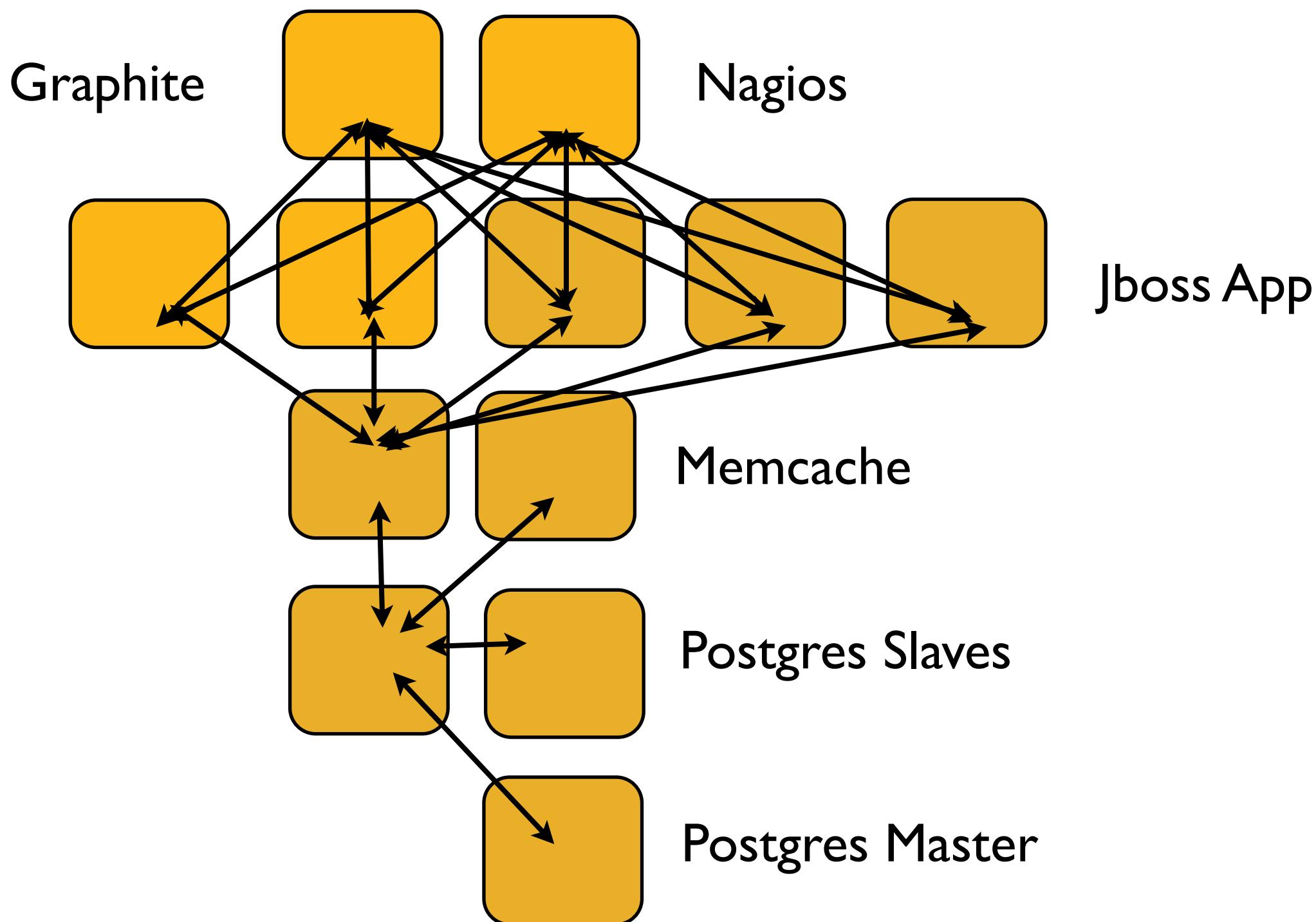
listen application 0.0.0.0:80
  balance roundrobin
    <% @pool_members.each do |member| -%>
      server <%= member[:hostname] %> <%= member[:ipaddress] %>:>
weight 1 maxconn 1 check
  <% end -%>
<% if node["haproxy"]["enable_admin"] -%>
listen admin 0.0.0.0:22002
  mode http
  stats uri /
<% end -%>
```

# Pass results into Templates

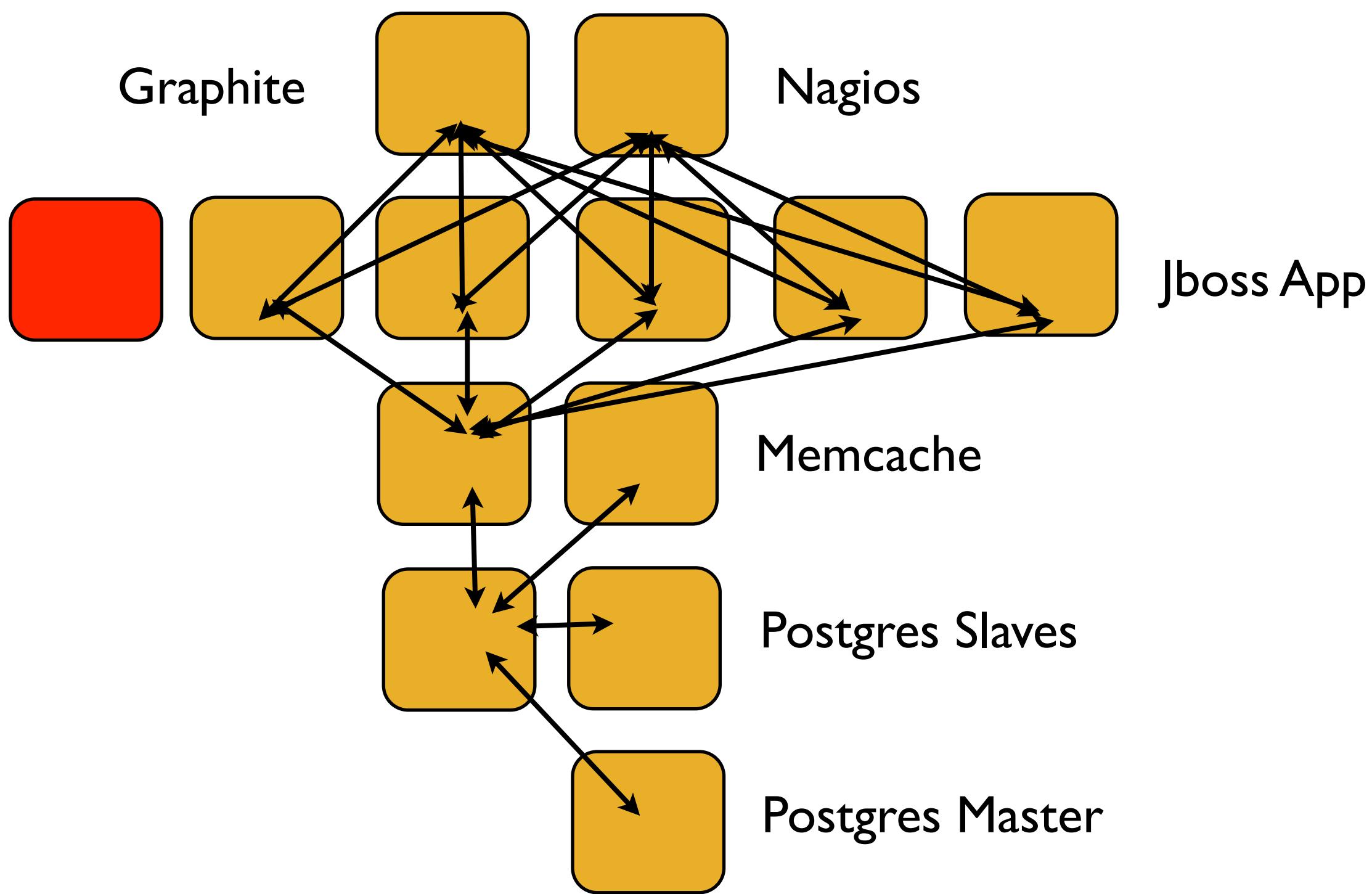
```
# Set up application listeners here.

listen application 0.0.0.0:80
  balance roundrobin
  <% @pool_members.each do |member| -%>
    server <%= member[:hostname] %> <%= member[:ipaddress] %>:>
  weight 1 maxconn 1 check
  <% end -%>
<% if node["haproxy"]["enable_admin"] -%>
listen admin 0.0.0.0:22002
  mode http
  stats uri /
<% end -%>
```

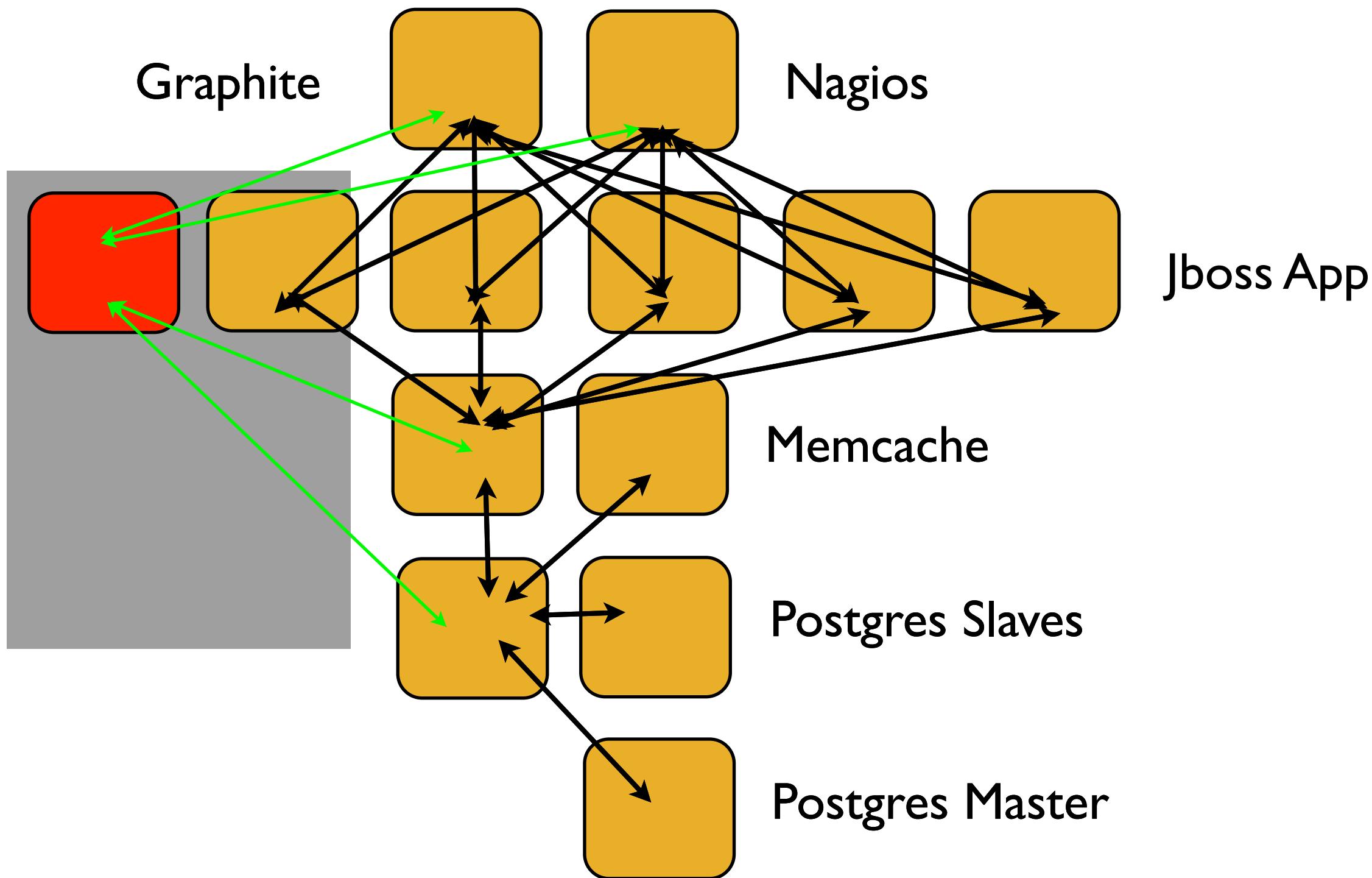
# So when this...



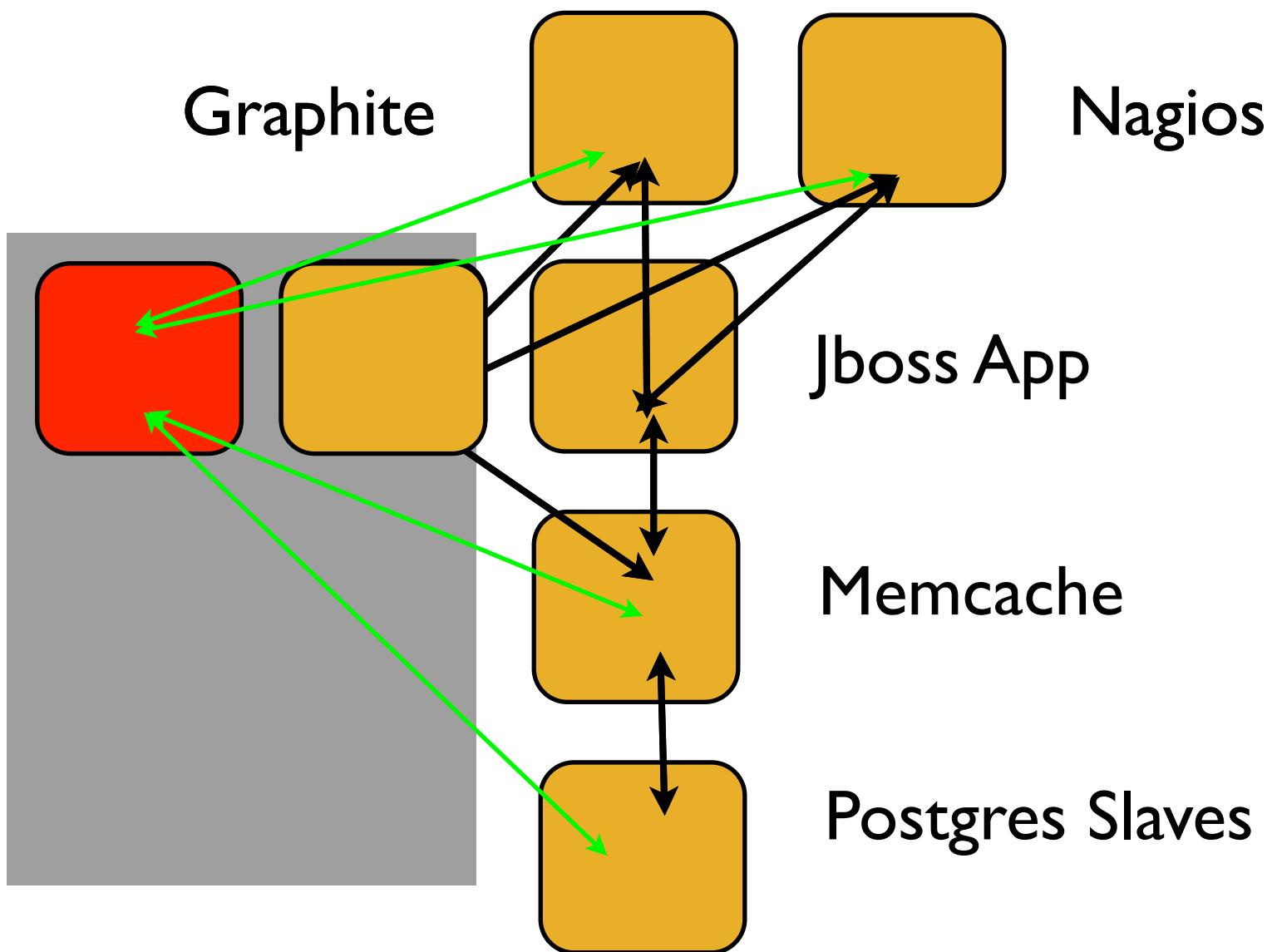
# ...becomes this



# ...this can happen automatically



# Count the Resources



- **12+ resource changes for 1 node addition**

- Load balancer config
- Nagios host ping
- Nagios host ssh
- Nagios host HTTP
- Nagios host app health
- Graphite CPU
- Graphite Memory
- Graphite Disk
- Graphite SNMP
- Memcache firewall
- Postgres firewall
- Postgres authZ config

Monday, 30 June 14

... this is the core of what Chef does.

Gives you tools to manage complexity while being flexible enough to evolve with your Infrastructure.

# Manage Complexity

- Determine the desired state of your infrastructure
- Identify the Resources required to meet that state
- Gather the Resources into Recipes
- Compose a Run List from Recipes and Roles
- Apply a Run List to each Node in your Environment
- Your infrastructure adheres to the policy modeled in Chef

# Configuration Drift

- Configuration Drift happens when:
  - Your infrastructure requirements change
  - The configuration of a server falls out of policy
- Chef makes it easy to manage
  - Model the new requirements in your Chef configuration files
  - Run the chef-client to enforce your policies

# Review Questions

- What is a Node?
- What is a Resource?
- What is a Recipe? How is it different from a Cookbook?
- What is a Run List?
- What is a Role?

Monday, 30 June 14

What you want coming out of this section is that folks have a high level understanding of the parts of the system. The rest of training goes into more detail about all these topics in due time.

Take 5.

# Workstation Setup

Getting started

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Login to Enterprise Chef
  - View your Organization in Enterprise Chef
  - Describe Knife, the Chef command line utility
  - Use Knife on your Workstation

# Legend

v1.2.3



Monday, 30 June 14

Quick detour to provide you some context for the slide format used throughout this course.

# 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 0xffffffff broadcast 10.100.0.255
            media: autoselect
            status: active
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
    ether 0a:cf:e9:1f:79:a3
    media: autoselect
    status: inactive
```

# Legend: Example of editing a file on your workstation



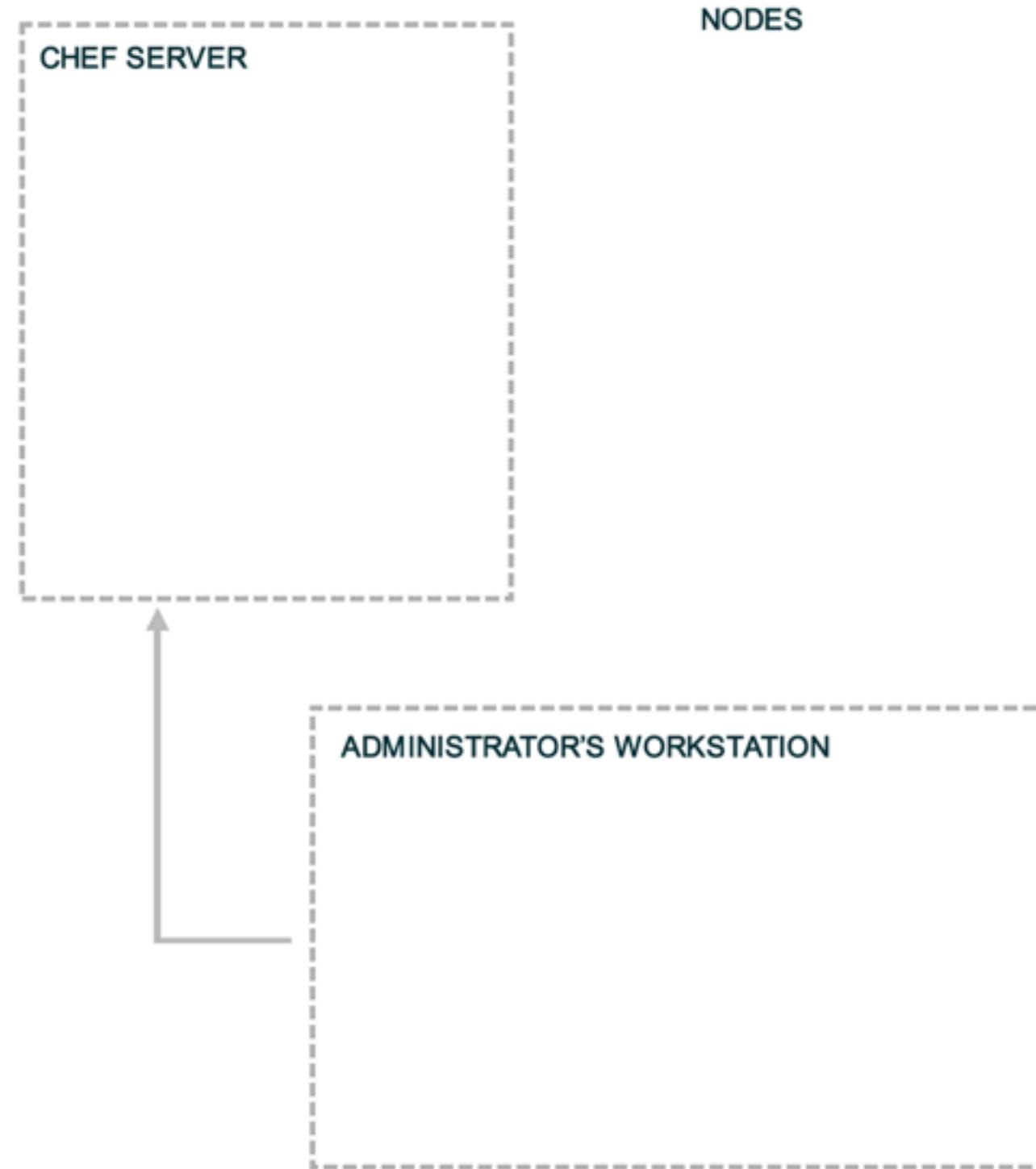
**OPEN IN EDITOR:** ~/hello\_world

Hi!

I am a friendly file.

**SAVE FILE!**

# Landscape of a Chef-managed Infrastructure



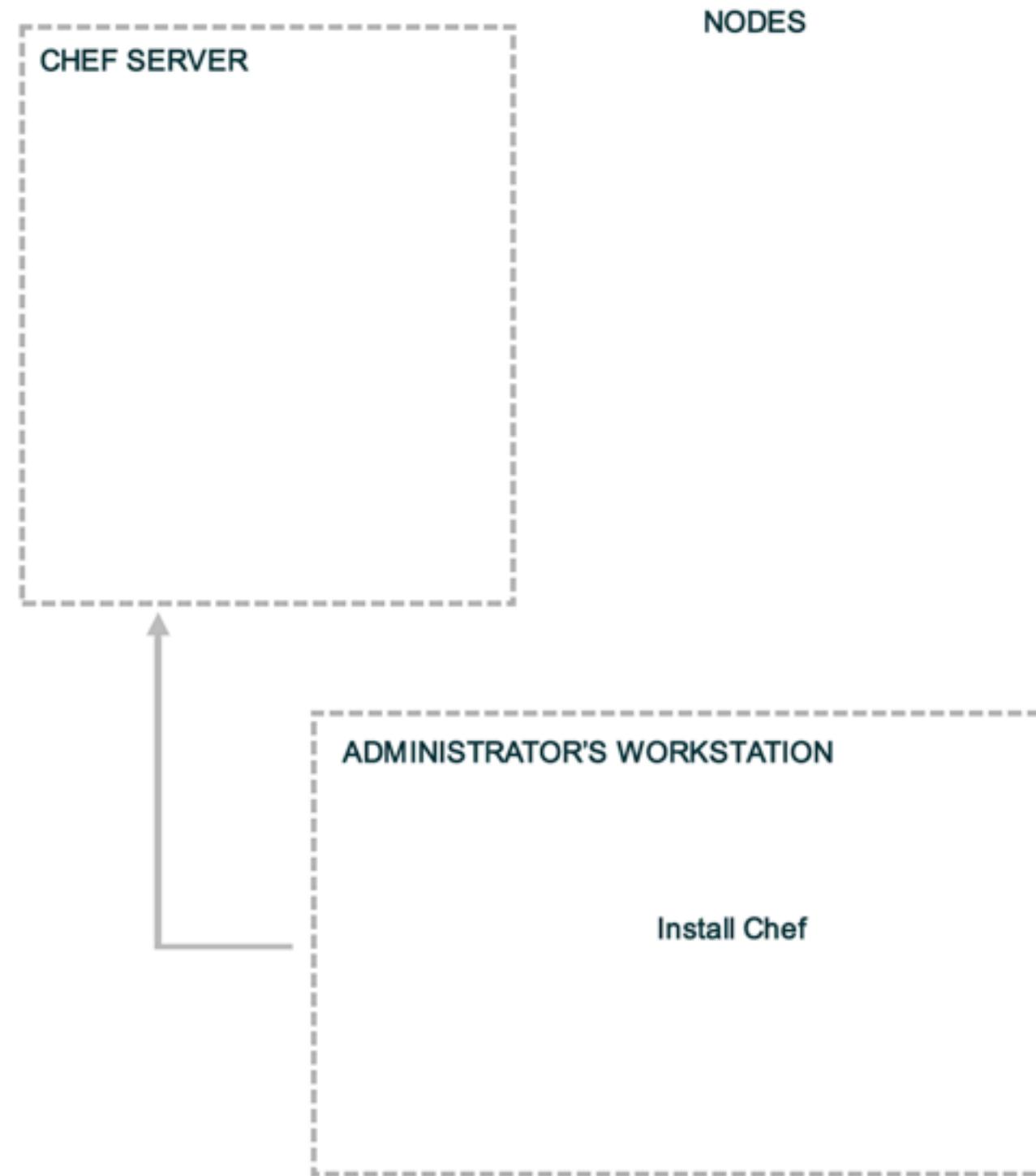
Monday, 30 June 14

This diagram will be used throughout to help students remember which of the three components we're talking about / working on.

Three main areas

1. Workstation
2. Enterprise Chef
3. Nodes in your infrastructure

# Landscape of a Chef-managed Infrastructure



Monday, 30 June 14

Start by installing Chef on the Workstation

# Install Chef

- Install Chef (if not already installed)
- <http://www.opscode.com/chef/install>

# Install Chef

## Download options

Chef Client

Chef Server

### Installing the Chef Client

Select the kind of system you would like to install the Chef Client on.

The versions listed have been tested and are supported.

[Select an Operating System] ▾

[Select a Version] ▾

[Select an Architecture] ▾

Monday, 30 June 14

Browse to the install page <http://www.opscode.com/chef/install> and show the various options.

Highlight:

- \* Selecting different operating systems, versions, and hardware
- \* curl and pipe to bash or download a package
- \* Windows install

# Install on Mac OSX

## Download options

Chef Client

Chef Server

### Installing the Chef Client

Select the kind of system you would like to install the Chef Client on. The versions listed have been tested and are supported.

OS X

10.7

x86\_64

### Quick Install Instructions

Open a shell on the target system and run the following command to download and install the latest version of the Chef client:

```
curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

### Downloads

You can install manually by downloading the package below after you have selected a Chef version. For more information about manual installation, [please read the documentation](#).

11.8.0-1

[chef-11.8.0-1.mac\\_os\\_x.10.7.2.sh](#)

# Install on Enterprise Linux

## Download options

Chef Client

Chef Server

### Installing the Chef Client

Select the kind of system you would like to install the Chef Client on. The versions listed have been tested and are supported.

Enterprise Linux



6



x86\_64



### Quick Installation Instructions

Open a root shell on the target system and run the following command to download and install the latest version of the Chef client:

```
curl -L https://www.opscode.com/chef/install.sh | bash
```

### Downloads

You can install manually by downloading the package below after you have selected a Chef version. For more information about manual installation, [please read the documentation](#).

11.8.0-1



[chef-11.8.0-1.el6.x86\\_64.rpm](#)

# Workstation Setup - Mac OS X / Linux

```
$ curl -L http://www.opscode.com/chef/install.sh | sudo bash
```

```
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
                                         Dload  Upload   Total   Spent   Left  Speed
100  6515  100  6515      0       0  20600      0  ---:---:---  ---:---:--- 31172
Downloading Chef for ubuntu...
Installing Chef
Selecting previously unselected package chef.
(Reading database ... 47446 files and directories currently installed.)
Unpacking chef (from .../tmp.MqRJP6lz/chef_amd64.deb) ...
Setting up chef (11.8.0-1.ubuntu.11.04) ...
Thank you for installing Chef!
Processing triggers for initramfs-tools ...
update-initramfs: Generating /boot/initrd.img-3.2.0-48-virtual
```

Monday, 30 June 14

Alternatively, Chef may be programmatically installed via the install.sh script. This is the method used during chef bootstraps.

This installs chef-client, knife, ohai and ruby along with any supporting ruby gems. This is exactly the same as gets installed on a node during bootstrap. Difference is the node does not use knife, while the workstation client does not use ohai – but each could be configured to perform either role!

# Workstation Setup - Windows

- Windows
- 2008 (Windows 7) or 2012 (Windows 8)
- i686 (32-bit) or x86\_64 (64-bit)
- 11.8.0

## Download options

Chef Client    Chef Server

### Installing the Chef Client

Select the kind of system you would like to install the Chef Client on. The versions listed have been tested and are supported.

Windows

2008r2

x86\_64

### Downloads

You can install manually by downloading the package below after about manual installation, [please read the documentation](#).

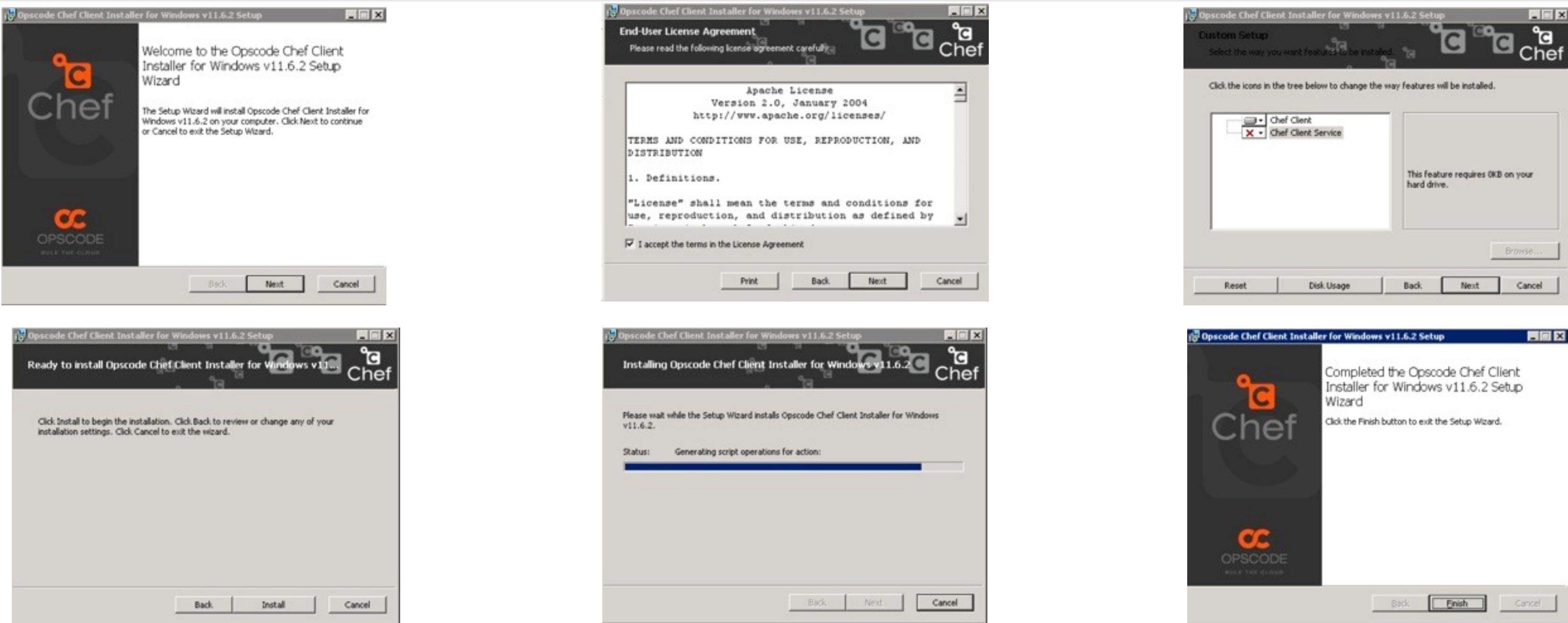
11.8.0-1

[chef-client-11.8.0-1.windows.msi](#)

**Download and install this file**



# Install on Windows



Monday, 30 June 14  
Screen shot of Windows installer

# 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

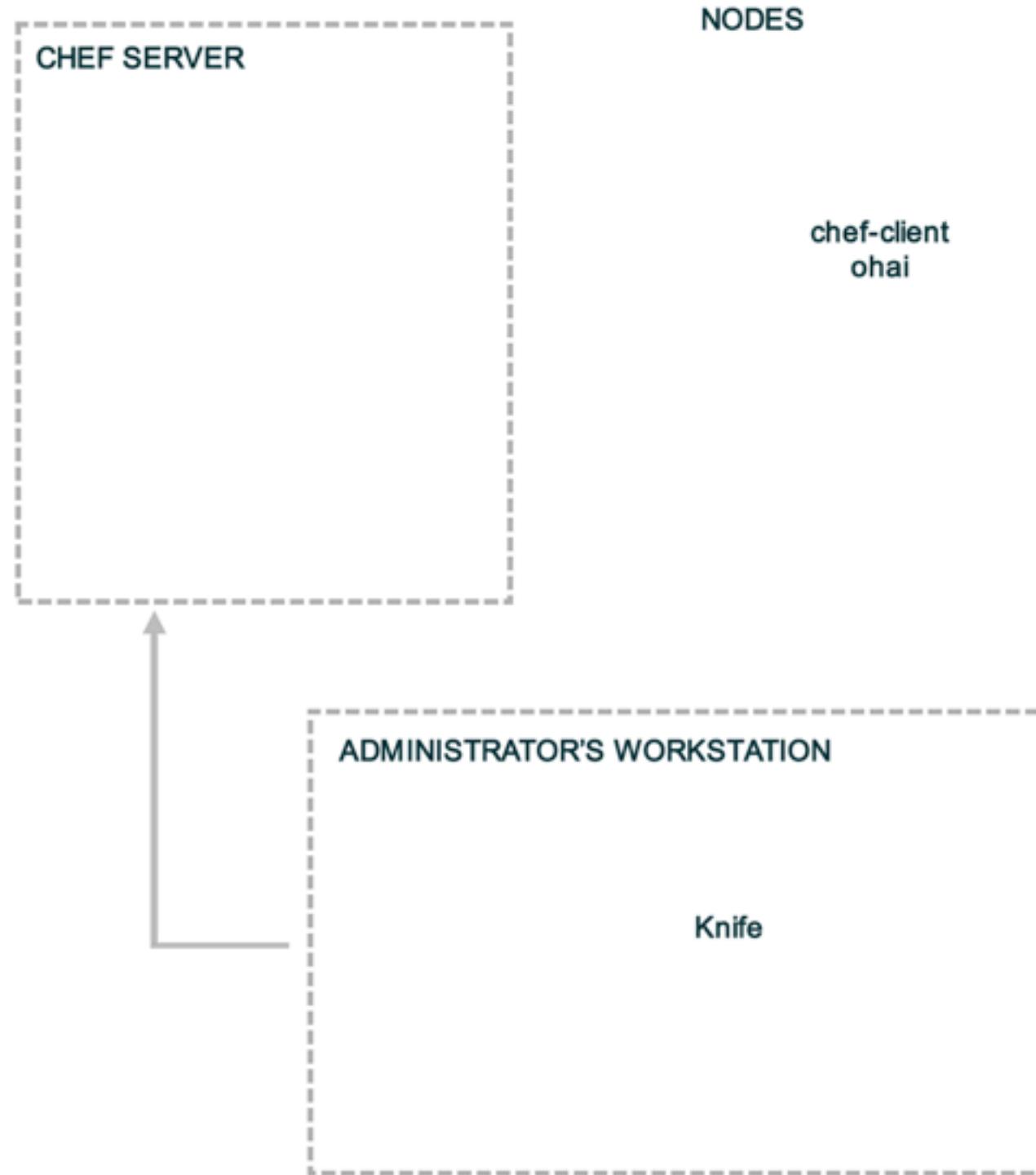
Monday, 30 June 14

A quick mention that we just used the omnibus installer which puts Chef and all of its dependencies on the workstation. The tools are installed under /opt/chef (linux/mac) or C:\Chef (Windows).

Also mention that 'gem install chef' is also a valid installation method if students are already using rvm, rbenv, or chruby to manage their Ruby installations or if they'd prefer to use the system ruby (which should be at version 1.9 or better to use this method).

Opscode manages an open source project called "Omnibus". Use omnibus to create OS-specific packages.

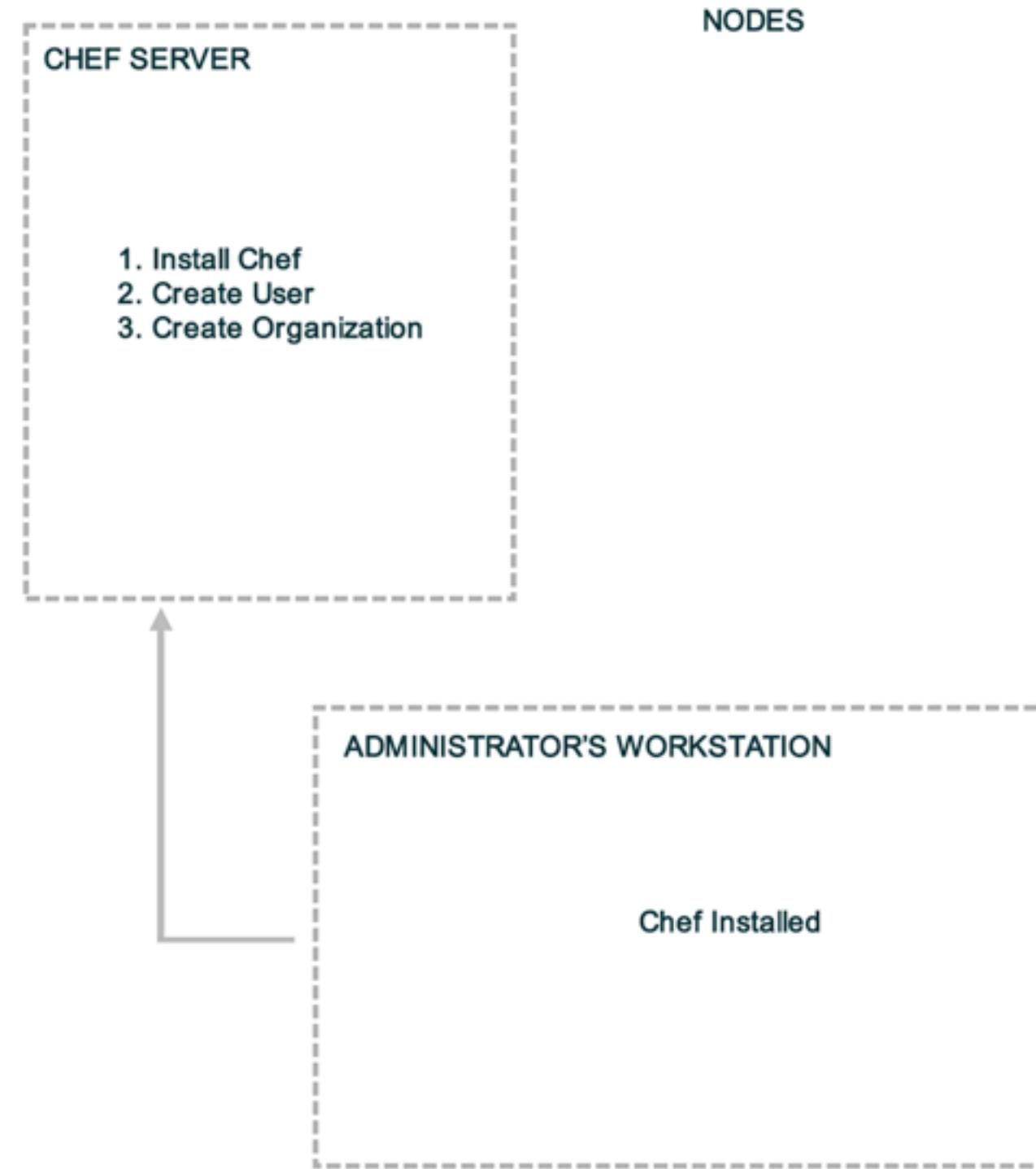
# Workstation or Node?



Monday, 30 June 14

We've just installed chef-client, knife, ohai and ruby along with any supporting ruby gems. This is exactly the same as gets installed on a node during bootstrap. Difference is the node does not use knife, while the workstation client does not use chef-client – but each could be configured to perform either role!

# Landscape of a Chef-managed Infrastructure



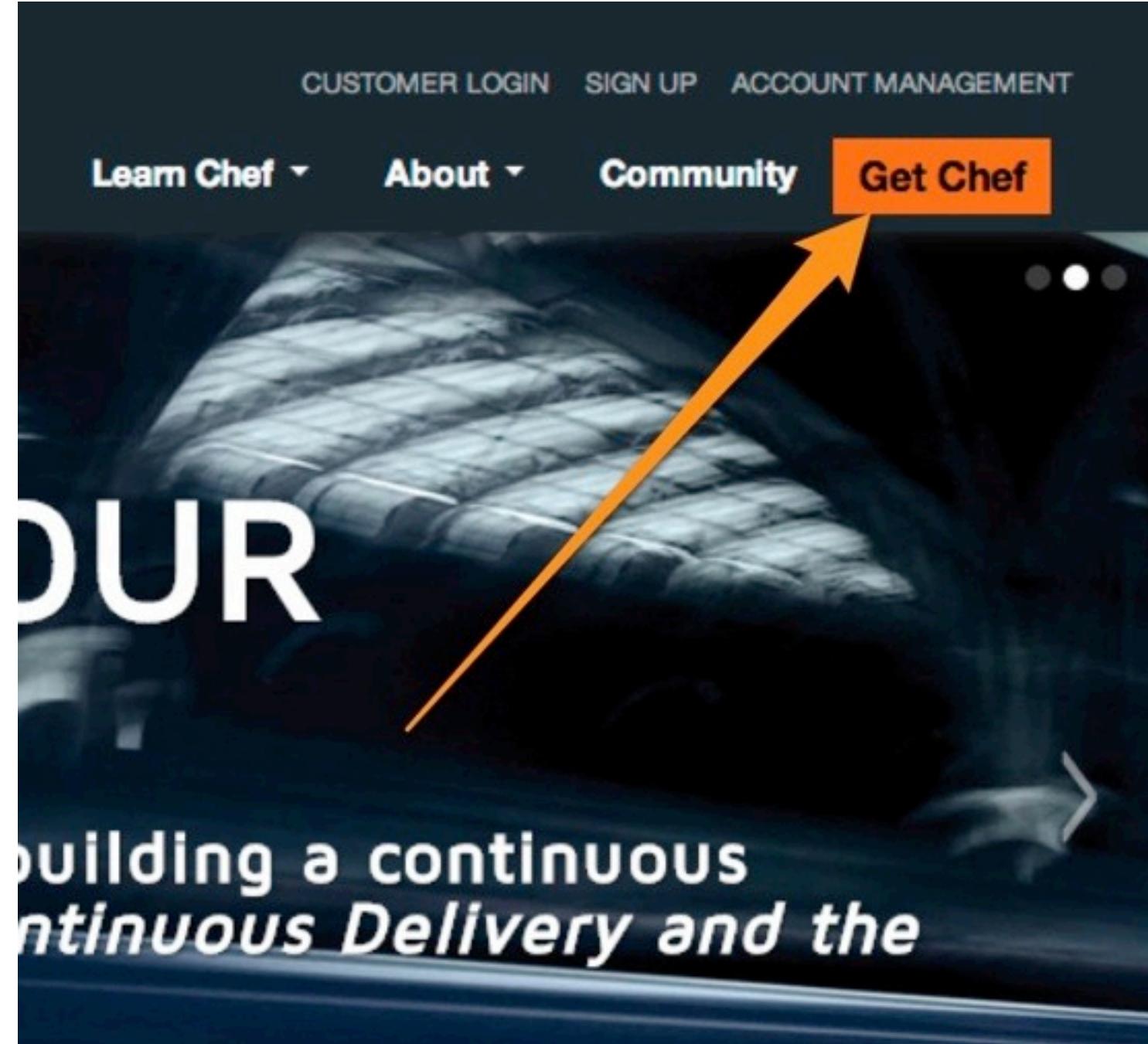
Monday, 30 June 14

With Chef installed on the workstation, it's now time to get the Chef Server setup.

We'll be using Hosted Enterprise Chef for this class.

# Your Chef Server for this class...

- Hosted Enterprise Chef
- <http://opscode.com>



# Create new account

- Sign up for a new account
- Chef Organization
  - provides multi-tenancy
  - name must be globally unique

## Start your free trial of Enterprise Chef

You're one step away from access to all the power and flexibility of Chef, hosted and supported by Opscode. Get ready to automate your infrastructure to accelerate your time to market, manage complexity, and safeguard your systems. Just complete the form to get started.

Full Name

Username

Email

Password

Company  (Optional)

Chef Organization

Organization is the name of your instance of Enterprise Chef.

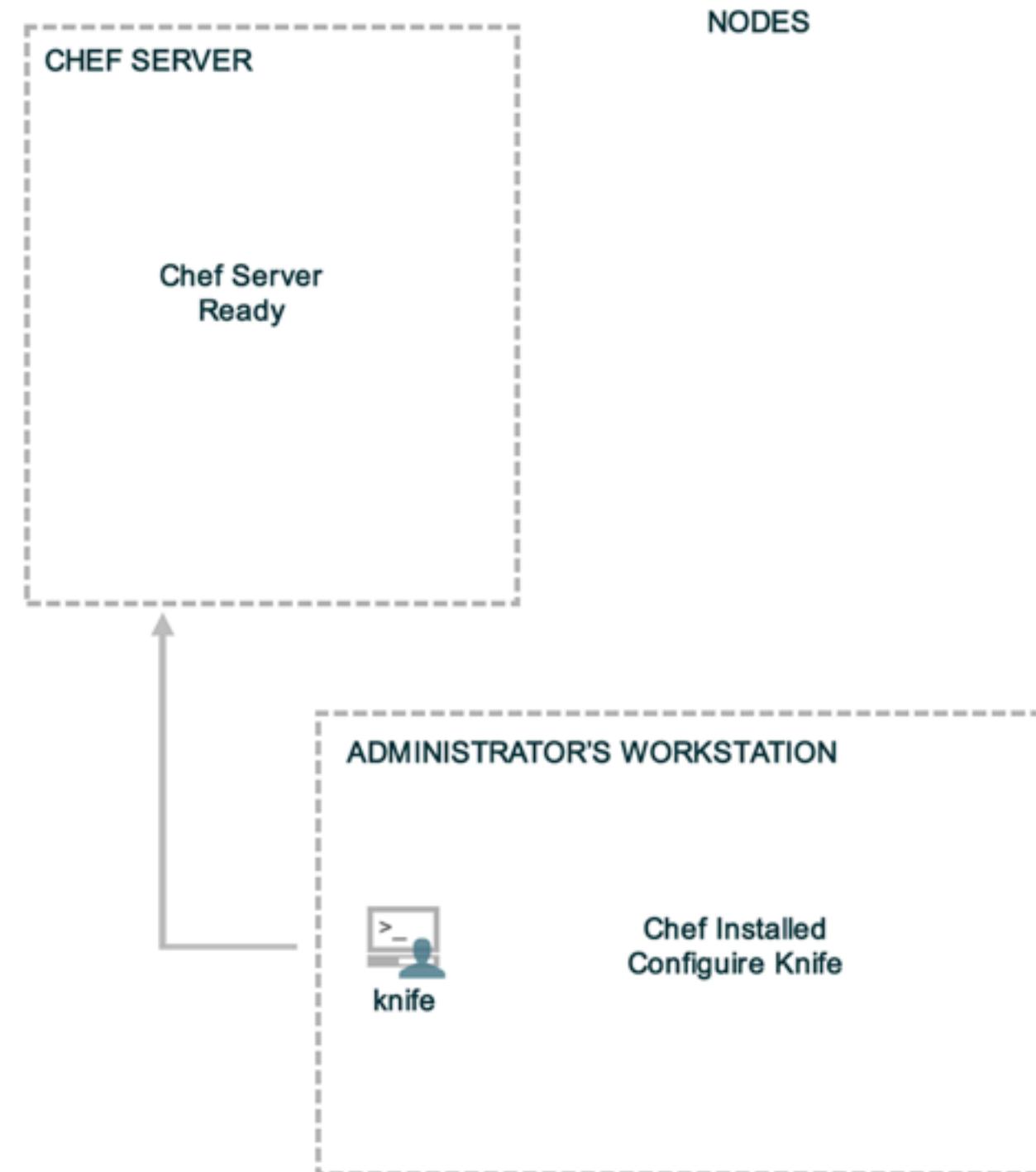
I agree to the [Terms of Service](#) and the [Master License and Services Agreement](#).

**Get Started**

Monday, 30 June 14

NOTE: Chef Organization must have a globally unique value.

# Landscape of a Chef-managed Infrastructure



Monday, 30 June 14

The next step is to configure knife to communicate with our Hosted Enterprise Chef organization. We'll do so by downloading the starter kit.

# Download "Starter Kit"

- You get a .zip file from clicking this
- Unzip the zipfile - you'll get a "chef-repo"
- Put the "chef-repo" somewhere, e.g.:
  - C:\Users\you\chef-repo (Win)
  - /Users/you/chef-repo (Mac)
  - /home/you/chef-repo (Linux)

Thank you for choosing Enterprise

Follow these three steps to be on your way to



Set up your

What's next?

[Chef Documentation](#)

The best place to start learning about Chef in general.

[Browse Community Cookbooks](#)

Hundreds of members of the Chef community have contributed cookbooks you can use or draw inspiration from.

Monday, 30 June 14

Download the starter kit from the Chef Server

Make a decision about where to store the work from this class.

Unzip the zip file in that directory.

A chef-repo directory, or folder, is now available.

# Knife is the command-line tool for Chef

- Knife provides an interface between a local Chef repository and the Chef Server
- Knife lets you manage:
  - Nodes
  - Cookbooks and recipes
  - Roles
  - Stores of JSON data (data bags), including encrypted data
  - Environments
  - Cloud resources, including provisioning
  - The installation of Chef on management workstations
  - Searching of indexed data on the Chef Server

Monday, 30 June 14

It is also where you will spend the bulk of your time when working with Chef. Knife is the primary user interface for working with Chef. As a bonus, it very closely mirrors how our actual APIs are structured – so working with knife also gets you familiar with our lower-level abstractions that you may wind up using as you grow.

\* This class teaches best practices – and all the pro Chefs use the command line interface almost exclusively.

# A quick tour of the chef-repo

- Every infrastructure managed with Chef has a Chef Repository ("chef-repo")
- Type all commands in this class from the chef-repo directory
- Let's see what's inside the chef-repo...

# Verify that knife is working

```
$ cd chef-repo
```

```
[ ~/chef-repo ] $
```

# A quick tour of the chef-repo

```
$ ls -al
```

```
total 40
drwxr-xr-x@ 11 opscode  opscode  374 Aug 15 09:42 .
drwxr-xr-x+ 92 opscode  opscode 3128 Aug 15 09:43 ..
drwxr-xr-x@  3 opscode  opscode 102  Aug 15 2013 .berkshelf
drwxr-xr-x@  5 opscode  opscode 170  Aug 15 2013 .chef
-rw-r--r--@  1 opscode  opscode 495  Aug 15 2013 .gitignore
-rw-r--r--@  1 opscode  opscode 1433 Aug 15 2013 Berksfile
-rw-r--r--@  1 opscode  opscode 2416 Aug 15 2013 README.md
-rw-r--r--@  1 opscode  opscode 3567 Aug 15 2013 Vagrantfile
-rw-r--r--@  1 opscode  opscode 588  Aug 15 2013 chefignore
drwxr-xr-x@  3 opscode  opscode 102  Aug 15 2013 cookbooks
drwxr-xr-x@  3 opscode  opscode 102  Aug 15 2013 roles
```

# What's inside the .chef directory?

```
$ ls .chef
```

```
ORGNAME-validator.pem  
USERNAME.pem  
knife.rb
```

# What's inside the .chef directory?

- `knife.rb` is the configuration file for Knife.
- The other two files are certificates for authentication with the Chef Server
  - We'll talk more about that later.

# knife.rb

- Default location
  - `~/.chef/knife.rb`
  - `%HOMEDRIVE% : %HOME PATH%\.chef` (Windows)
- Use a project specific configuration
  - `.chef/knife.rb` of the current directory
  - `chef-repo/.chef/knife.rb`
- [http://docs.opscode.com/config\\_rb\\_knife.html](http://docs.opscode.com/config_rb_knife.html)

Monday, 30 June 14

knife will look for it's configuration (knife.rb) in:

1. current working directory's `.chef/`
2. current user's home directory's `.chef/`
3. `/etc/chef/knife.rb`

Community plugins to help manage knife.rb files:

- \* knife-block – <https://github.com/greenandsecure/knife-block>
- \* ChefVM – <https://github.com/trobrock/chefvm>

knife.rb should be built in such a way that it can be checked into version control and shared by a team. The docs site as one example of such a knife.rb [http://docs.opscode.com/config\\_rb\\_knife.html#many-users-same-repo](http://docs.opscode.com/config_rb_knife.html#many-users-same-repo)

[http://docs.opscode.com/config\\_rb\\_knife.html](http://docs.opscode.com/config_rb_knife.html) for additional details

# knife.rb



**OPEN IN EDITOR:** [chef-repo/.chef/knife.rb](#)

```
current_dir = File.dirname(__FILE__)
log_level           :info
log_location        STDOUT
node_name           "USERNAME"
client_key          "#{$current_dir}/USERNAME.pem"
validation_client_name "ORGNAME-validator"
validation_key       "#{$current_dir}/ORGNAME-validator.pem"
chef_server_url     "https://api.opscode.com/organizations/ORGNAME"
cache_type           'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path        ["#{$current_dir}/../cookbooks"]
```

Monday, 30 June 14

The other files in the .chef directory:

- \* client\_key
- \* validation\_key

The reason your Organization name needs to be globally unique:

- \* chef\_server\_url

Want to dig deeper into knife.rb? Check these (possibly outdated) links:

Fully commented knife.rb – <https://gist.github.com/jtimberman/1718805>

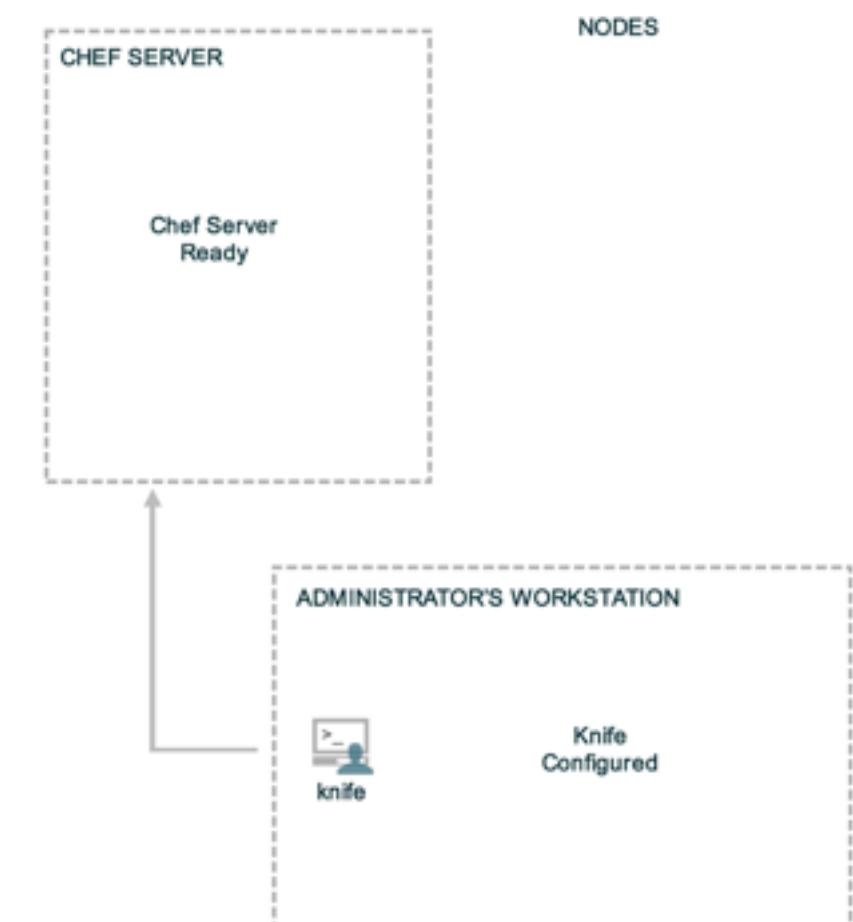
A knife.rb for Our Time – <http://dougireton.com/blog/2013/03/17/a-knife-dot-rb-for-our-time/>

# Verify Knife

```
$ knife --version  
Chef: 11.8.0
```

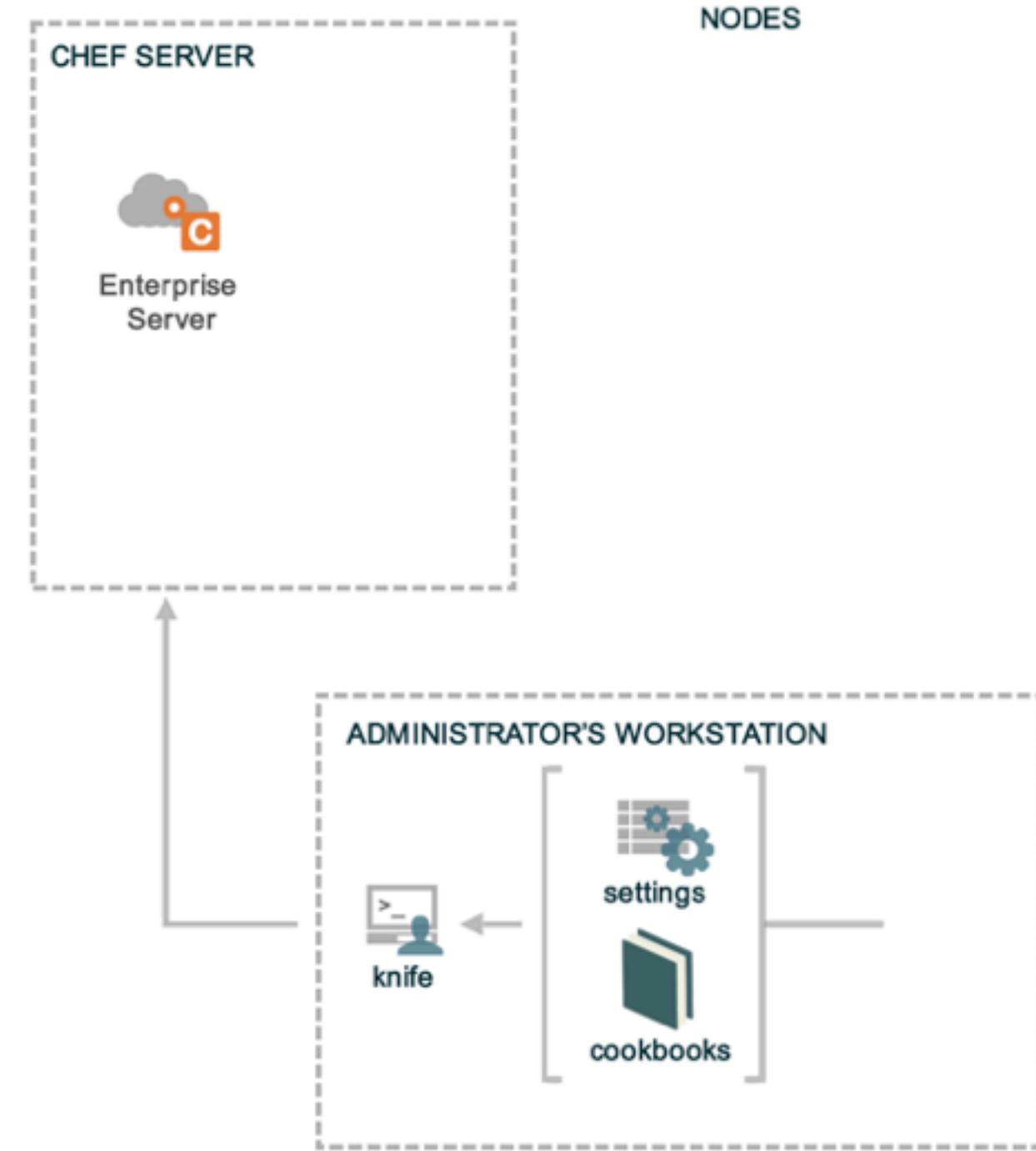
```
$ knife client list  
ORGNAME-validator
```

- Your version may be different, that's ok!



# knife client list

- Read the `chef_server_url` from `knife.rb`
- HTTP GET to `#{{chef_server_url}}/clients`
- Display the result



Monday, 30 June 14

What happens when you execute 'knife client list'?

To find out, run 'knife client list -VV'

```
$ knife client list -VV
DEBUG: Signing the request as nharveywss01
DEBUG: Sending HTTP Request via GET to api.opscode.com:443/organizations/nhcompany/clients
DEBUG: ---- HTTP Status and Header Data: ----
DEBUG: HTTP 1.1 200 OK
DEBUG: server: nginx
DEBUG: date: Tue, 22 Oct 2013 17:02:19 GMT
DEBUG: content-type: application/json
DEBUG: content-length: 189
DEBUG: connection: close
DEBUG: x-ops-api-info: flavor=hec;version=11.0.0;oc_erchef=0.21.31
DEBUG: ---- End HTTP Status/Header Data ----
nhcompany-validator
```

# Exercise: Run ‘knife help list’

```
$ knife help list
```

Available help topics are:

- bootstrap
- chef-shell
- client
- configure
- cookbook
- cookbook-site
- data-bag
- environment
- exec
- index
- knife
- node
- role
- search
- shef

Monday, 30 June 14

This is an optional slide!

Careful – this doesn’t work on Windows XP

Also, careful, help pages don’t work very well on Windows.

# Exercise: Run ‘knife client list’

```
$ knife client list
```

**ORGNAME-validator**

Monday, 30 June 14

This is an optional slide!

Make sure every student can successfully run ‘knife client list’, which means they are fully set up.

## Best Practice: Use a text editor with a project drawer, or equivalent

- Chef is about Infrastructure as Code
- People who code for a living use text editors that are designed for the task
- Vim, Emacs, Sublime Text, Notepad++, etc.

Monday, 30 June 14

This is an optional slide!

Have a project drawer! Chef organizes code into folders, and using the command line to constantly open/close individual files is maddening

Here is where we see the diff in Ops vs. Devs :-)

Text editors: because that's how you roll now. Either you are a glutton for punishment or you will realize these tools exist to make your life easier and you will use them.

# Set Knife's Editor: Windows



**OPEN IN EDITOR:** ~/chef-repo/.chef/knife.rb

```
# Choose one of the following.  
knife[:editor] = "C:\\path\\to\\editor.exe --flags"
```

## Notepad++

```
"C:\\progra~2\\notepa~1\\notepad++.exe -nosession -multiInst"
```

## Sublime Text

```
"C:\\progra~1\\sublim~1\\sublime_text.exe --wait"
```

## Textpad

```
"C:\\progra~2\\textpa~1\\TextPad.exe"
```

## Vim

```
"C:\\progra~2\\vim\\vim74\\gvim.exe"
```

**SAVE FILE!**

Monday, 30 June 14

This is an optional slide!

Most editors only ship as 32-bit versions hence Progra~2 (which maps to "Program Files (x86)")  
The ones that don't go in Progra~1 (which maps to "Program Files")

# Set Knife's Editor: Linux/Mac



**OPEN IN EDITOR:** `~/chef-repo/.chef/knife.rb`

```
# Choose one of the following.  
knife[:editor] = "/path/to/editor --flags"
```

## Vim

`"/usr/bin/vim"`

## Emacs

`"/usr/bin/emacs"`

## Nano

`"/usr/bin/nano"`

## Sublime Text

`"/Applications/Sublime\ Text\ 2.app/Contents/SharedSupport/bin/subl -w"`

## Text Wrangler

`"/Applications/TextWrangler.app/Contents/Helpers/edit -w"`  
**SAVE FILE!**

Monday, 30 June 14

This is an optional slide!

Of course, students can still set their EDITOR in environment variables and Knife will use that.

# Using your favorite EDITOR

## Linux/OSX/Unix



**OPEN IN EDITOR:** `~/.bash_profile`

```
export EDITOR=vim
```

**SAVE FILE!**

```
$ source ~/.bash_profile
```

## Windows in cmd.exe

```
$ setx EDITOR "\"%ProgramFiles%\Windows NT\Accessories\wordpad.exe\""
```

or

```
$ setx EDITOR "\"%ProgramFiles%\Notepad++\notepad++.exe\" -nosession -multiInst"
```

close cmd.exe and start a new cmd.exe session

Monday, 30 June 14

This is an optional slide!

Common editors:

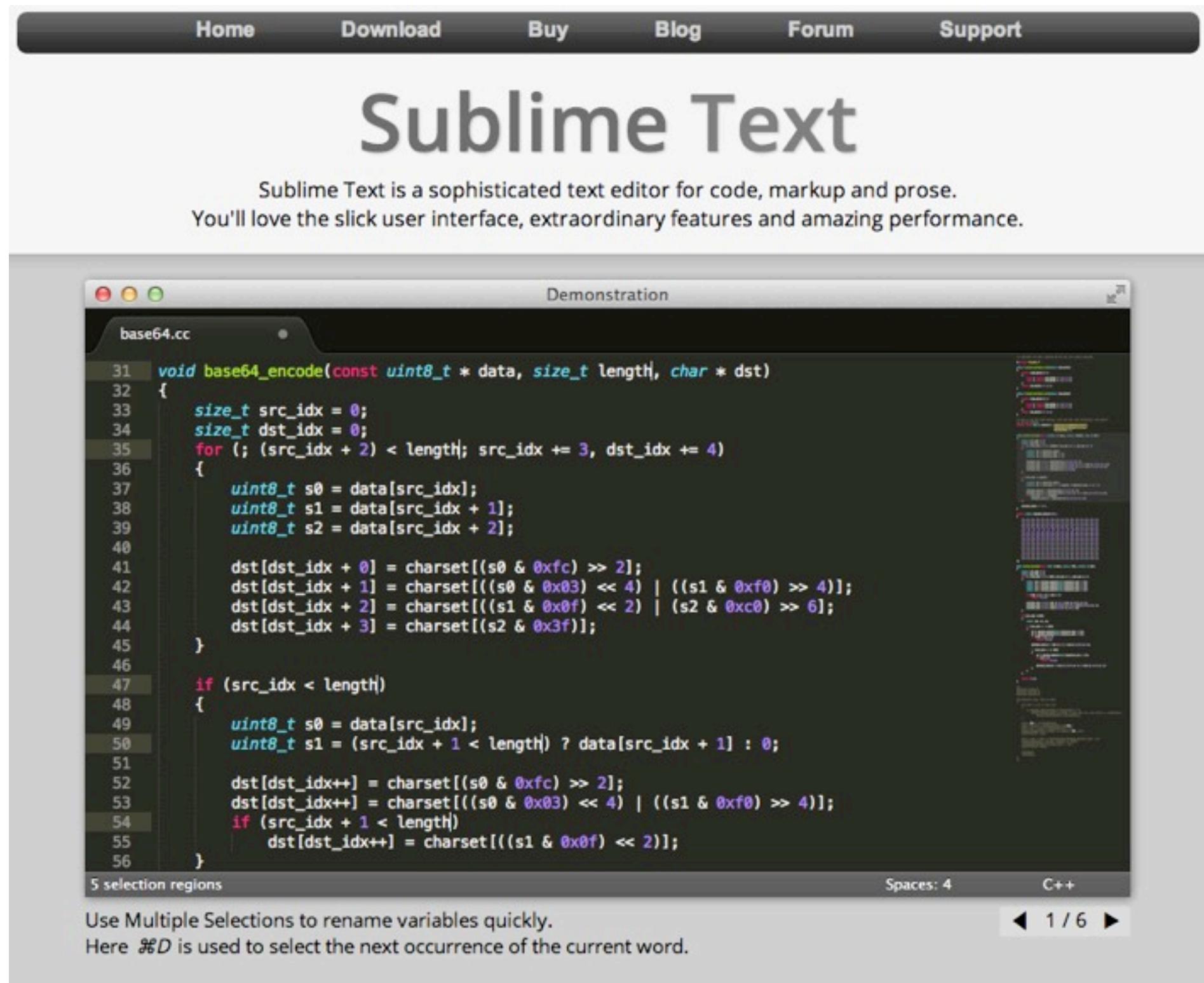
vim  
emacs  
nano  
textmate  
sublime [http://www.sublimetext.com/docs/2/osx\\_command\\_line.html](http://www.sublimetext.com/docs/2/osx_command_line.html)

On windows, the best is wordpad, and you can find it at %ProgramFiles%\Windows NT\Accessories\WORDPAD.EXE  
Windows folks using Notepad++: set wait via "multiInst" flag "%ProgramFiles%\blahblah\notepad++.exe -multiInst"

In Windows, quit cmd.exe and start a new cmd.exe to get the EDITOR set. Verify with the 'env' command's output.

If you do not have a preferred text editor on your workstation already...

- Download Sublime Text
- Free trial, not time bound
- Works on every platform
- [sublimetext.com](http://sublimetext.com)



Monday, 30 June 14

This is an optional slide!

# Using your favorite EDITOR

## Linux/OSX/Unix



**OPEN IN EDITOR:** `~/.bash_profile`

```
export EDITOR='/Applications/Sublime\ Text\ 2.app/Contents/SharedSupport/bin/subl -w'
```

**SAVE FILE!**

```
$ source ~/.bash_profile
```

## Windows in cmd.exe

```
$ setx EDITOR "\"%ProgramFiles%\Sublime Text 2\sublime_text.exe\" --wait"
```

close cmd.exe and start a new cmd.exe session

Monday, 30 June 14

This is an optional slide!

Common editors:

vim  
emacs  
nano  
textmate  
sublime

On windows, the best is wordpad, and you can find it at %ProgramFiles%\windows NT\Accessories\WORDPAD.EXE  
Windows folks using Notepad++: set wait via "multiInst" flag "%ProgramFiles%\blahblah\notepad++.exe -multiInst"

In Windows, quit cmd.exe and start a new cmd.exe to get the EDITOR set. Verify with the 'env' command's output.

# Exercise: Verify an EDITOR is configured

```
$ knife client create editor-test
```

```
{  
  "name": "editor-test",  
  "public_key": null,  
  "admin": false,  
  "json_class": "Chef::ApiClient",  
  "chef_type": "client"  
}
```

**Save the file and Exit your editor.**

Monday, 30 June 14

This is an optional slide!

Common editors:

vim  
emacs  
nano  
textmate  
sublime

On windows, the best is wordpad, and you can find it at %ProgramFiles%\windows NT\Accessories\WORDPAD.EXE  
Windows folks using Notepad++: set wait via "multiInst" flag "%ProgramFiles%\blahblah\notepad++.exe -multiInst"

# Exercise: Verify an EDITOR is configured

```
Created client[editor-test]
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAv313jX0crYxdNsqXTflh2N6e3kGSUTCJRVwCXIFh6YmdoY3c
bcIWESebcL5y5Vta3HjtSmohkSIW3JF0BLK39fQ8V0dM0wSBmZm9NvJwKucuwRes
u8EszMeLY/9tGFD9TK0iFlPN1jtRE1Ym61QRziRyAvkbkPEZacIVdayNM5TS8LuN
BSdYIZM8vnqRr6cf7WJn0m+1IKdcR1plHwtQT9EgYyPBVlwW9R8fR1i/QHeOyGz2
RT1YYAq+qvUimv3OQozXca/nkxroFWj80s9RJCgPdI44n51ws/PPmPQYeWykZft
9+pg5dud+Q1WbxR5yH01FDGVGaoFBB6mny+JaQIDAQABoIBAQCeKdC8gfFU+1Hm
trEAi5IhIcfQxhJHkzJeop+kro0j6zHSxayiz6OQacl+9x9JiAplfjrvBPhSuvht
GIrecot10FSitlajGN6+8vgRUXAKT8cTYC2hKu6I+eyNHOWMJyEA2yQDkxa582aE
9SzSRS8ruHifMght0GZwLwmwl2RNjBpbqWK6YI24yv61jgqkcUoC7gGECMKWFWoE
4DLYKnmza+0WeVe+xeC9R9gCz4qqq56U06RptcJ1jlQNzjLshv/1x1tRgyiy1k7Q
INmMPYcproh2Frmx5sipBml2nxGUNfJncMMeFV9AGddIdvPYzY2Wp5vZukTUVTxB
p003T2sBAoGBAOFRX6cEAvh4P8qZ3UqzDbBuuREJ6XGWNOdMjk4CgJn7owGXBig
XIiB4du301Cfd2G7p2FFipzaxylEVHTd9djT5NA6FBvUL6ATo0cs61teTwixwbtH
1RhM2im8DYx+RR2PjpsaerQw3DBVx19EUaU7LS5W7CSMuHuEjhpru1QhAoGBANmQ
2c3NGshBy06d0fKSXIU5h2nyM7N33tsj08upgerA1QXXcRLjWJ2Xr12W+npeFtTV
LtSsBV9HirayhfVqiSiLj1ulS+GbFHMZy8i9vuO09JM+ysI0NY1XPhxcNjpqRZLn
/ppAbBaZCunx9S7zLRN/YQZ0dsScRoYkH//UcwxAoGBAMiuVHaKESdC2vZVco6s
yz2CEJ1Hab98XGRCgGCkw+vh/z5UR+yFlj8TB5pNMhT0zxmCd+OM7Ye+dIchOio
JQDZQWvgvrZG16CIvR15kHi01/ATeoyWnwqEsK8JSjv+3wpEKWyl+fHxrrvyOp/0
Vo/HFCe4zZyEJXAGD6SdvXxhAoGBAKfUWF+sV3uhXW7QVFIt21cd8LqmjoFj97K+
KXRS1xg7FljWTi60194Bk9KzU5cv5ckuFJwPFiqfHPAtRuCyj1Ppw/ALA/lrFm5
zXyV+nnZ3gr0bj7XPXRBl3UTIPXg4riXY8yj431vi38iGcvU5LHEshjGUFIIMZ060
8UZNIU6ZAoGAGXIrvnC/vCsmt8wEP13F5xfp9P3w+7gZEx18lco1O4bsxtTIF3+BP
M/OQbtBm294EBzyyRZzTEXhaAByGI07vXeRuv9H8cpFnSlrMaAXB+WV+drh0ll3J
AWLIT8ILi3Vhbmg+1Kx7CsK8JinBey3dPvfJ8L9l+gbsEU+3Cv8+U8M=
-----END RSA PRIVATE KEY-----
```

Monday, 30 June 14

This is an optional slide!

Make sure this output appears only after your editor pops up, and you save it

# Exercise: Clean up after EDITOR test

```
$ knife client delete editor-test
```

```
Do you really want to delete editor-test? (Y/N) Y  
Deleted client[editor-test]
```

Monday, 30 June 14

This is an optional slide!

Clean up after yourself

# A few knife tips

- Commands are always structured as follows
  - knife
  - NOUN (client)
  - VERB (list)
- You can get more help with
  - knife NOUN help
  - knife --help just shows options

Monday, 30 June 14

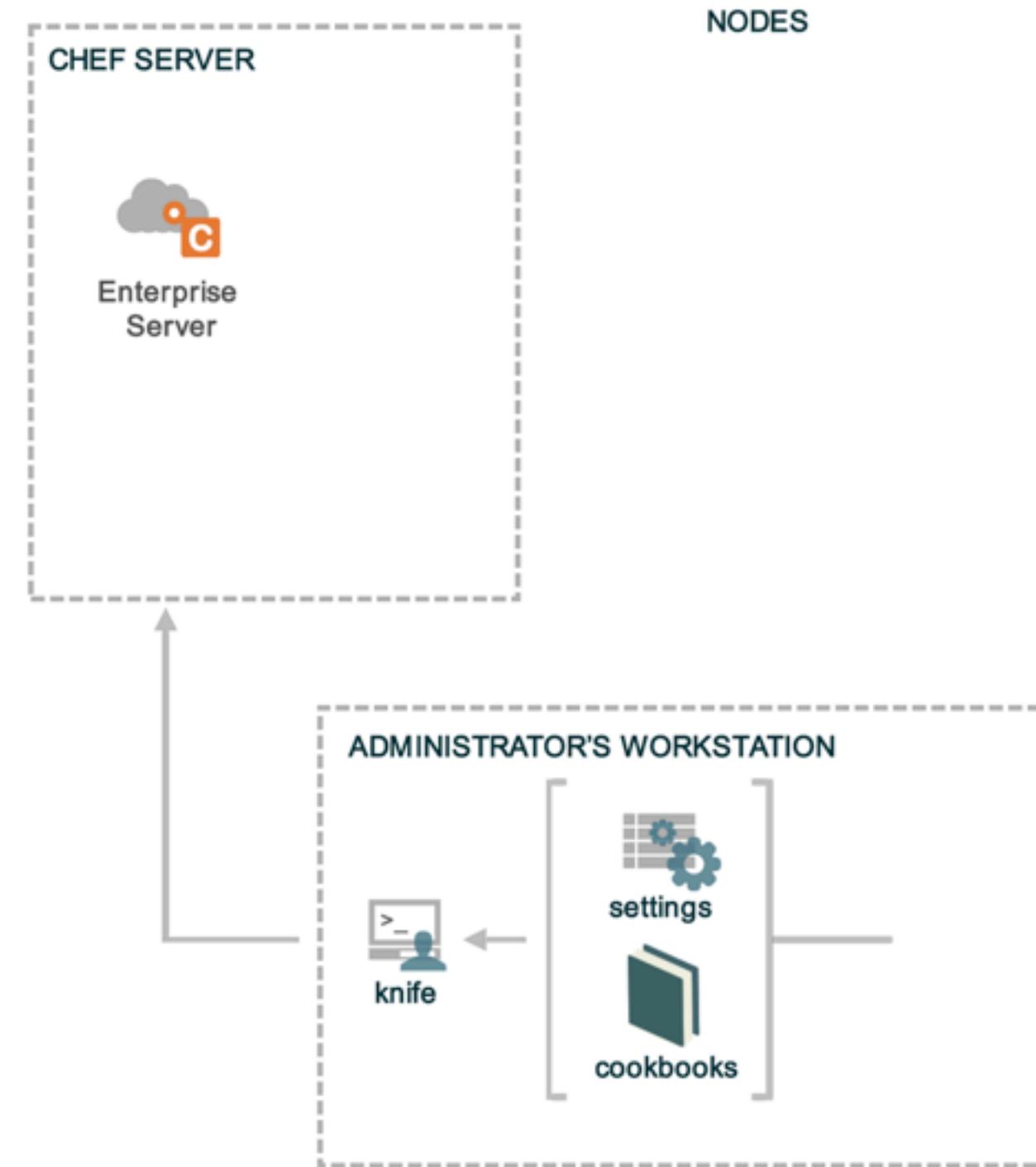
This is an optional slide!

Should probably be removed since "knife NOUN VERB" is common but certainly not consistent or "always"

There are exceptions, but this is true for all the core primitives.  
(this is true until it isn't)

(exceptions are things like ssh, search, etc)

# Checkpoint

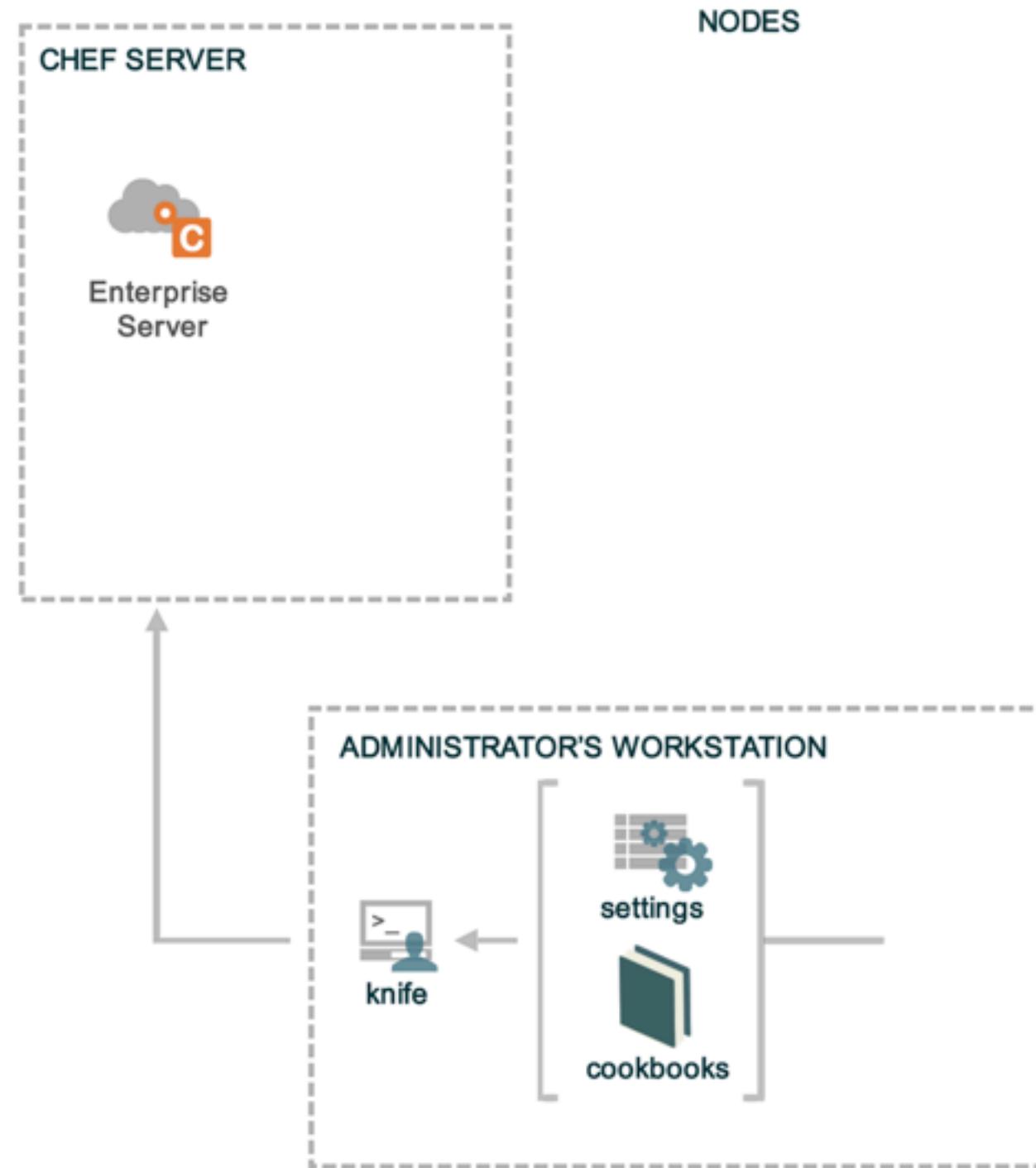


Monday, 30 June 14

Chef Server is up and running

Knife is installed and configured

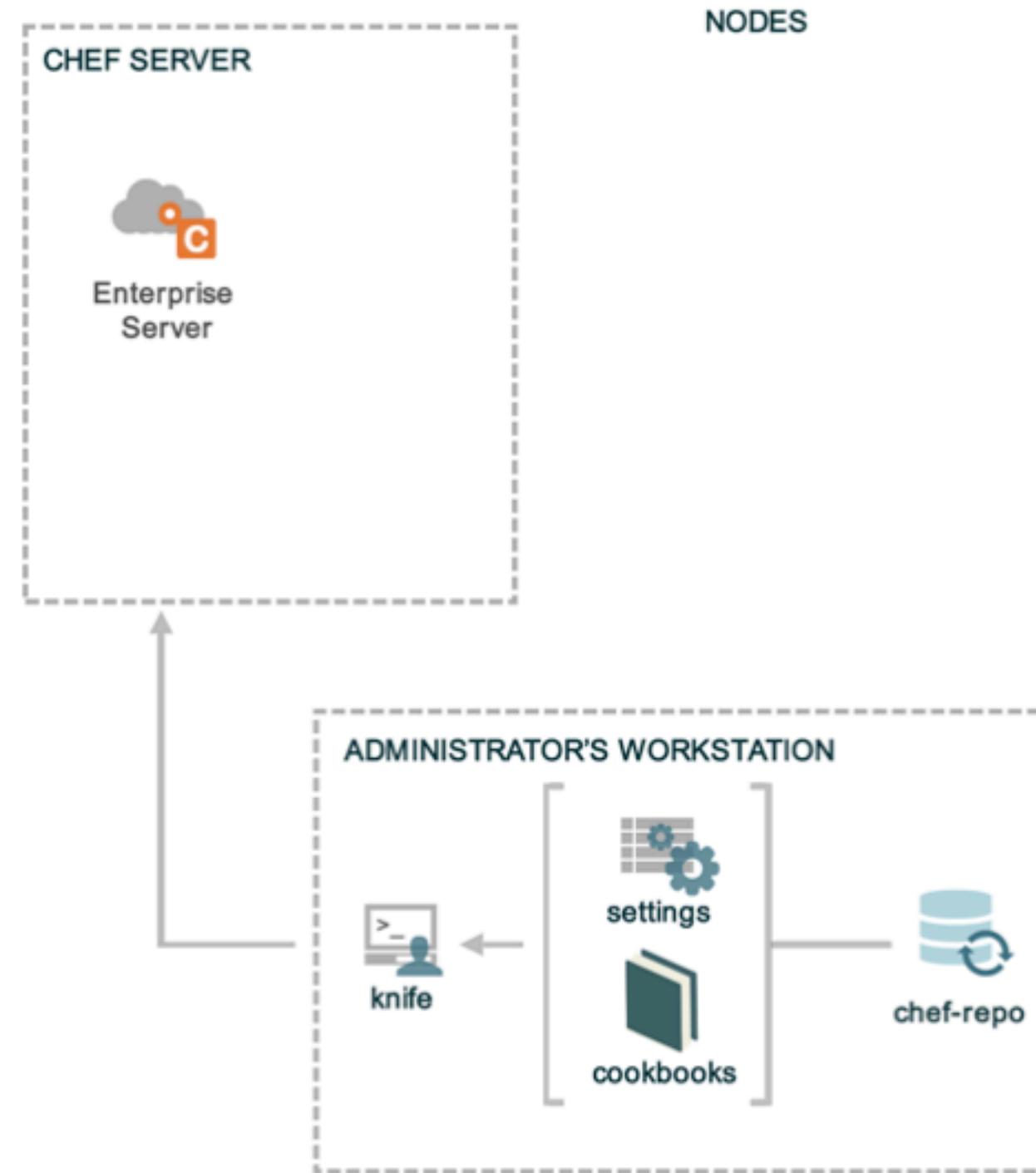
# What's Next?



Monday, 30 June 14

What is the next step towards managing our infrastructure as code?

# Source Code Repository



Monday, 30 June 14

We should now create a source code repository for our code

# Initialize a git repository

```
$ git init
```

```
Initialized empty Git repository in /  
Users/opscode/chef-repo/.git/
```

Monday, 30 June 14

This is an optional slide!

If git is installed (it should be), initialize a git repository

# Check git status

```
$ git status
```

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .berkshelf/
#       .chef/
#       .gitignore
#       Berksfile
#       README.md
#       Vagrantfile
#       chefignore
#       cookbooks/
#       roles/
nothing added to commit but untracked files present (use "git add" to track)
```

Monday, 30 June 14

This is an optional slide!

Check the status of the current git repository

# Add all files to git

```
$ git add .
```

```
[ ~/chef-repo ] $
```

Monday, 30 June 14

This is an optional slide!

Add all the files to the git repository.

There is no output

# Add files to git

```
$ git commit -m "add the starter kit from Chef"
```

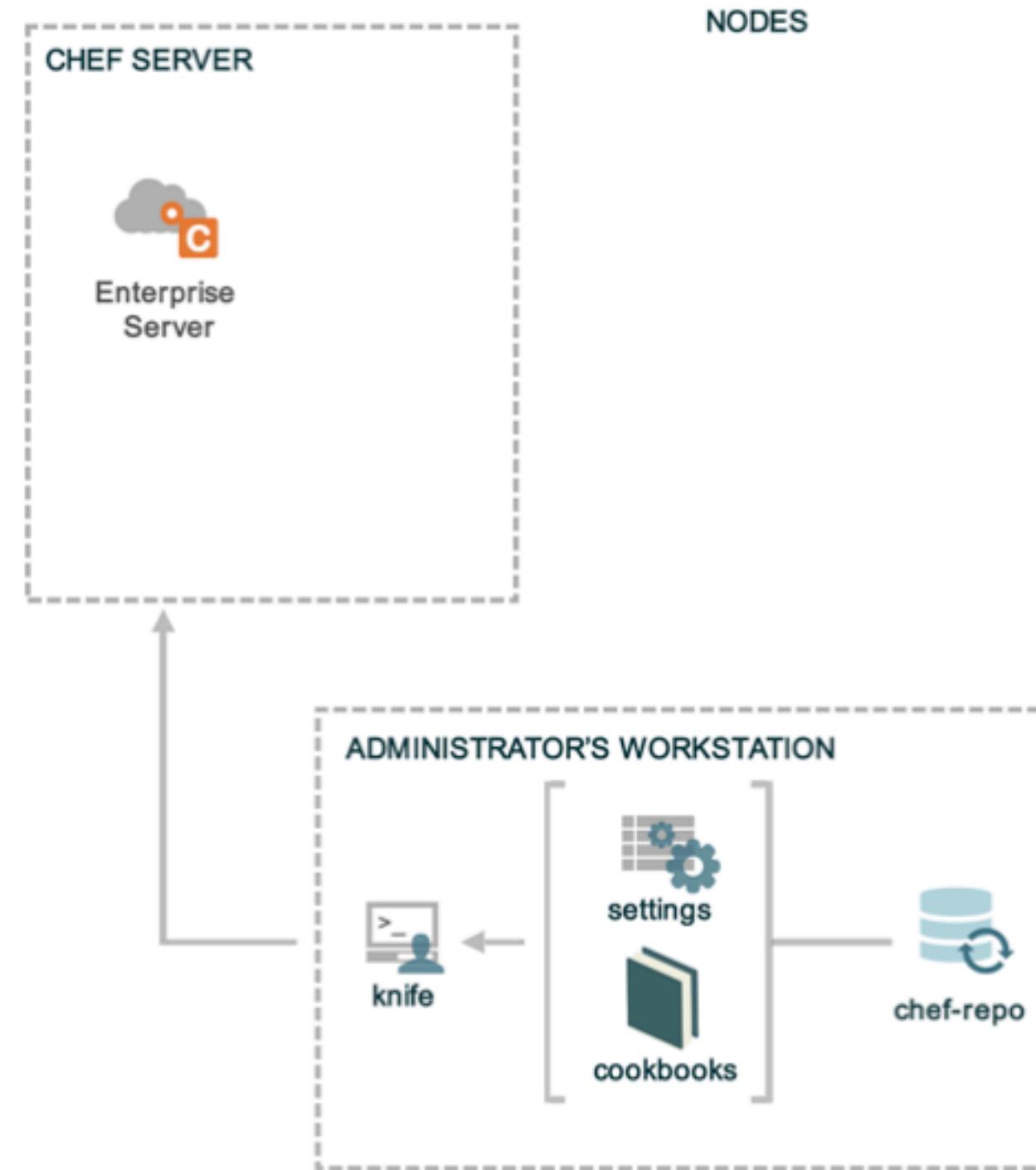
```
[master (root-commit) 2af68fb] add the starter kit from Chef
13 files changed, 360 insertions(+)
create mode 100644 .berkshelf/config.json
create mode 100644 .chef/knife.rb
create mode 100644 .gitignore
create mode 100644 Berksfile
create mode 100644 README.md
create mode 100644 Vagrantfile
create mode 100644 chefignore
create mode 100644 cookbooks/starter/attributes/default.rb
create mode 100644 cookbooks/starter/files/default/sample.txt
create mode 100644 cookbooks/starter/metadata.rb
create mode 100644 cookbooks/starter/recipes/default.rb
create mode 100644 cookbooks/starter/templates/default/sample.erb
create mode 100644 roles/starter.rb
```

Monday, 30 June 14

This is an optional slide!

Commit the files to the git repository

# Checkpoint



Monday, 30 June 14

Chef Server is up and running  
Knife is installed and configured  
Source Code repository has been created

# Review Questions

- What is the chef-repo?
- What is knife?
- What is name of the knife configuration file?
- Where is the knife configuration file located?
- What is your favorite text editor? :)

Monday, 30 June 14

-)  
-)  
-)  
-) knife.rb  
-) <pwd>/./chef/knife.rb or ~./chef/knife.rb or /etc/chef/knife.rb  
-)

# Organization Setup

Setup an Organization

v1.2.3

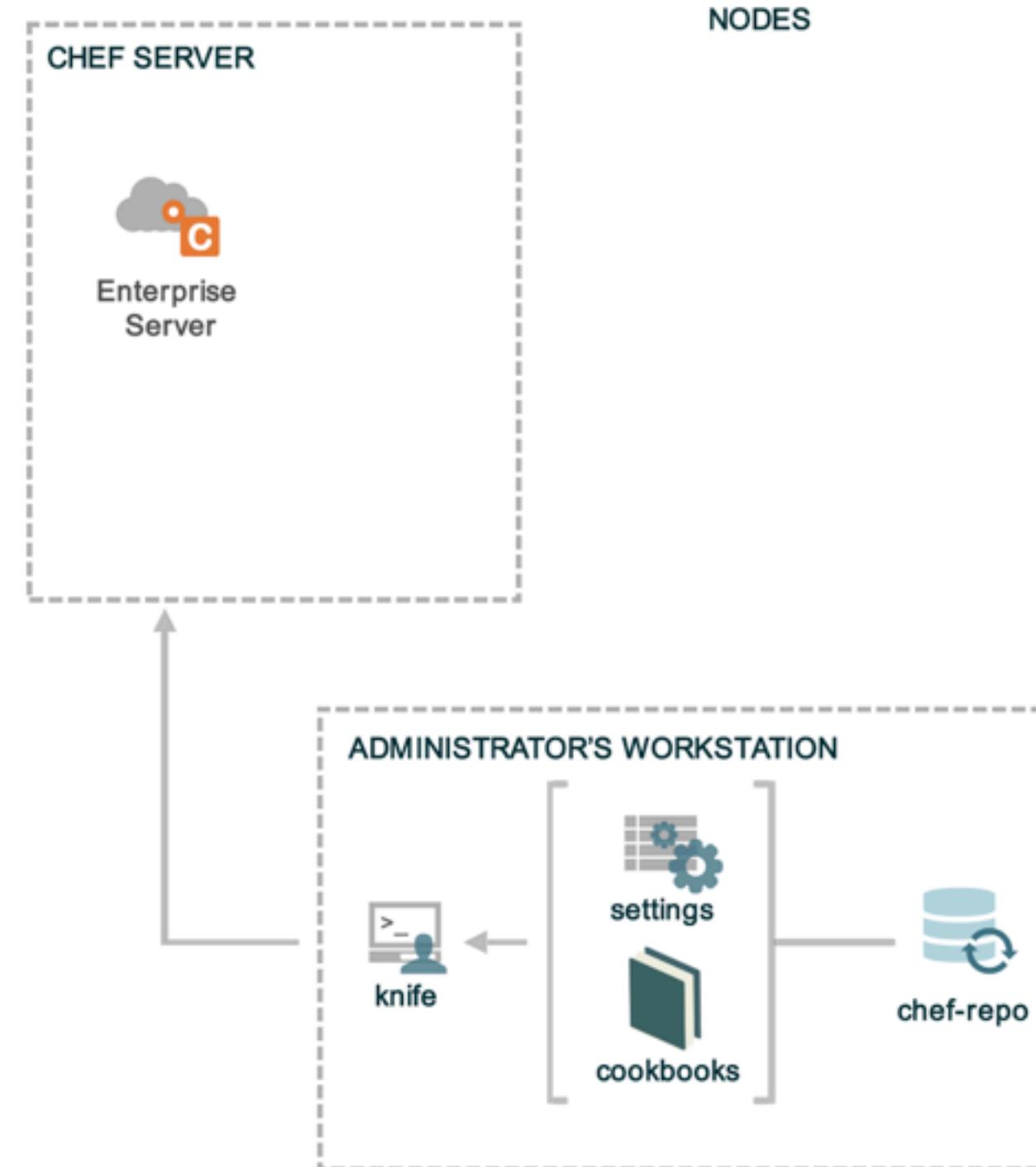


Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Explain the purpose of Organizations
  - Manage your Chef Organization

# Checkpoint

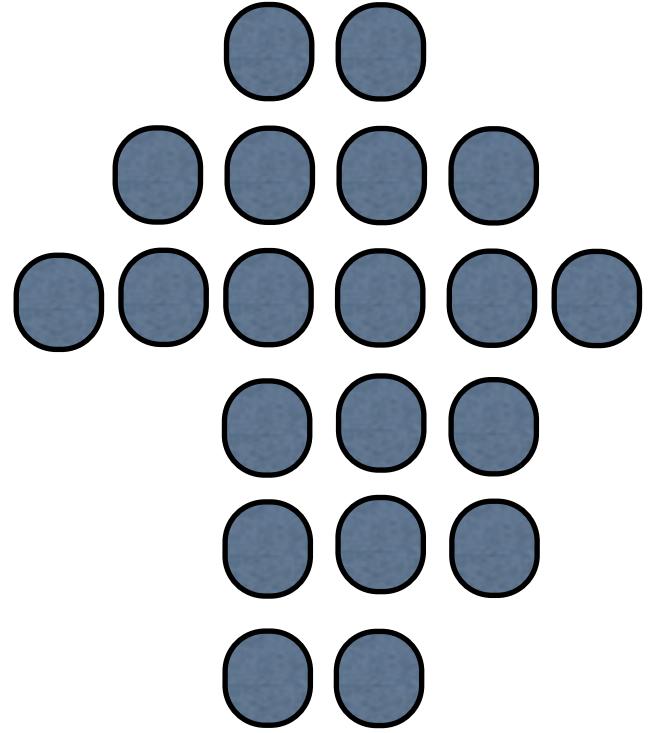


Monday, 30 June 14

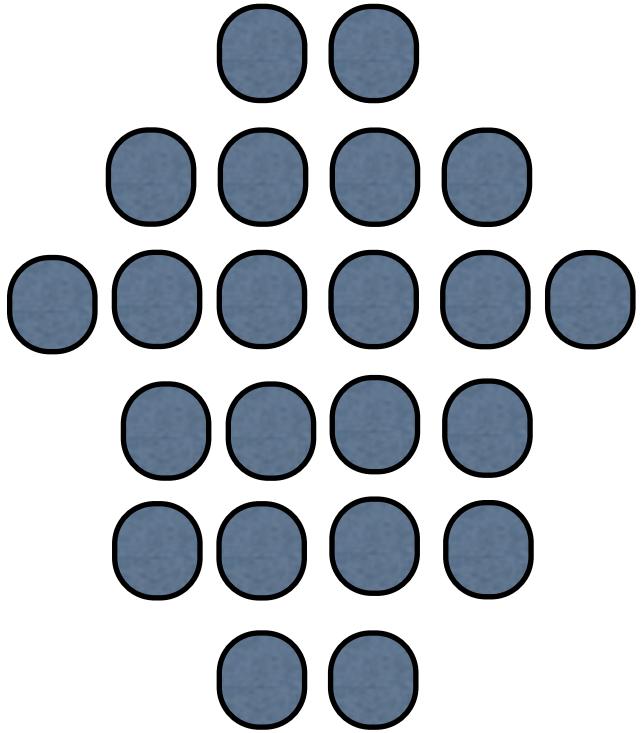
Chef Server is up and running  
Knife is installed and configured  
Source Code repository has been created

# Organizations

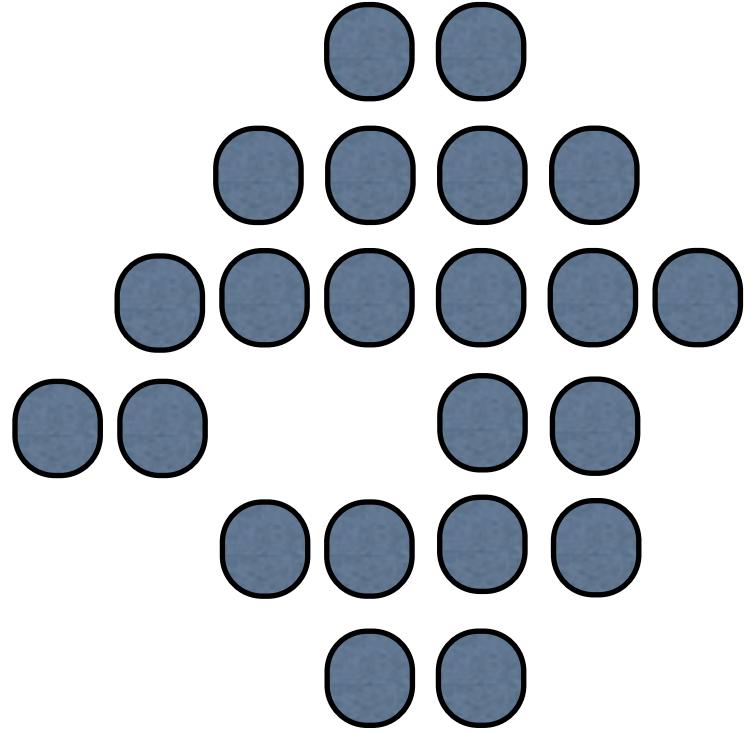
My Infrastructure



Your Infrastructure



Their Infrastructure

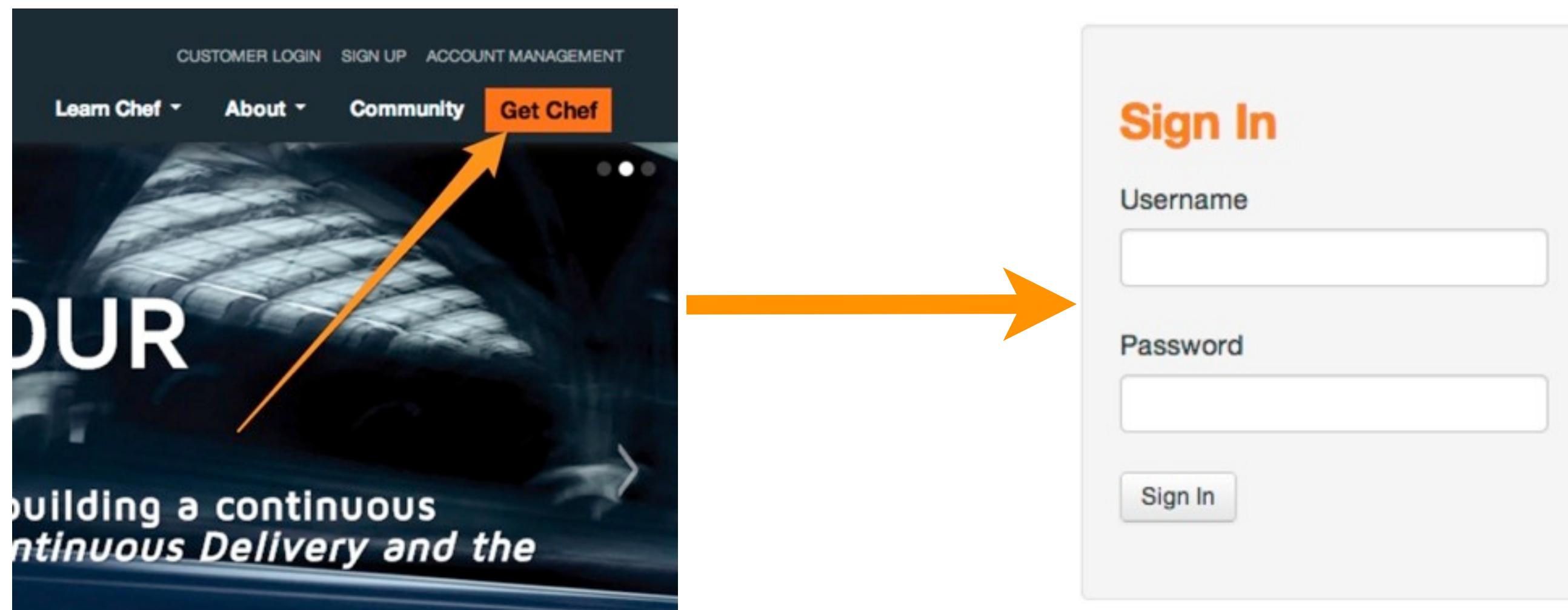


# Organizations

- Provide multi-tenancy in Enterprise Chef
- Nothing is shared between Organizations - they're completely independent
- May represent different
  - Companies
  - Business Units
  - Departments

# Manage Organizations

- Login to your Hosted Enterprise Chef



Monday, 30 June 14

Login to Enterprise Chef to view your nodes

# Organizations

The screenshot shows the Opscode Manage interface. At the top, there is a navigation bar with tabs: Nodes, Policies, and Administrative. The Administrative tab is highlighted with an orange border. On the left, a sidebar menu has "Organizations" selected, indicated by a right-pointing arrow. Other options in the sidebar include Create, Reset Validation Key, Generate Knife Config, Leave Organization, and Starter Kit. Below the sidebar are sections for Users, Groups, and Global Permissions. The main content area is titled "Showing All Organizations" and lists one organization: nhcompany.

Organization
nhcompany

Monday, 30 June 14

Click the Administrative tab to see the Organizations

The name of your organization was specified when signing up for Enterprise Chef.

# Manage Organization

- Reset Validation Key
- Generate Knife Config
- Leave Organization
- Starter Kit

The screenshot shows the Opscode Manage interface. At the top, there are tabs for Nodes, Policies, and Administrative. On the right, it says "Organization nhcompany | Logged in as Nathan Harvey". Below the tabs, there's a sidebar with "OPSCODE MANAGE" and links for Create, Reset Validation Key, Generate Knife Config, Leave Organization, and Starter Kit. The main area shows a table titled "Showing All Organizations" with one row for "nhcompany". A context menu is open over the "nhcompany" row, listing the same five actions: Reset Validation Key, Generate Knife Config, Leave Organization, and Starter Kit.

Monday, 30 June 14

From the Organizations page you can:

Reset the organization's validation key  
Generate a knife.rb  
Leave the Organization  
Recreate and download the starter kit

# Manage Organization

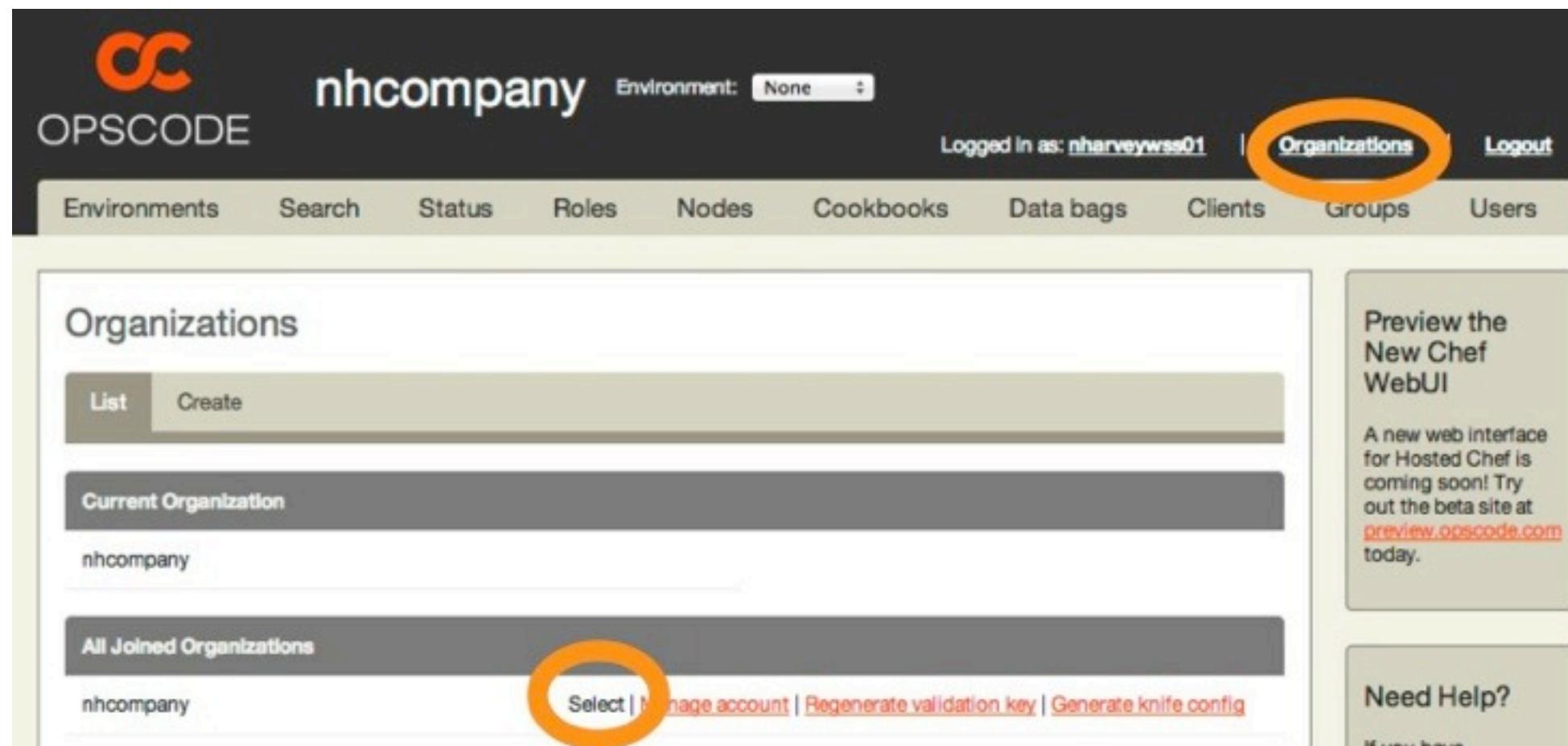
- Each Organization may have multiple Users
- Manage an Organization's Users via the legacy Enterprise Server interface
  - <http://manage.opscode.com>

Monday, 30 June 14

NOTE: as of 2013-10-15 – Inviting users isn't supported. Switch back to [manage.opscode.com](http://manage.opscode.com) to show this functionality if asked / necessary.

# Invite a User

- Navigate to <http://manage.opscode.com>
- Click the "Organizations" link
- Select the appropriate Organization



# Invite a User

- Navigate to the "Users" tab
- Click the "Invite" tab

The screenshot shows the Opscode Chef WebUI interface. At the top, there is a navigation bar with the OPSCODE logo, the organization name "nhcompany", an "Environment: None" dropdown, and user information "Logged in as: nharveywss01". Below the navigation bar, there is a horizontal menu with links: Environments, Search, Status, Roles, Nodes, Cookbooks, Data bags, Clients, Groups, and Users. The "Users" link is highlighted with a large orange circle. Below the menu, there is a section titled "Invite Users to Organization" with two tabs: "List" and "Invite". The "Invite" tab is also highlighted with a large orange circle. The main form area has two input fields: "Username(s)" and "Organization". The "Username(s)" field contains the placeholder text "The username(s) of the User(s) you want to invite to this organization. Separate by comma." and is empty. The "Organization" field contains the text "nhcompany". At the bottom of the form is a single "Invite" button.

# Review Questions

- What is an Organization?
- How do you regenerate the Starter Kit for your Organization?

Monday, 30 June 14

- ) a tenant of Enterprise Chef
- ) via the web ui

# Node Setup

Setup a Node to manage

v1.2.3

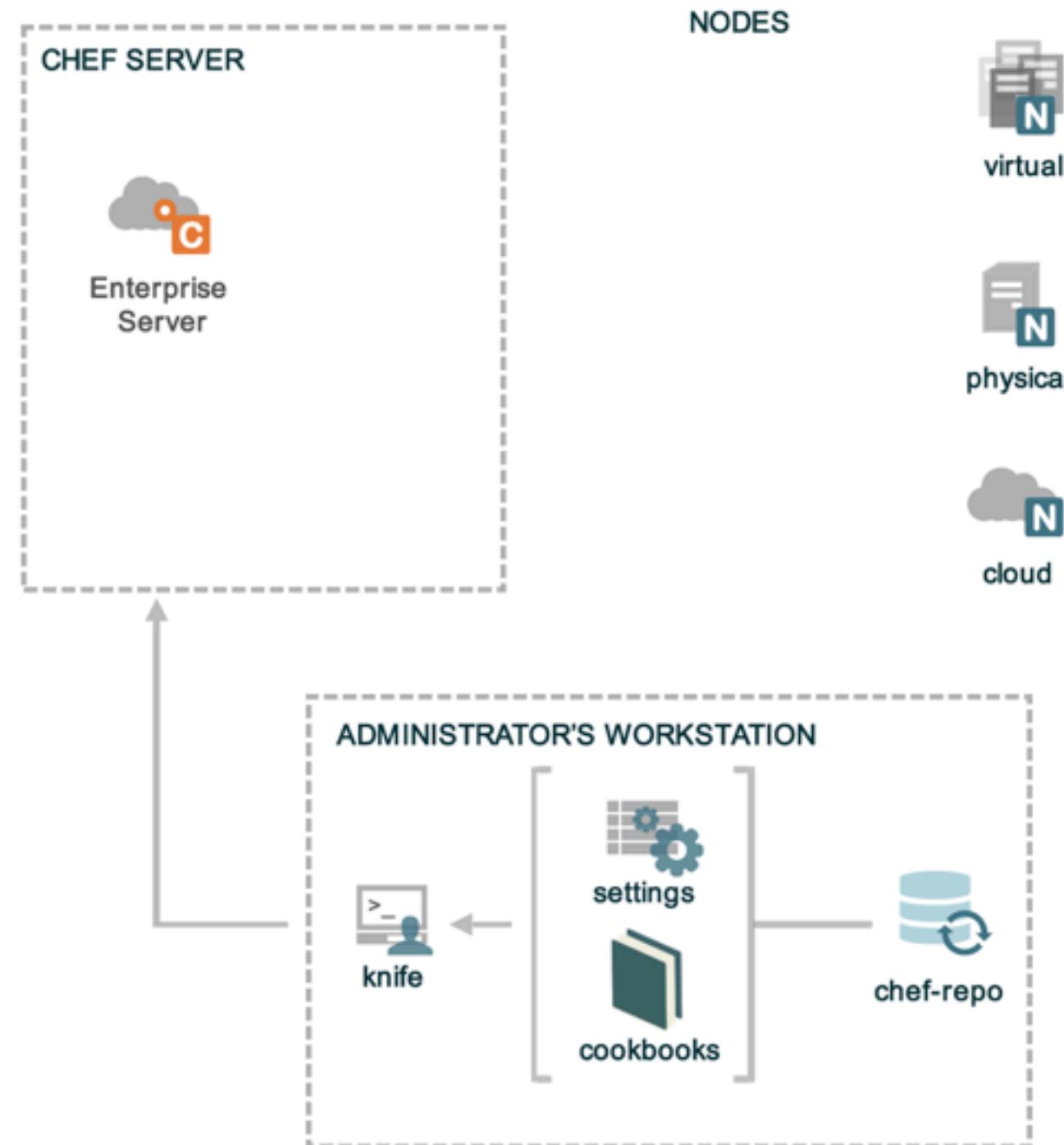


Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Install Chef nodes using "knife bootstrap"
  - Explain how knife bootstrap configures a node to use the Organization created in the previous section
  - Explain the basic configuration needed to run chef-client

# Nodes



Monday, 30 June 14

Chef Server is up and running  
Knife is installed and configured  
Source Code repository has been created

# Nodes

- Nodes represent the servers in your infrastructure  
these may be
  - Physical or virtual servers
  - Hardware that you own
  - Compute instances in a public or private cloud

# We Have No Nodes Yet

The screenshot shows the Opscode Manage web interface. At the top, there's a dark header bar with the "OPSCODE" logo and "MANAGE" text. Below the header, a navigation bar has three tabs: "Nodes" (which is active and highlighted in light blue), "Policies", and "Administrative". On the left side, there's a sidebar with a "Nodes" section containing links for "Delete", "Manage Tags", "Reset Key", "Edit Run List", and "Edit Attributes". The main content area is titled "Showing All Nodes" and contains a message: "There are no items to display." with an information icon.

Monday, 30 June 14

There aren't any nodes yet

# Training Node

- The labs require a node to be managed
- We allow for three different options
  - Bring your own Node
  - Launch an instance of a public AMI on EC2
  - Use the Chef Fundamental training lab

Monday, 30 June 14

Use this if cloudshare or some other on-demand training lab / environment is available.

# Training Node

- The labs require a node to be managed
- We allow for two different options
  - Bring your own Node
  - Launch an instance of a public AMI on EC2

Monday, 30 June 14

Use this if cloudshare or some other on-demand training lab / environment are unavailable.

# Bring Your Own Node

- Use your own Virtual Machine (VM) or Server
- Required for the labs:
  - Ubuntu 10.04+
  - 512 MB RAM
  - 15 GB Disk
  - sudo or root level permissions

# EC2 Public AMI

- Opscode publishes a public AMI on EC2 that may be used
  - Search for ‘oc-training-public’
  - m1.small should be sufficient
  - Open ports 22, 80-90 in security group
- SSH Credentials
  - Login: **opscode**
  - Password: **opscode**

# EC2 Public AMI

- Opscode publishes a public AMI on EC2 that may be used
  - Search for ‘oc-training-public’
  - m1.small should be sufficient
  - Open ports 22, 80-90 in security group
- SSH Credentials
  - Login: **opscode**
  - Password: **opscode**

Never use this for  
anything other  
than this class!

# Fundamentals Webinar Lab

- Register and login to CloudShare (see invite)
- Start Using This Environment

## Chef Fundamentals Webinar

Your dedicated hands-on environment is just a click away.

We believe you shouldn't have to waste time copying gigabytes of software, shipping machines, or traveling, just to get your IT into people's hands.

When you click the 'Start Using' button to your right, you'll have instant access worldwide to a full, enterprise-grade IT environment, available through your browser, mobile device, or dedicated client and VPN connection. You can connect your existing datacenter servers, change configurations, load new software, and provide this functionality to others. It's just like an on-premises data center, but with more flexibility and none of the headaches of backup, access, replication, logging, SLAs, and other annoyances - we take care of all of that for you.

CloudShare's SaaS platform is a quick and easy way to share copies of your complex IT environments, online. So you can collaborate with customers, partners, and colleagues - for Demos, Proofs-of-Concept, Training, or other enterprise applications.

Questions? Click [here](#) to email this environment's author or click [here](#) for CloudShare support.



# Lab - More details

## Chef Fundamentals Webinar

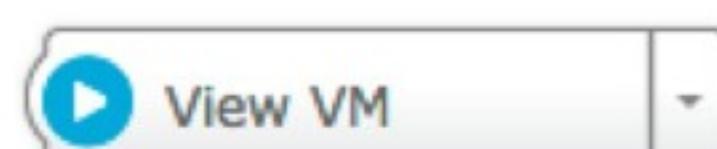
Ubuntu 10

**target1**

**Description:** OS: Ubuntu 10.04 Server  
Spec: 16 GB HD / 0.5 GB RAM  
**OS:** Linux  
**Status:** Stopped

[More details ►](#)



 View VM

 Reboot VM

# Lab - External Address

Chef Fundamentals Webinar

 **target1**

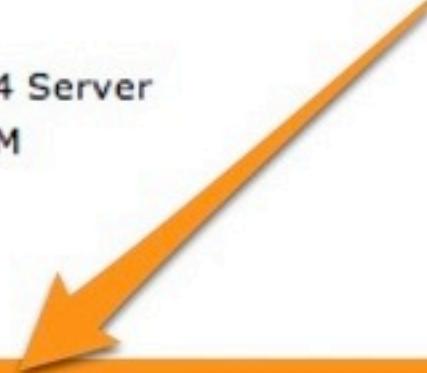
**Description:** OS: Ubuntu 10.04 Server  
Spec: 16 GB HD / 0.5 GB RAM  
**OS:** Linux  
**State:** Running  
[Hide details ▾](#)

**External Address:** [uv01nutokp4f6ejqrwf.vm.cld.sr](http://uv01nutokp4f6ejqrwf.vm.cld.sr) 

**Total Memory:** 512 MB  
**Disk Size:** 16 GB  
**CPU:** 1  
**The machine was prepared in 2 minutes**

**Credentials (show password)**

**Auto-login:** sysadmin (local user)  
**Username:** sysadmin  
**Password:** \*\*\*\*\*



Monday, 30 June 14

Copy the External Address. This will be used to ssh into the server.

# Lab - Login

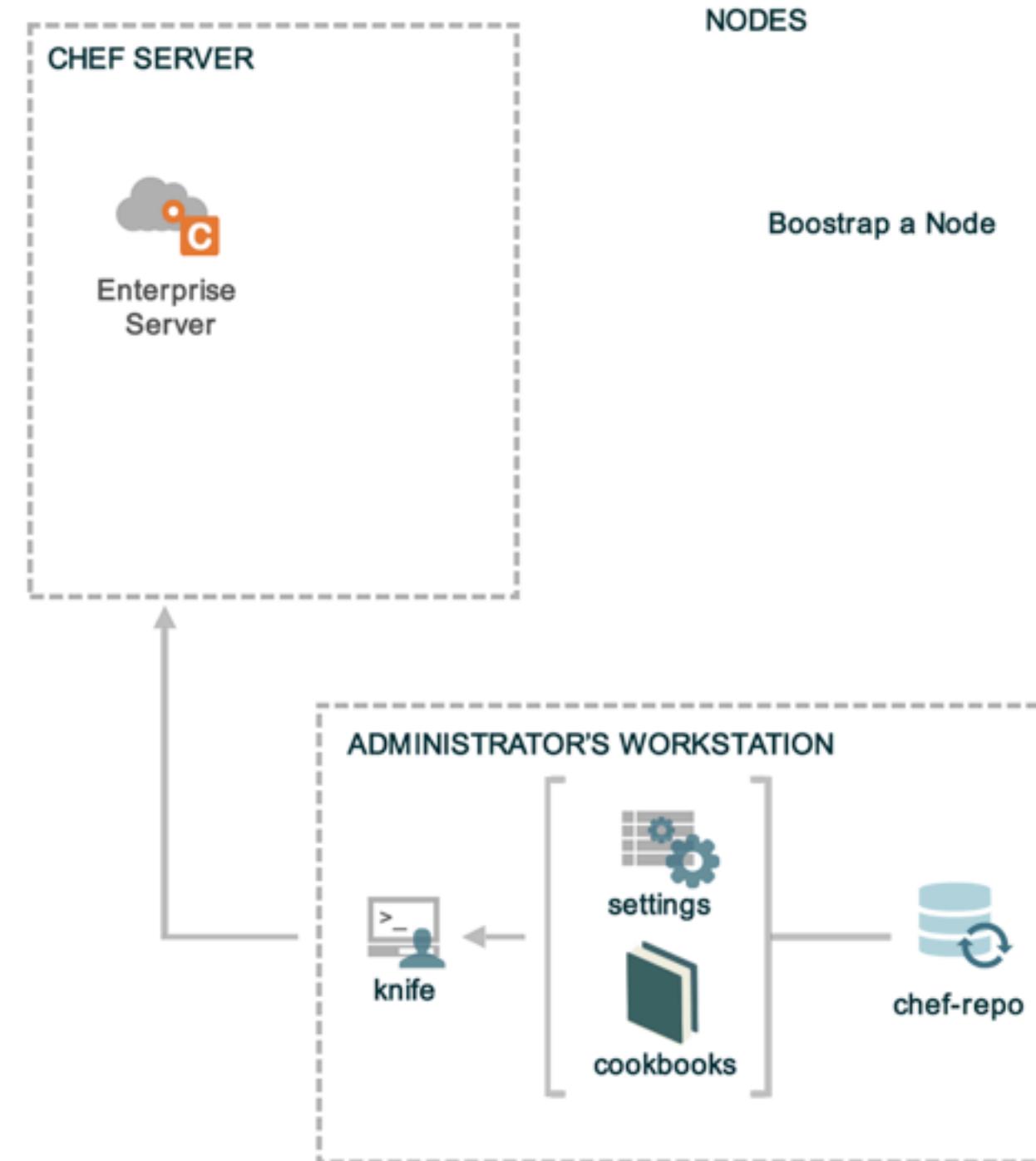
```
$ ssh opscode@<EXTERNAL_ADDRESS>
```

```
The authenticity of host 'ec2-54-211-119-145.compute-1.amazonaws.com  
(54.211.119.145)' can't be established.  
RSA key fingerprint is b9:a6:89:f5:3d:ad:  
33:b6:c5:90:66:e7:b3:30:f1:d8.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added  
'ec2-54-211-119-145.compute-1.amazonaws.com,54.211.119.145' (RSA) to  
the list of known hosts.  
opscode@ec2-54-211-119-145.compute-1.amazonaws.com'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

# Checkpoint



Monday, 30 June 14

Chef Server is up and running  
Knife is installed and configured  
Source Code repository has been created

# "Bootstrap" the Target Instance

```
$ knife bootstrap IPADDRESS --sudo -x opscode -P opscode -N "node1"
```

```
Bootstrapping Chef on 23.22.141.166
23.22.141.166 knife sudo password:
Enter your password:
...
...
23.22.141.166 Creating a new client identity for node1 using the validator key.
23.22.141.166 resolving cookbooks for run list: []
23.22.141.166 Synchronizing Cookbooks:
23.22.141.166 Compiling Cookbooks...
23.22.141.166 [2013-11-05T16:11:13+00:00] WARN: Node node1 has an empty run
list.
23.22.141.166 Converging 0 resources
23.22.141.166 Chef Client finished, 0 resources updated
```

Monday, 30 June 14

**This command should be entered on a single line.**  
(using the EC2 fqdn sucks. Have your students name the node.)

The IPADDRESS is your VM's IP or external name

If you get prompted for the password again, this is \*sudo\* not Chef. The -P is passed to the SSH connection.

It will take a few minutes for this command to run, so let's talk about what it is doing.

local workstation

managed node  
(VM)

Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the chef\_server\_url, validation\_client\_name for the Chef Server and the contents of the validation\_key from the knife.rb file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (-r '[..]').

```
$ knife bootstrap IPADDRESS --sudo -x USERNAME -P PASSWORD -N node1
```

local workstation

managed node  
(VM)

Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the chef\_server\_url, validation\_client\_name for the Chef Server and the contents of the validation\_key from the knife.rb file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (-r '[..]').

```
$ knife bootstrap IPADDRESS --sudo -x USERNAME -P PASSWORD -N node1
```

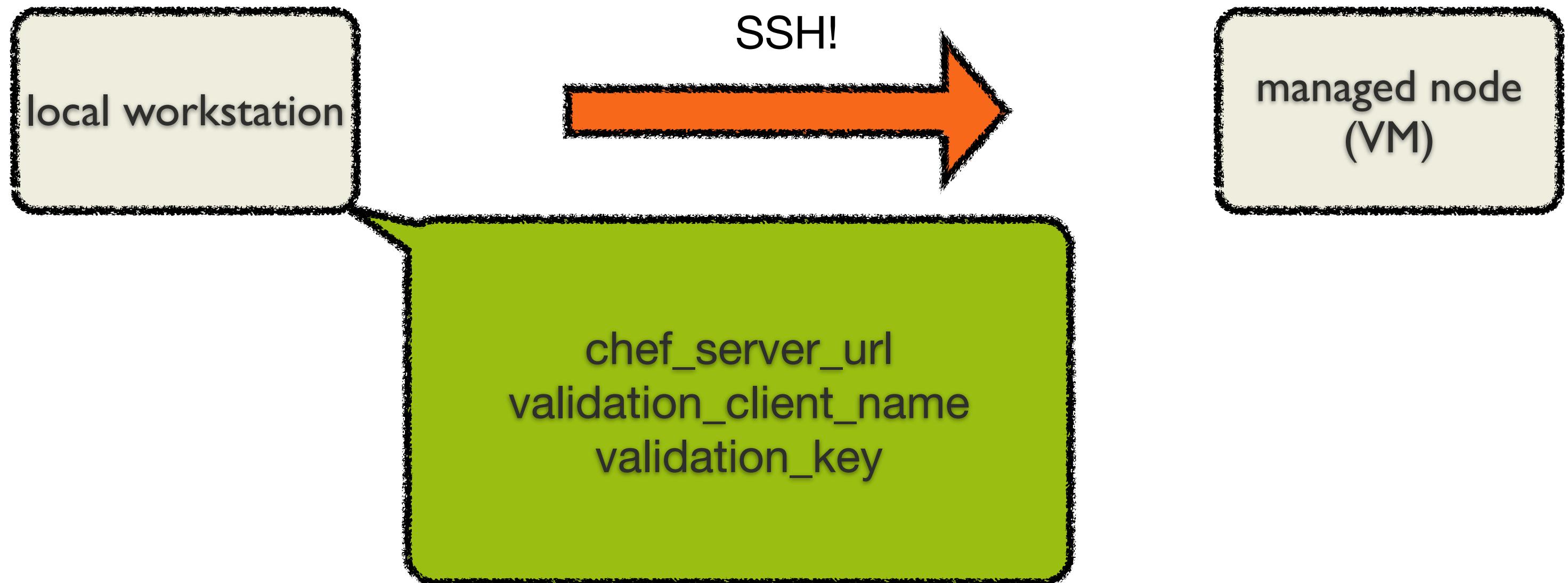


Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the chef\_server\_url, validation\_client\_name for the Chef Server and the contents of the validation\_key from the knife.rb file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (-r '[..]').

```
$ knife bootstrap IPADDRESS --sudo -x USERNAME -P PASSWORD -N node1
```



Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the `chef_server_url`, `validation_client_name` for the Chef Server and the contents of the `validation_key` from the `knife.rb` file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (`-r '[..]'`).

```
$ knife bootstrap IPADDRESS --sudo -x USERNAME -P PASSWORD -N node1
```

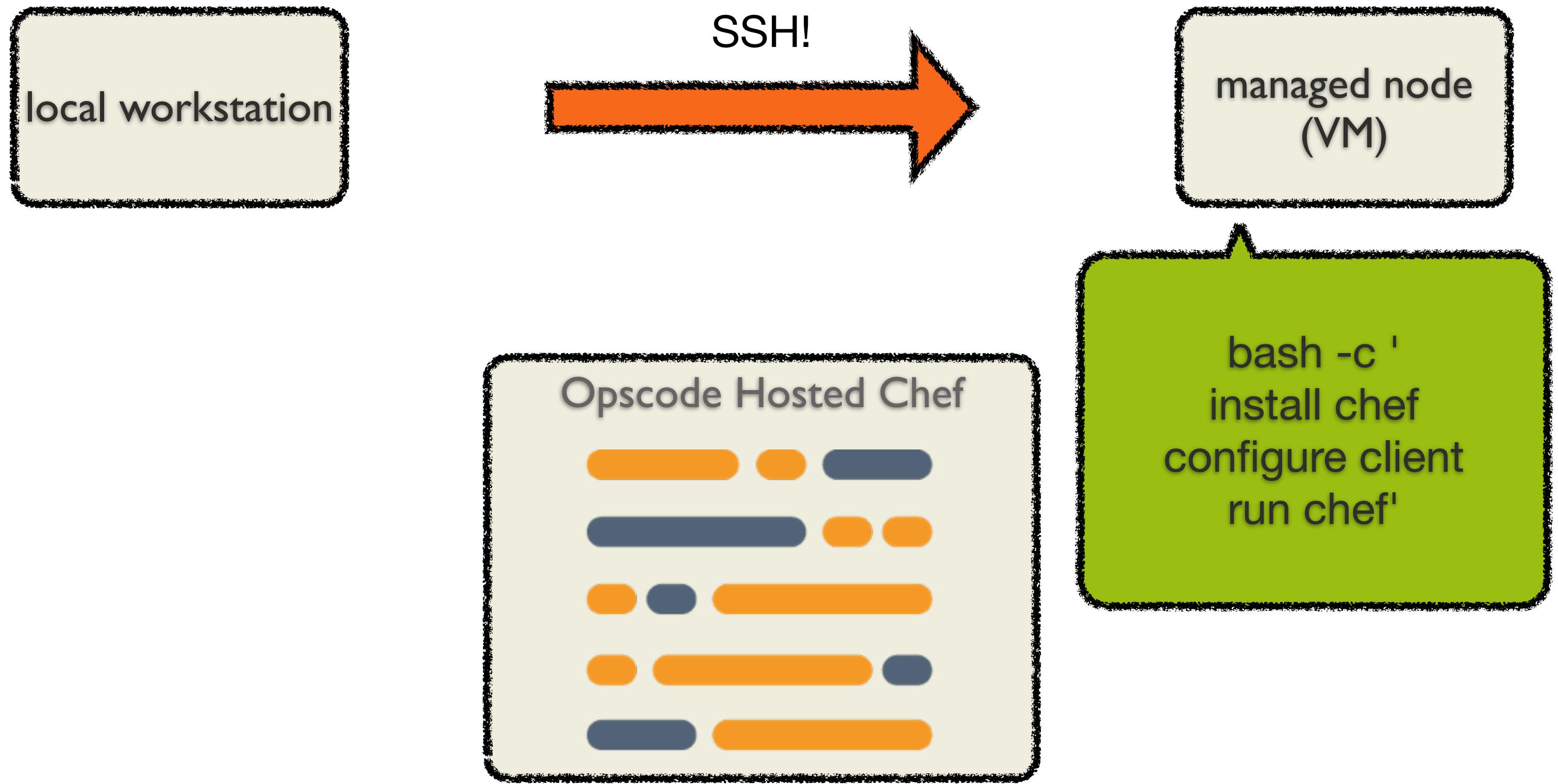


Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the chef\_server\_url, validation\_client\_name for the Chef Server and the contents of the validation\_key from the knife.rb file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (-r '[..]').

```
$ knife bootstrap IPADDRESS --sudo -x USERNAME -P PASSWORD -N node1
```

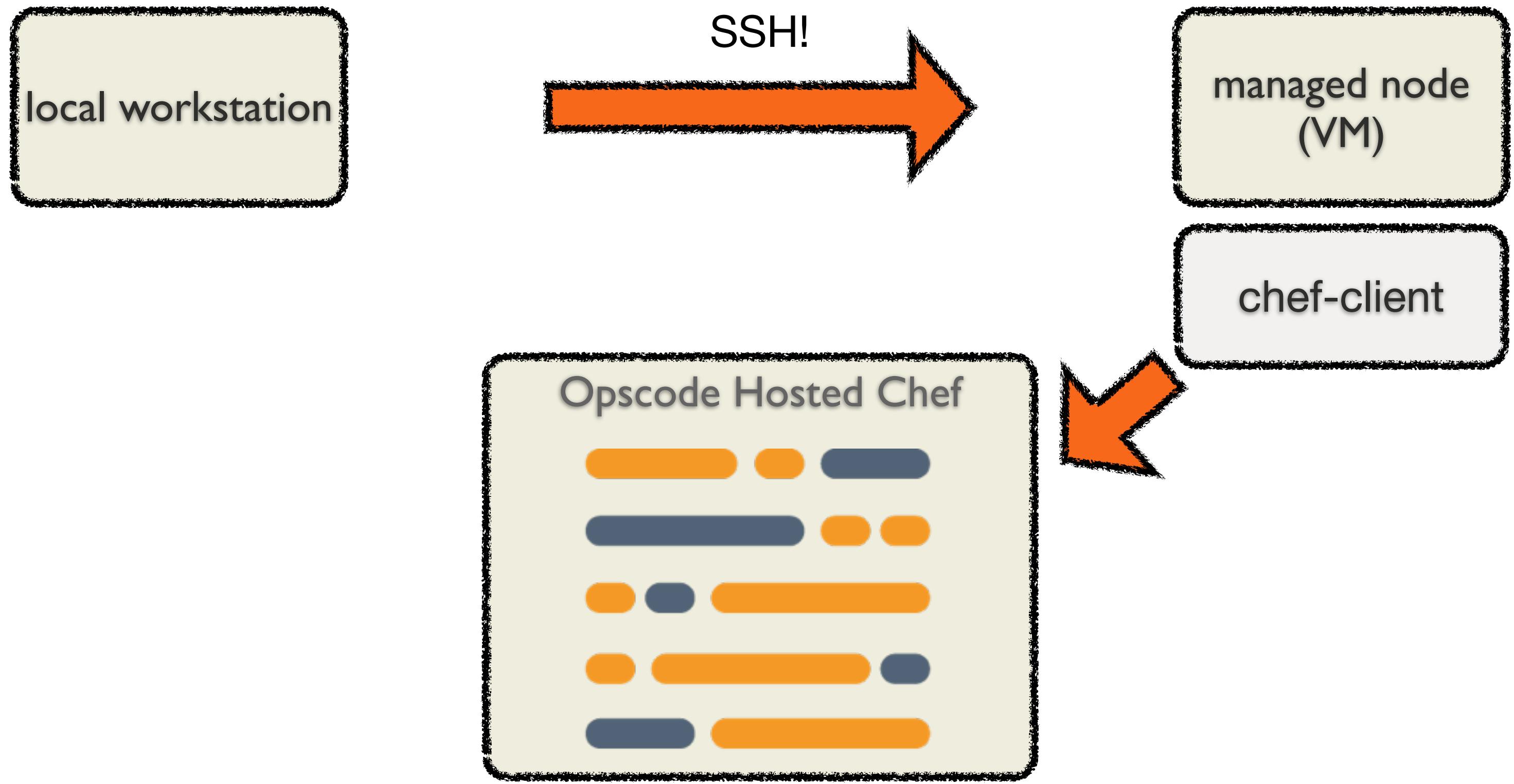


Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the `chef_server_url`, `validation_client_name` for the Chef Server and the contents of the `validation_key` from the `knife.rb` file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (`-r '[..]'`).

```
$ knife bootstrap IPADDRESS --sudo -x USERNAME -P PASSWORD -N node1
```



Monday, 30 June 14

What does this bootstrap command do?

1. Connect to the VM via SSH, using the specified user (opscode) and password (opscode)
2. Renders the specified template (chef-full) with the `chef_server_url`, `validation_client_name` for the Chef Server and the contents of the `validation_key` from the `knife.rb` file.
3. We have the Opscode Hosted Chef server defined as the server url.
4. The template is executed as a script under sudo on the VM, the script: installs chef-full package, configures the system to connect to the server, and runs chef-client with the specified run list (`-r '[..]'`).

# 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

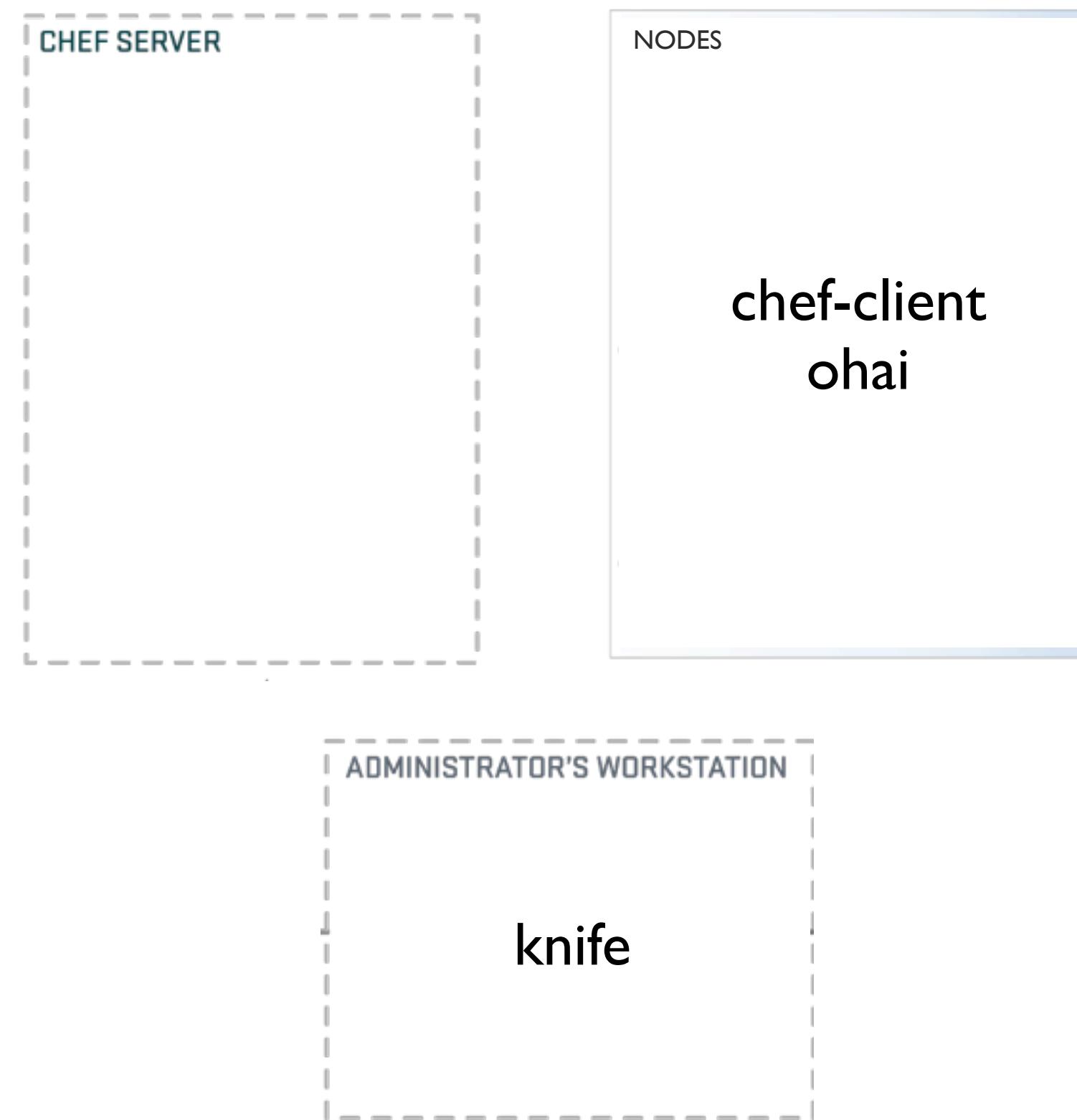
Monday, 30 June 14

A quick mention that we just used the omnibus installer which puts Chef and all of its dependencies on the workstation. The tools are installed under /opt/chef (linux/mac) or C:\Chef (Windows).

Also mention that 'gem install chef' is also a valid installation method if students are already using rvm, rbenv, or chruby to manage their Ruby installations or if they'd prefer to use the system ruby (which should be at version 1.9 or better to use this method).

Opscode manages an open source project called "Omnibus". Use omnibus to create OS-specific packages.

# Workstation or Node?



Monday, 30 June 14

We've just installed chef-client, knife, ohai and ruby along with any supporting ruby gems. This is exactly the same as gets installed on a node during bootstrap. Difference is the node does not use knife, while the workstation client does not use chef-client – but each could be configured to perform either role!

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

```
$ ssh opscode@IPADDRESS  
  
opscode@node1:~$ ls /etc/chef  
client.pem  client.rb  first-boot.json  validation.pem  
  
opscode@node1:~$ which chef-client  
/usr/bin/chef-client
```

# Examine /etc/chef/client.rb

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

```
log_level      :auto
log_location    STDOUT
chef_server_url "https://api.opscode.com/organizations/ORGNAME"
validation_client_name "ORGNAME-validator"
node_name       "node1"
```

# Change the log level on your test node

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

```
log_level          :info
log_location       STDOUT
chef_server_url   "https://api.opscode.com/organizations/ORGNAME"
validation_client_name "ORGNAME-validator"
node_name          "node1"
```

- Set the default log level for chef-client to **:info**
- More configuration options can be found on the docs site:  
[http://docs.opscode.com/config\\_rb\\_client.html](http://docs.opscode.com/config_rb_client.html)

Monday, 30 June 14

We change the `log_level` so that later sessions have more detailed log outputs. The automatic setting for logging since Chef 11 has minimal output that hides some of the lesson information. If you are using Chef 10, then the log level will be `:info` by default.

Use ‘`sudoedit`’ or ‘`sudo vim`’ or whatever is your favorite editor is.

# View Node on Chef Server

- Login to your Hosted Enterprise Chef



Monday, 30 June 14

Login to Enterprise Chef to view your nodes

# View Node on Chef Server

The screenshot shows the Opscode Manage web interface. The top navigation bar has tabs for Nodes, Policies, and Administrative. The Nodes tab is selected, showing a table titled "Showing All Nodes". A single row is present in the table:

Node Name	Platform	FQDN	IP Address
node1	ubuntu	ip-10-239-4-99.ec2.i...	10.239.4.99

Below the table, a section titled "Node: node1" contains three tabs: Details, Attributes, and Permissions. Under "Details", it says "Last Check In: 20 Minutes Ago" (2013-10-31 10:41:24 UTC). Under "Attributes", it says "Uptime: 13 Minutes" (Since 2013-10-31 10:48:30 UTC). At the bottom, there is a "Tags" section with a "+ Add" button and a message: "There are no items to display."

Monday, 30 June 14

# View Node on Chef Server

The screenshot shows the Opscode Manage interface for viewing a node. The top navigation bar includes 'Nodes' (selected), 'Policies', and 'Administrative'. On the left, a sidebar under 'Nodes' lists actions: Delete, Manage Tags, Reset Key, Edit Run List, and Edit Attributes. The main content area displays 'Showing All Nodes' with a table:

Node Name	Platform	FQDN	IP Address
node1	ubuntu	ip-10-239-4-99.ec2.i...	10.239.4.99

Below the table, a section titled 'Node: node1' contains tabs for 'Details', 'Attributes' (selected), and 'Permissions'. The 'Attributes' tab shows expanded data:

Expand All   Collapse All

tags:  
+ languages  
+ kernel  
os: linux  
os\_version: 3.2.0-32-virtual  
ohai\_time: 1383216084.508307  
+ network  
+ counters  
hostname: ip-10-239-4-99  
fqdn: ip-10-239-4-99.ec2.internal  
domain: ec2.internal  
ipaddress: 10.239.4.99

Monday, 30 June 14

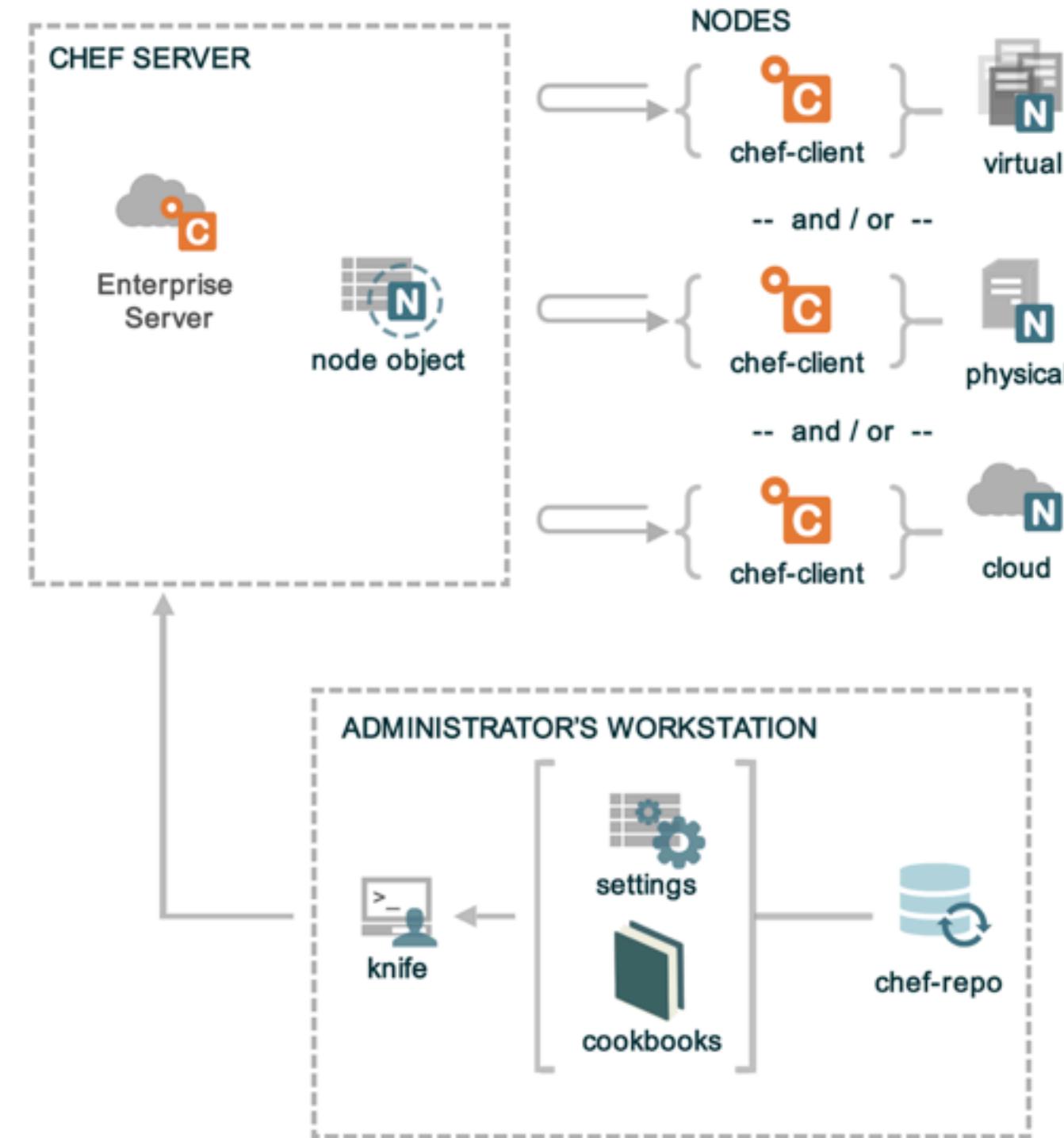
# Node

- The node is registered with Chef Server
- The Chef Server displays information about the node
- This information comes from Ohai

# Ohai

```
"languages": {
  "ruby": {
    },
    "perl": {
      "version": "5.14.2",
      "archname": "x86_64-linux-gnu-thread-multi"
    },
    "python": {
      "version": "2.7.3",
      "builddate": "Aug 1
2012, 05:14:39"
    },
    "php": {
      "version": "5.3.10-1ubuntu3.6",
      "builddate": "(cli)
(built: Mar"
    }
  },
  "kernel": {
    "name": "Linux", "release": "3.2.0-32-virtual",
    "version": "#51-Ubuntu SMP Wed
Sep 26 21:53:42 UTC 2012",
    "machine": "x86_64",
    "modules": {
      "isofs": {
        "size": "40257",
        "refcount": "0"
      },
      "acpiphp": {
        "size": "24231",
        "refcount": "0"
      }
    },
    "os": "GNU/Linux"
  },
  "os": "linux",
  "os_version": "3.2.0-32-virtual",
  "ohai_time": 1369328621.3456137,
  "network": {
    "interfaces": {
      "lo": {
        "mtu": "16436",
        "flags": [
          "LOOPBACK", "UP", "LOWER_UP"
        ],
        "encapsulation": "Loopback",
        "addresses": {
          "127.0.0.1": {
            "family": "inet",
            "netmask": "255.0.0.0",
            "scope": "Node"
          },
          "::1": {
            "family": "inet6",
            "scope": "Node"
          }
        }
      },
      "eth0": {
        "type": "eth",
        "number": "0"
      }
    }
  }
}
```

# Checkpoint



Monday, 30 June 14

Chef Server is up and running

Knife is installed and configured

Source Code repository has been created

# Review Questions

- Where is the chef-client configuration file?
- What is the command to run chef?
- What does a knife bootstrap do?

Monday, 30 June 14

- ) /etc/chef
- ) chef-client
- ) Installs the chef-client on a target system so that it can run as a chef-client and communicate with a server.

# Chef 101 Terminology

So we're on the same page...

v1.2.3



Monday, 30 June 14

Before we go further, let's get all the terminology out on the table.

These slides can simply be shown and read quickly, they're for awareness, and will be explained in further detail throughout the workshop.

Some terms are intentionally left out of this list. We'll get to them in turn.

# **Configured, or managed systems are called Nodes**

Monday, 30 June 14

A node is a thing that Chef is managing. It can be an Ec2 instance, a Virtual Machine, a Rackmounted server in a data center.

# **chef-client runs on your nodes**

Monday, 30 June 14

When we're configuring systems with Chef, we run the client program, "chef-client" on the individual nodes. They're responsible for keeping themselves configured.

# **chef-client talks to a Chef Server**

We're using Hosted Chef today;  
you can buy Private Chef for your own data center

Monday, 30 June 14

The chef-client program connects to the configured Chef Server.

# **API Clients authenticate with RSA keys**

**chef-client and knife are both API clients**

**The server has the public key**

Monday, 30 June 14

API Clients are the thing that authenticate to the Server using RSA keypairs. Chef-client uses an API client on behalf of the system it is running as,

# **Knife is the command-line tool for Chef.**

Monday, 30 June 14

Knife is a plugin-based system that includes a number of plugins that work with the Chef server, and it can be extended for other purposes as we'll see!

# Dissecting your first chef-client run

The Anatomy of a Chef run

v1.2.3



Monday, 30 June 14

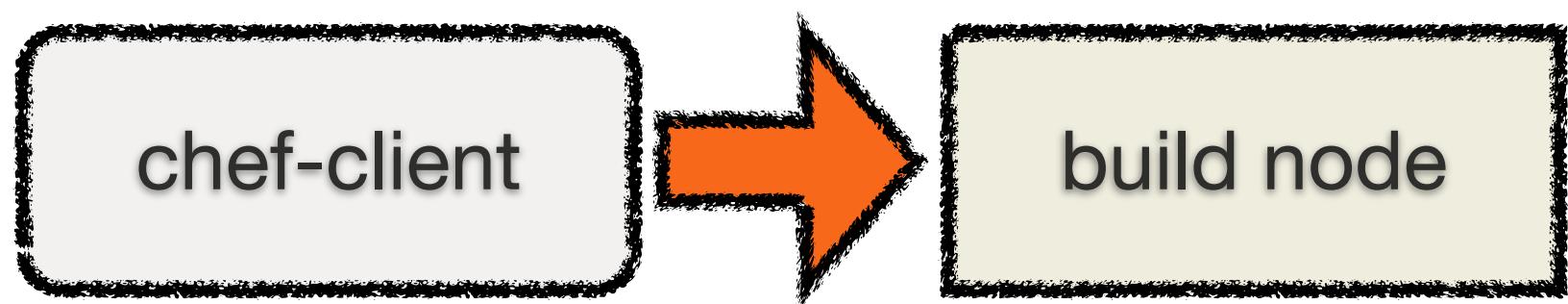
# Lesson Objectives

- After completing the lesson, you will be able to
  - List the steps taken by a chef-client during a run
  - Explain the basic security model of Chef

## chef-client

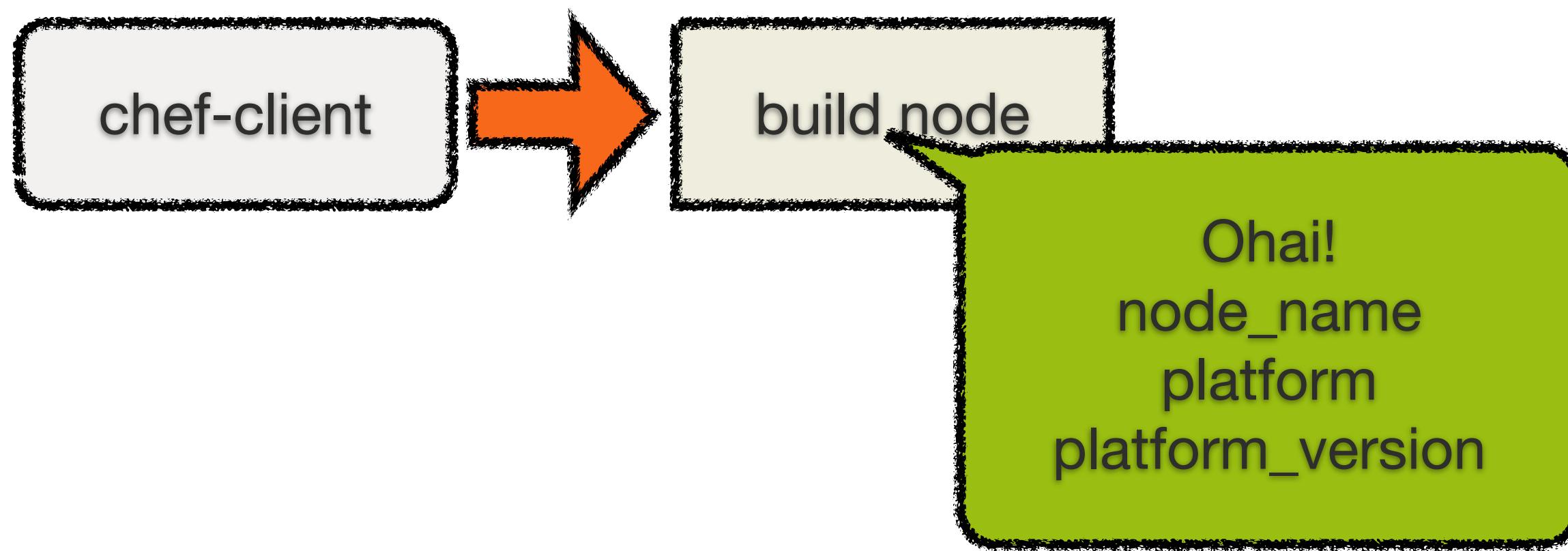
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



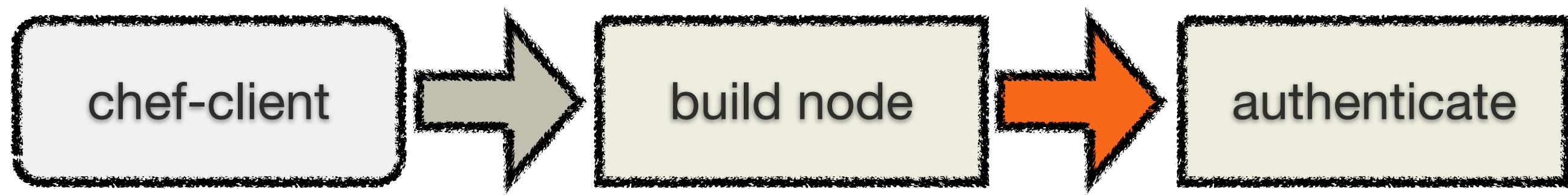
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



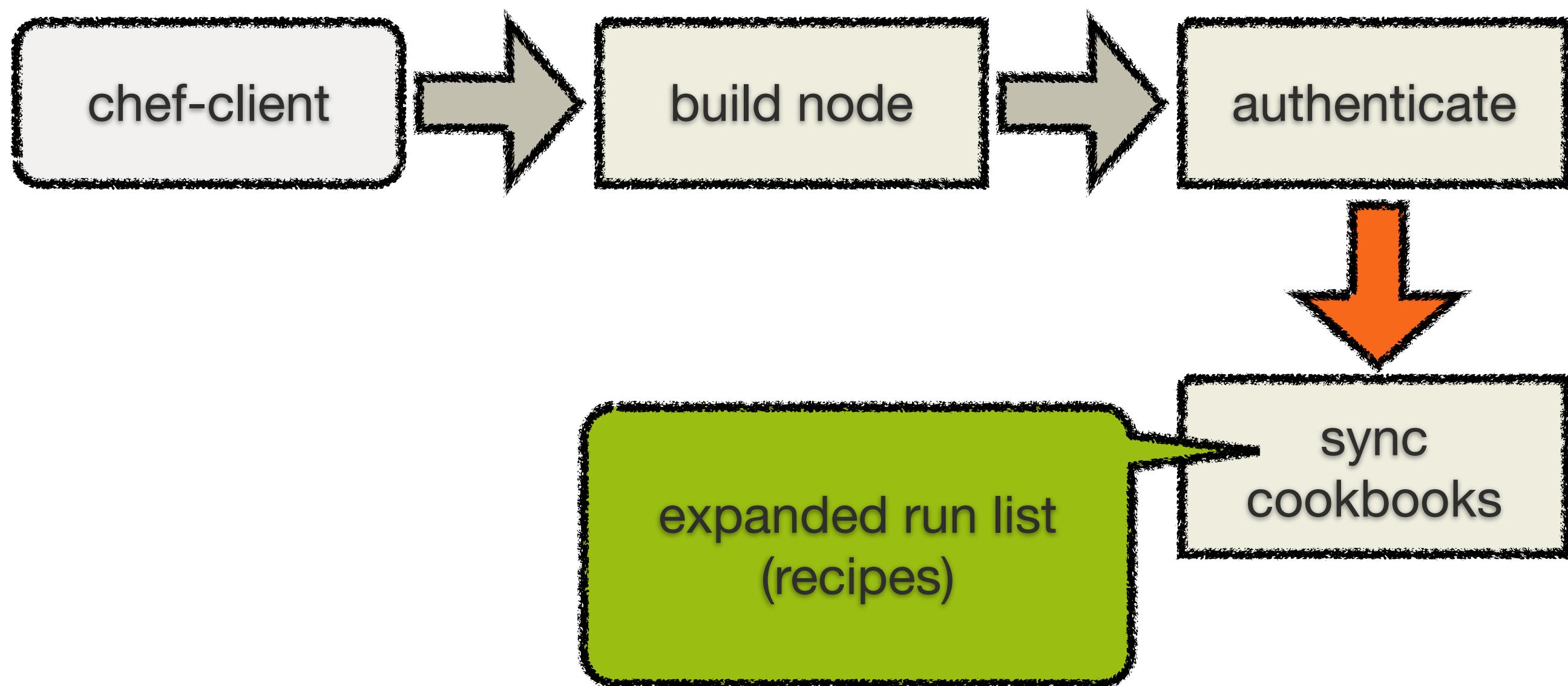
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



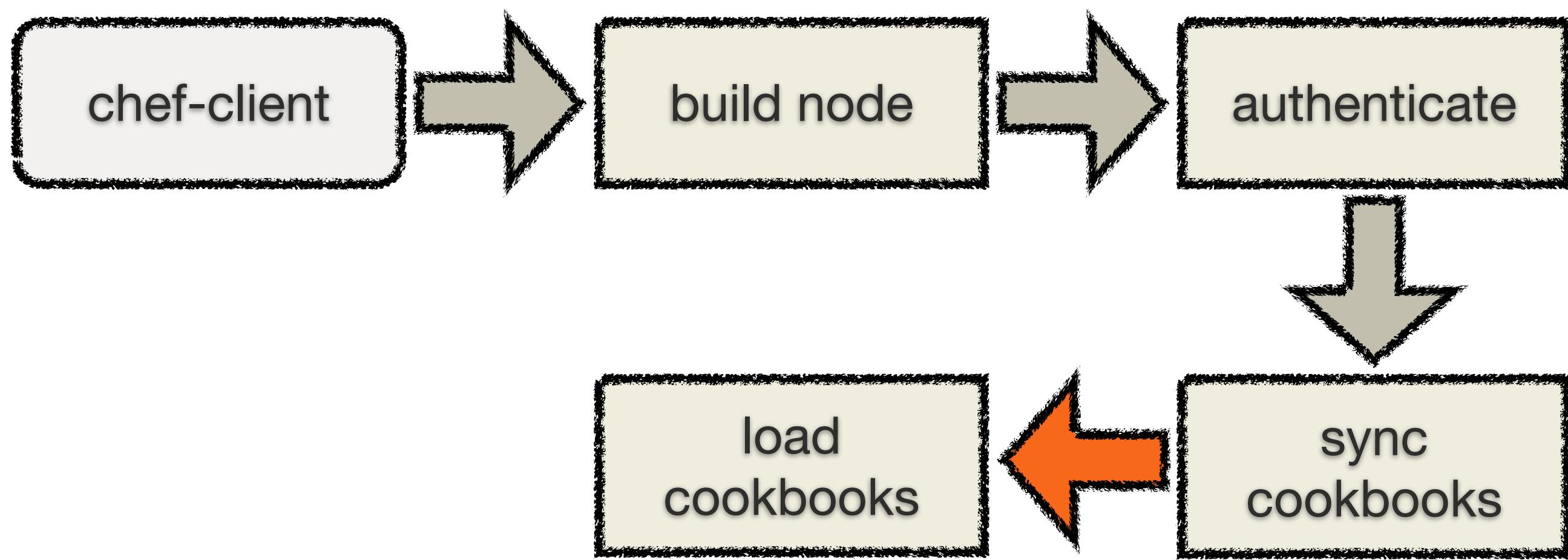
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



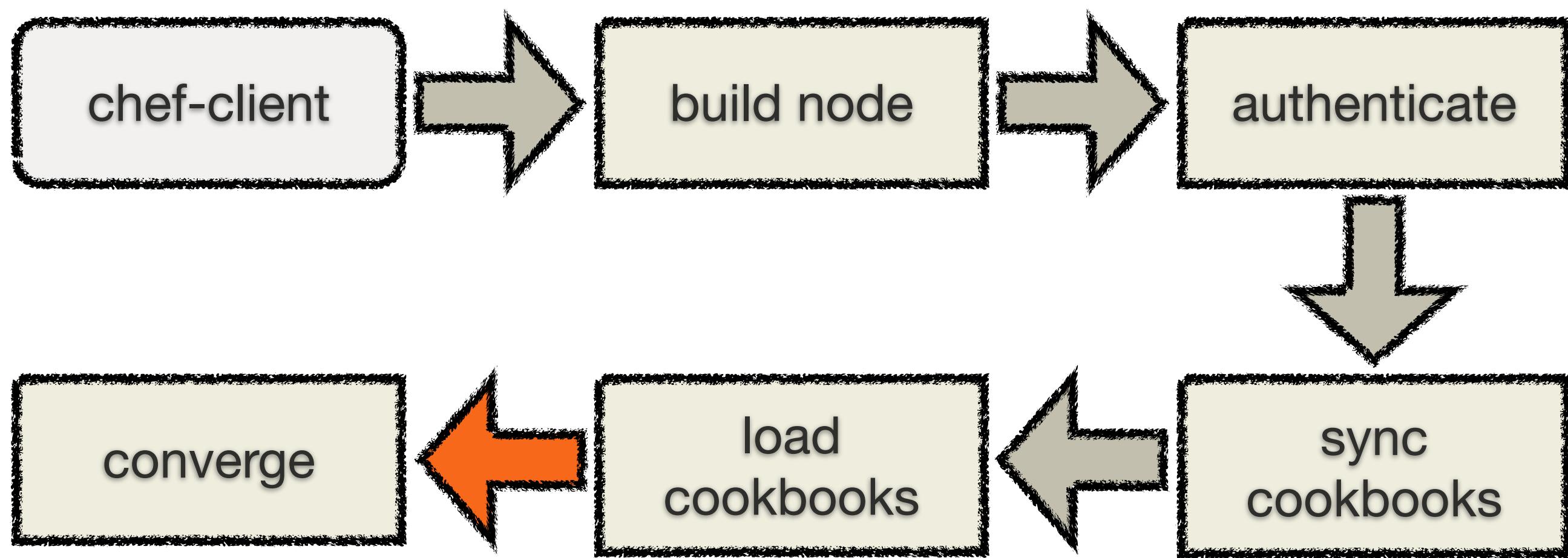
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



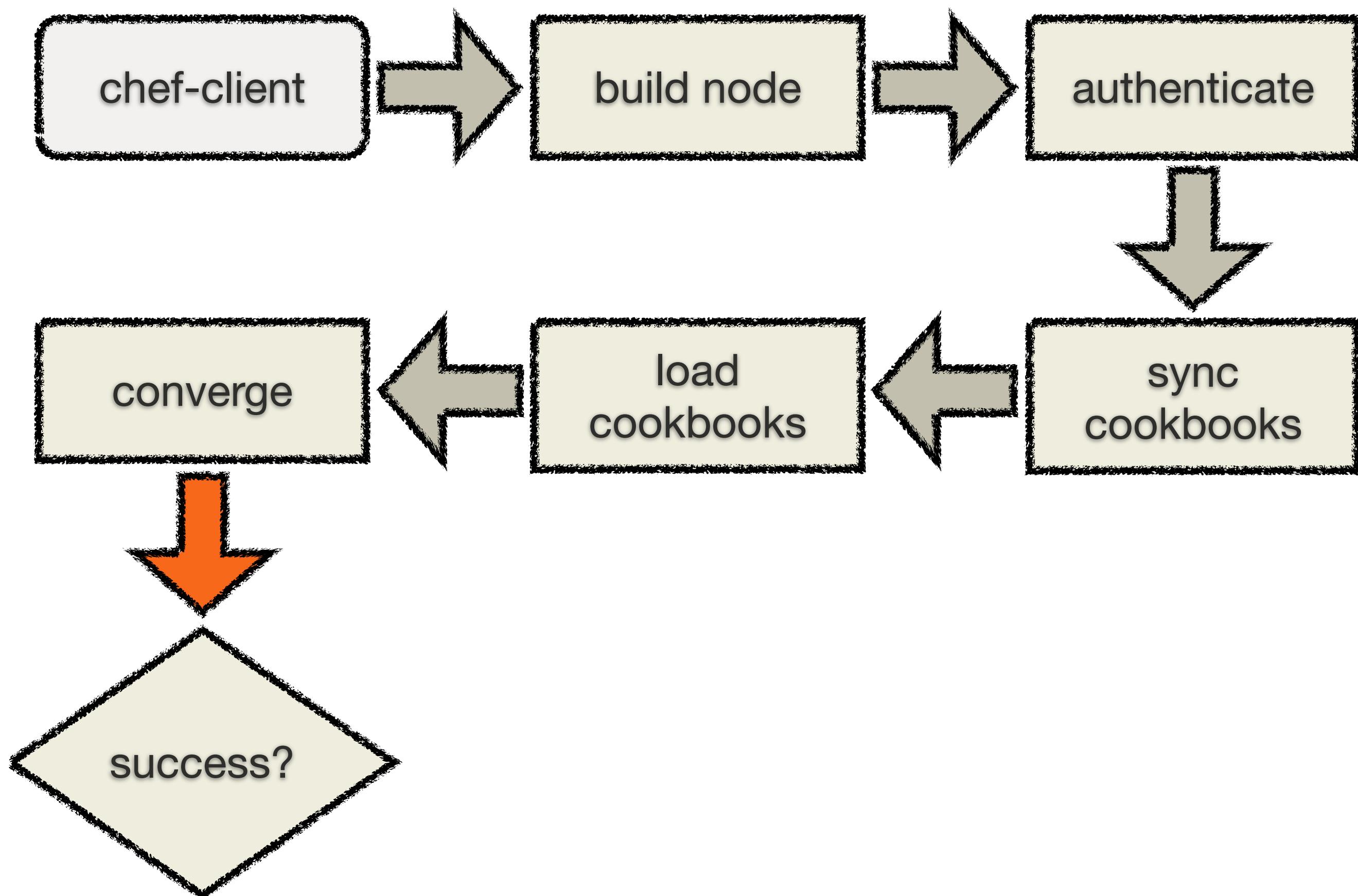
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



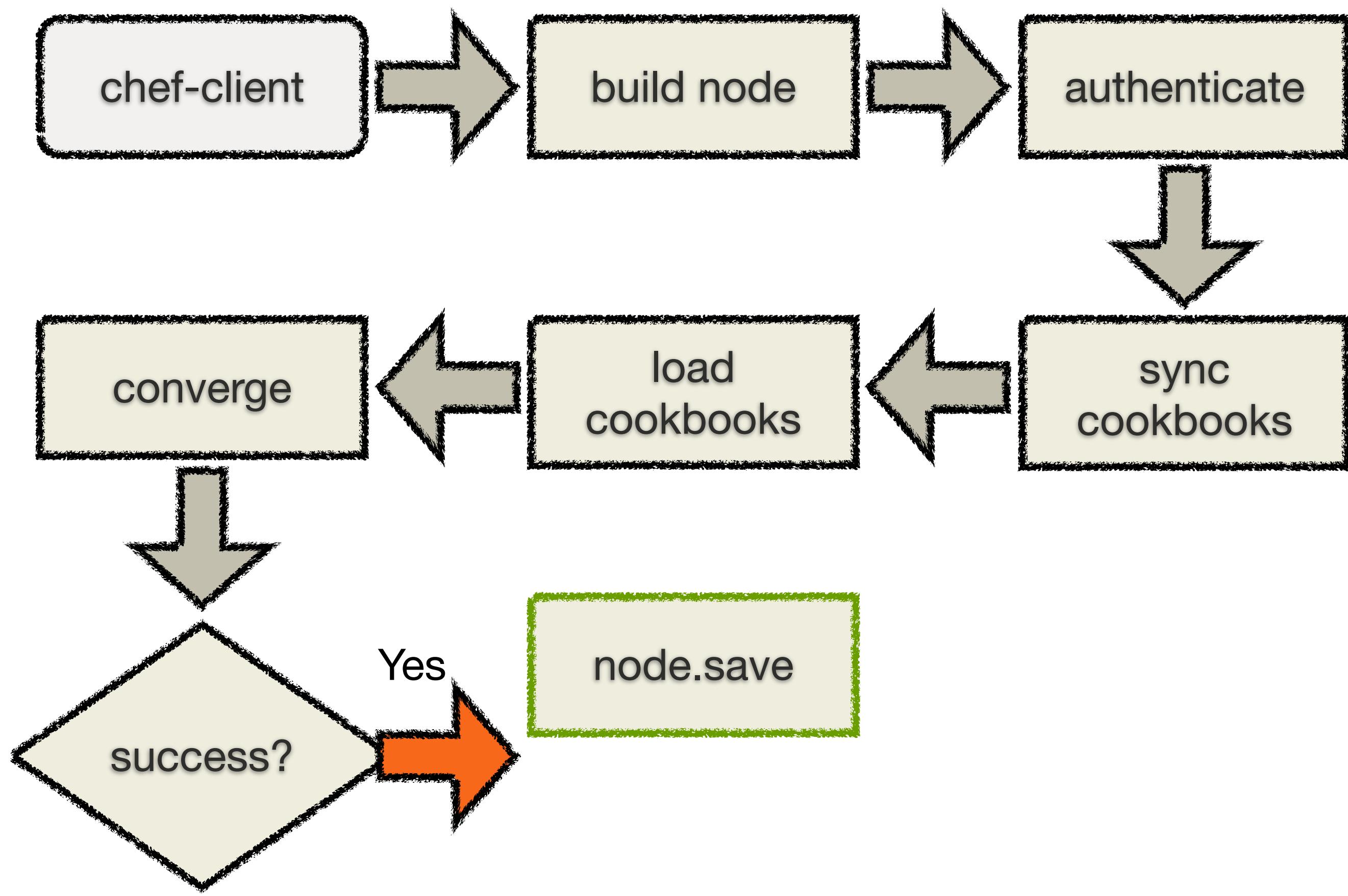
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



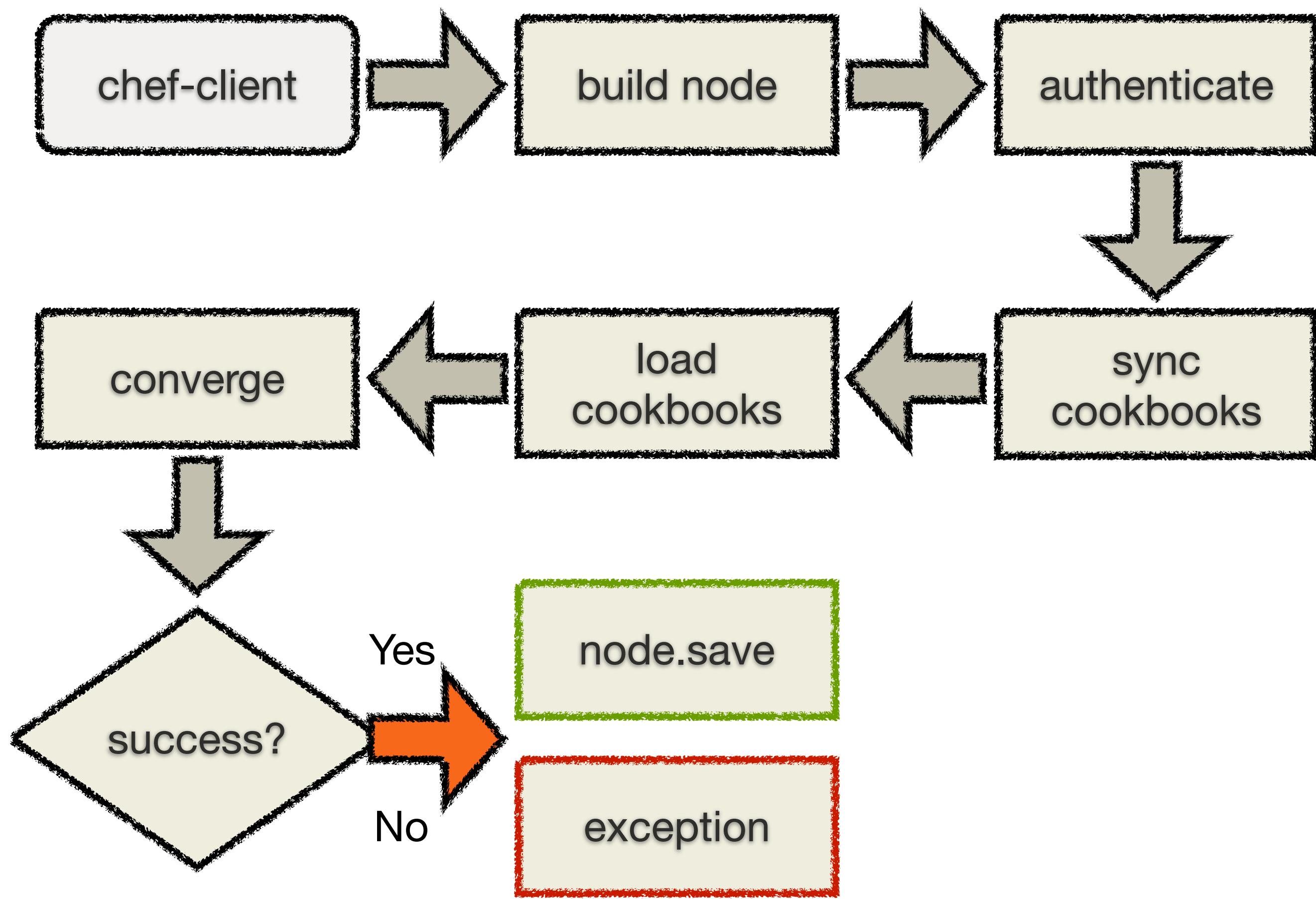
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



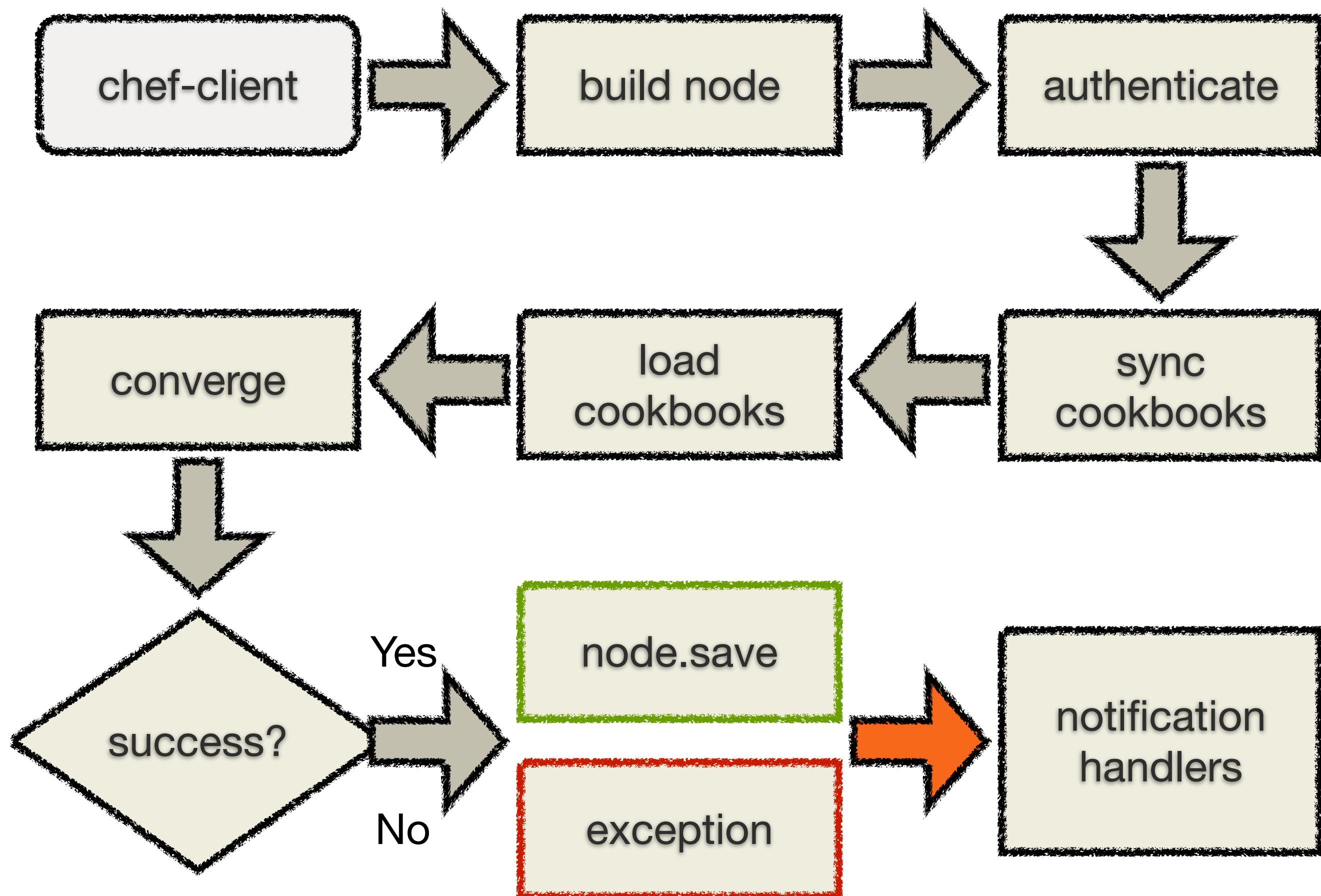
Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.



Monday, 30 June 14

1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node\_name (fqdn) and the node's platform/platform\_version
3. Chef authenticates with the server as the node\_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks required for the node's expanded run list are synchronized to the node.
5. All the cookbooks Ruby components are loaded, recipes are included as well.
6. Node is converged, which is when it configures the resources contained in the recipes.
7. Was the convergence successful? If so, the node is saved.
8. If not the run ends with an exception and a message is printed about the error, and the stack trace file path is included in this.
9. Any notification handlers are run, which can do different things depending on whether the run was successful or not, such as send email or connect to an external service like IRC or Campfire.

# Private Keys

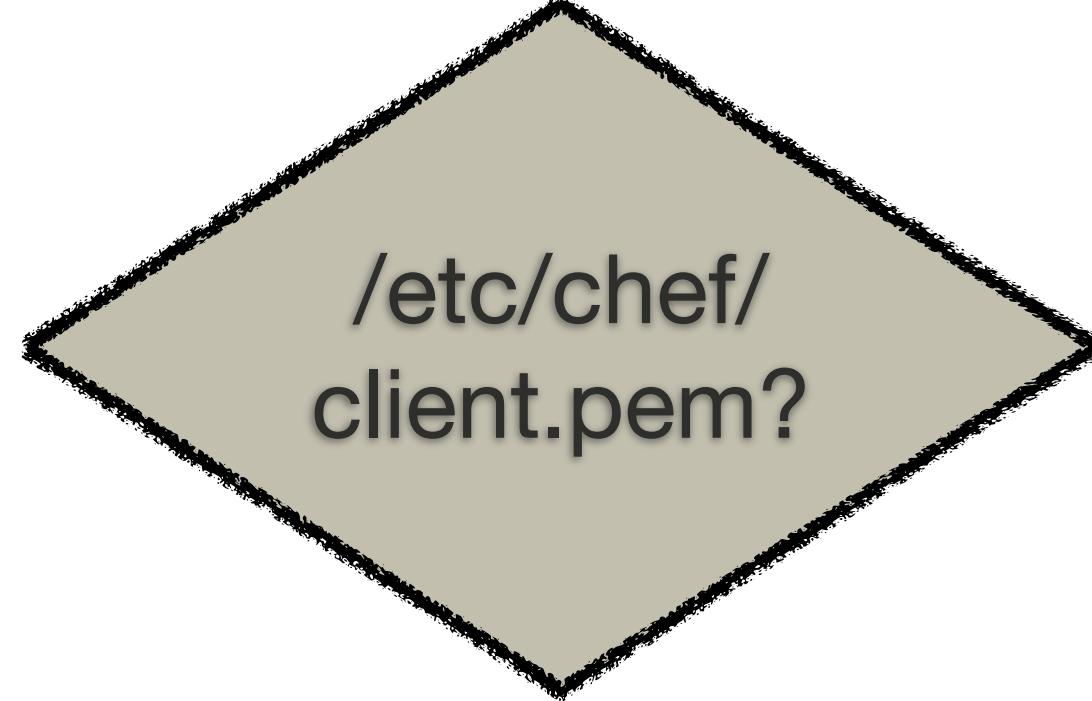
- Chef Server requires keys to authenticate.
  - client.pem - private key for API client
  - validation.pem - private key for ORGNAME-validator
- Next, let's see how those are used...

Monday, 30 June 14

You can "cat" these if you want :).

There are two .pem private key files on the system. The client.pem was created during the authentication cycle we just discussed. It is the private key for \*this\* node. The public key is stored on the Chef server.

The validation.pem is the private key for the validation client, also called the organization key. The validation.pem file can be deleted.



/etc/chef/  
client.pem?

Monday, 30 June 14

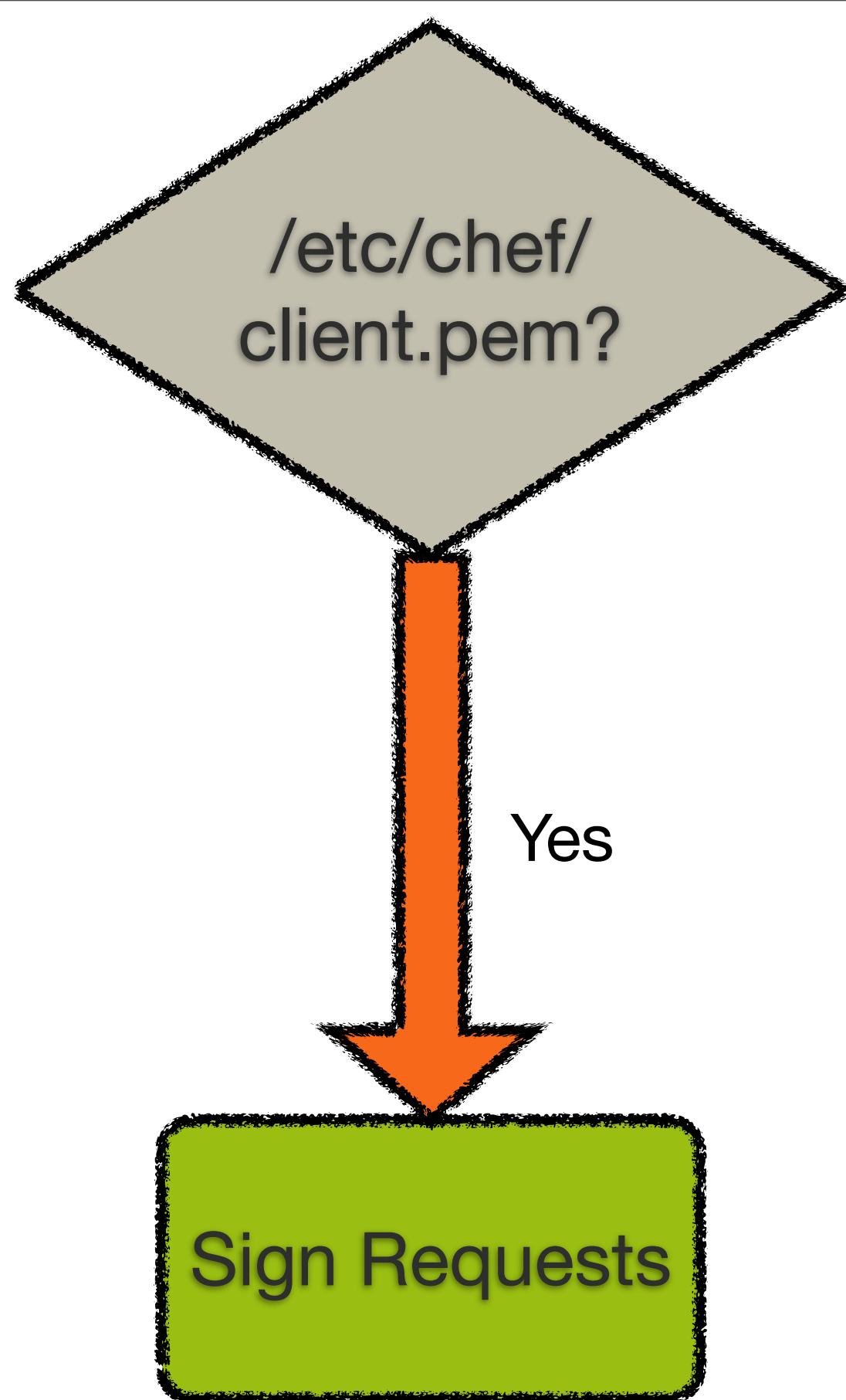
Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for /etc/chef/client.pem.

1. If the client.pem exists, chef will use it to sign requests with the chef server API.
2. If the client.pem does not exist, chef looks for /etc/chef/validation.pem, the validation\_key.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a 401 and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this node\_name and writes this to /etc/chef/client.pem
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate



Monday, 30 June 14

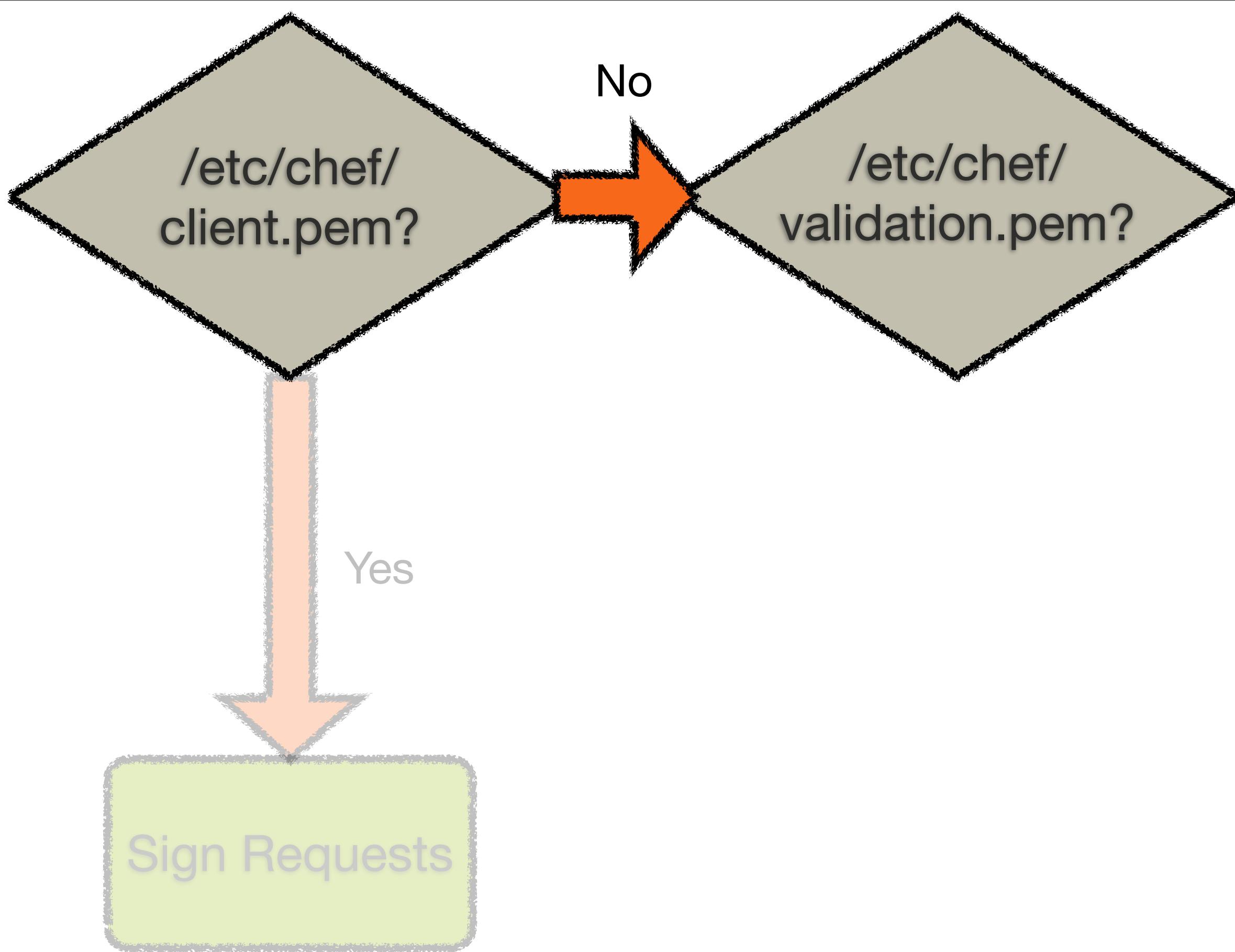
Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for /etc/chef/client.pem.

1. If the client.pem exists, chef will use it to sign requests with the chef server API.
2. If the client.pem does not exist, chef looks for /etc/chef/validation.pem, the validation\_key.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a 401 and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this node\_name and writes this to /etc/chef/client.pem
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate



Monday, 30 June 14

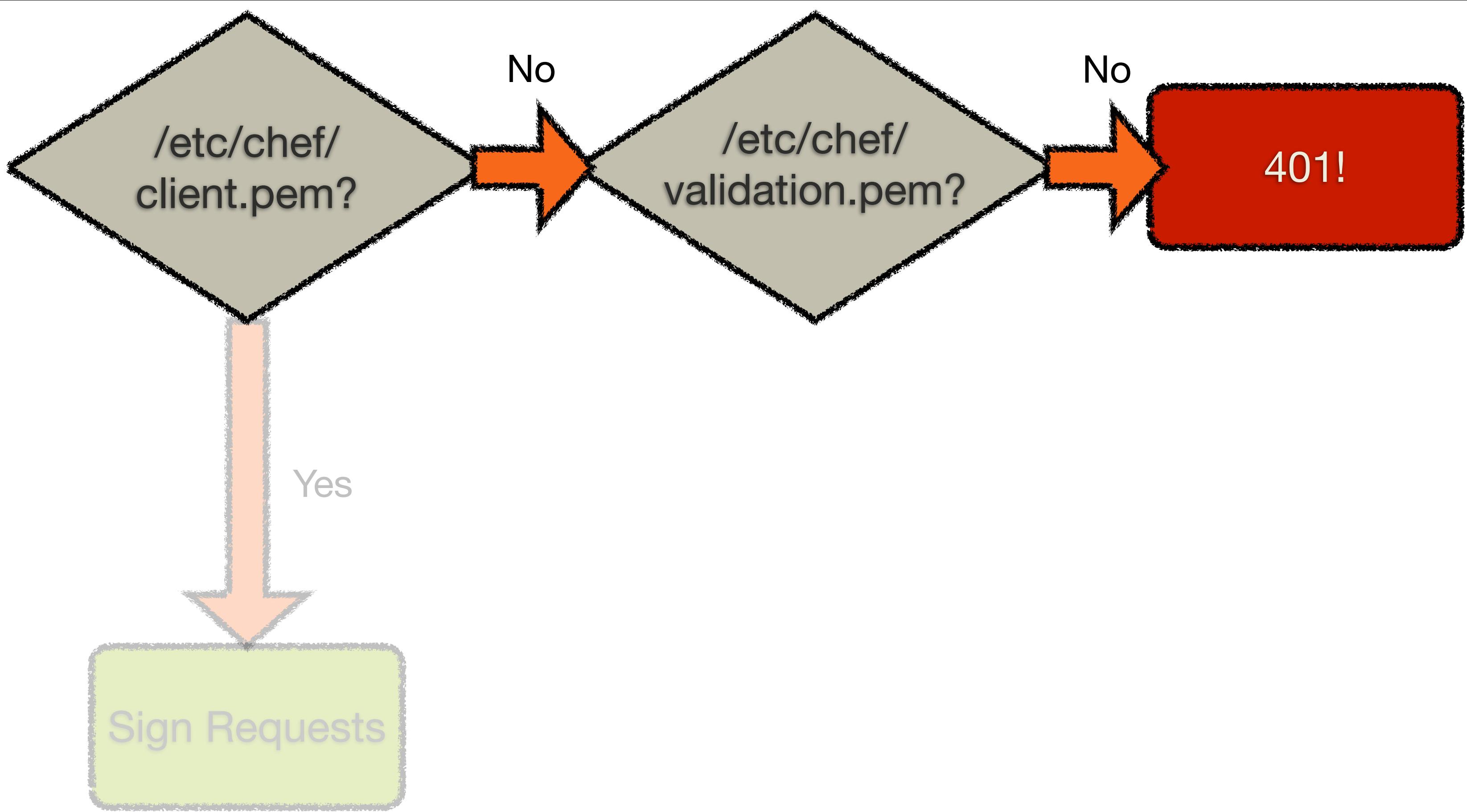
Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for /etc/chef/client.pem.

1. If the client.pem exists, chef will use it to sign requests with the chef server API.
2. If the client.pem does not exist, chef looks for /etc/chef/validation.pem, the validation\_key.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a 401 and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this node\_name and writes this to /etc/chef/client.pem
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate



Monday, 30 June 14

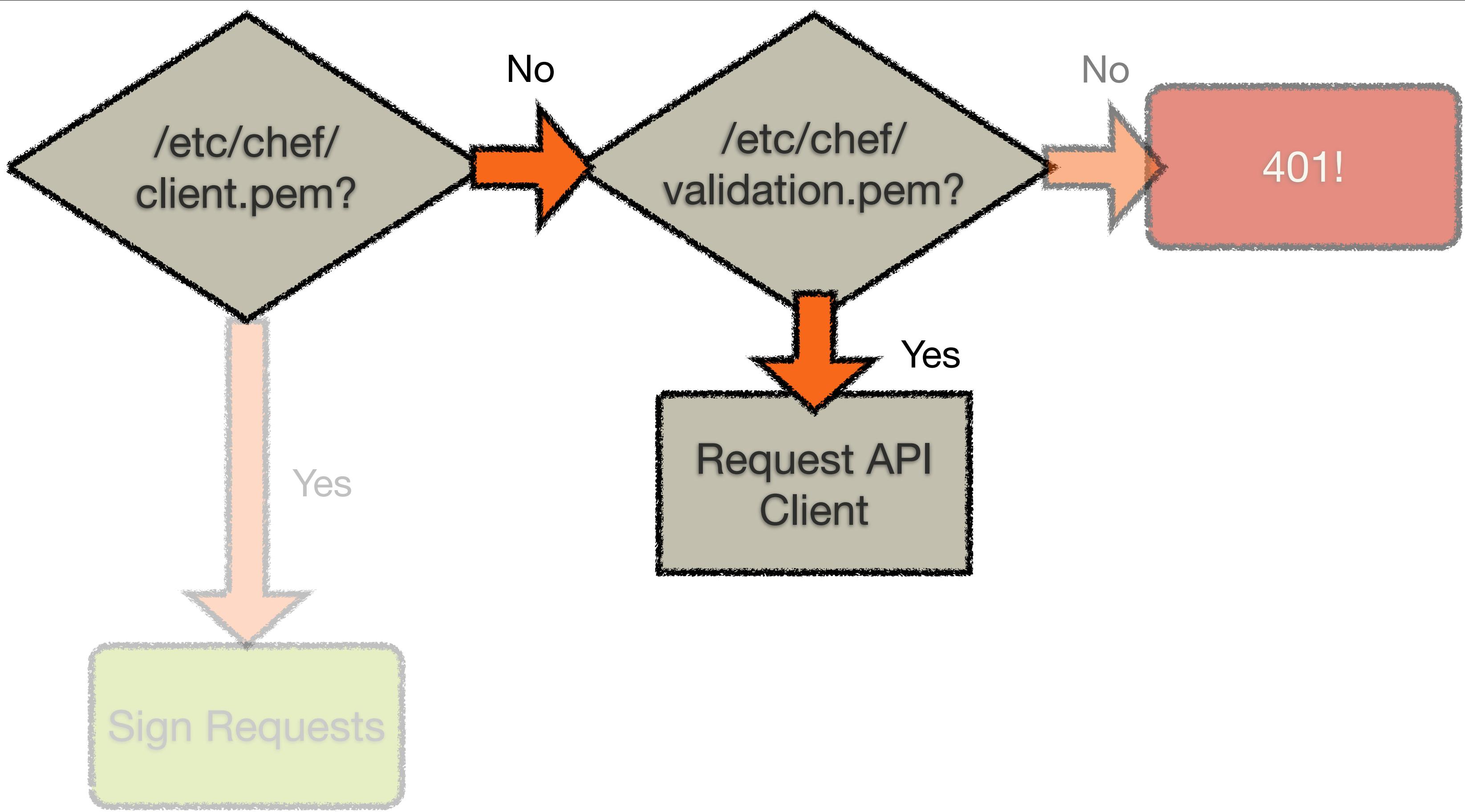
Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for /etc/chef/client.pem.

1. If the client.pem exists, chef will use it to sign requests with the chef server API.
2. If the client.pem does not exist, chef looks for /etc/chef/validation.pem, the validation\_key.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a 401 and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this node\_name and writes this to /etc/chef/client.pem
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate



Monday, 30 June 14

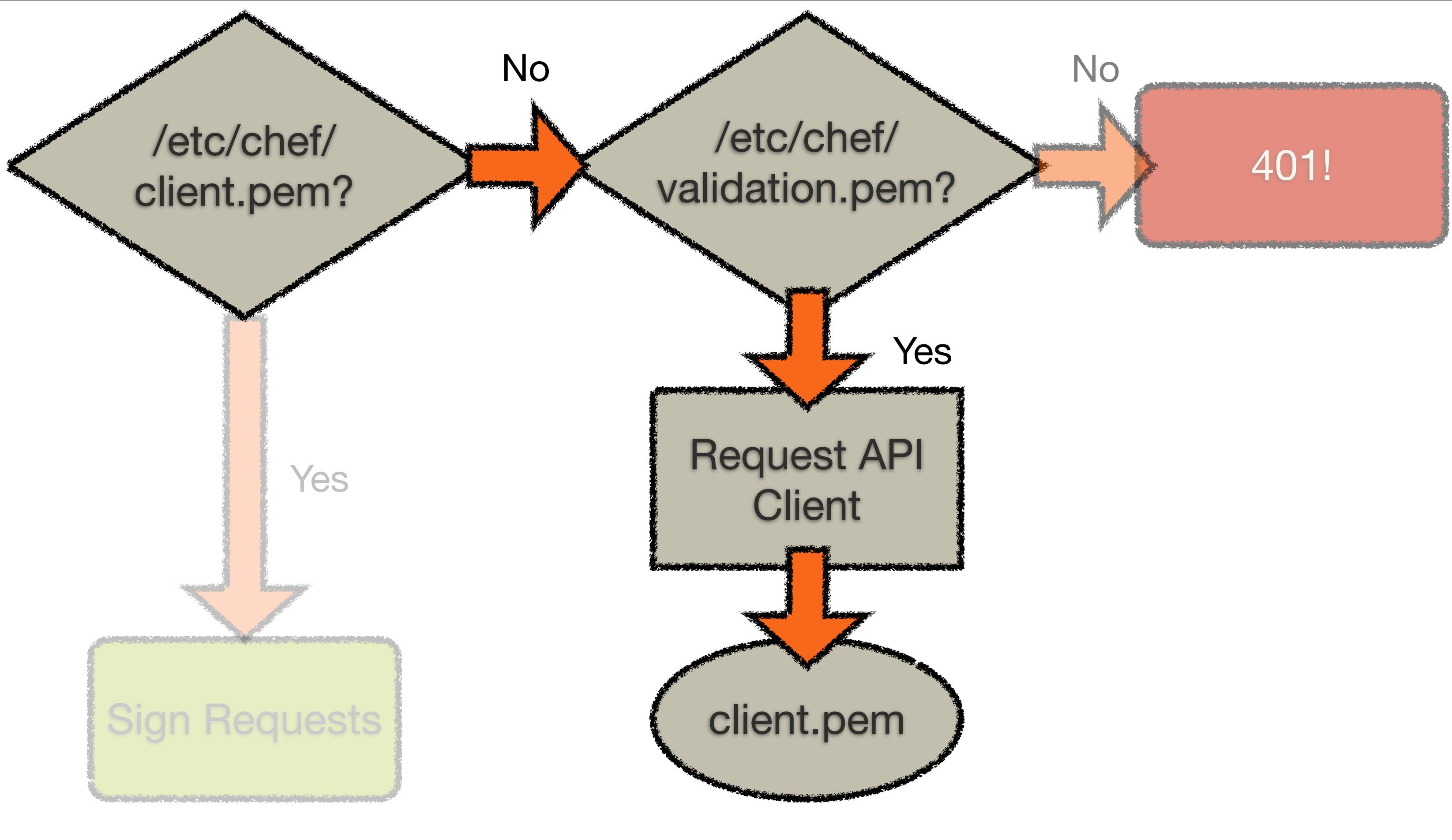
Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for `/etc/chef/client.pem`.

1. If the `client.pem` exists, chef will use it to sign requests with the chef server API.
2. If the `client.pem` does not exist, chef looks for `/etc/chef/validation.pem`, the `validation_key`.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a `401` and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this `node_name` and writes this to `/etc/chef/client.pem`
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate



Monday, 30 June 14

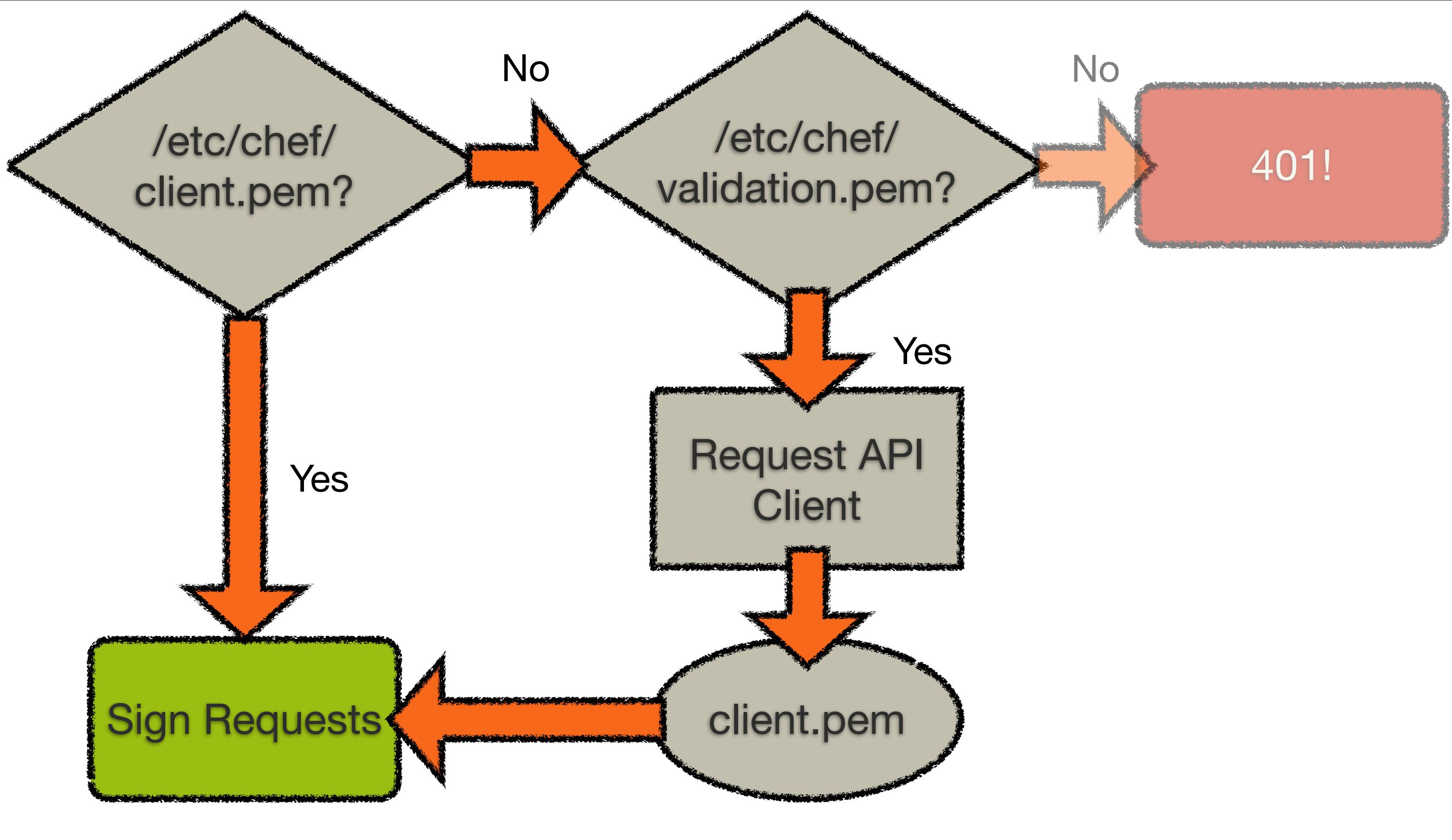
Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for `/etc/chef/client.pem`.

1. If the `client.pem` exists, chef will use it to sign requests with the chef server API.
2. If the `client.pem` does not exist, chef looks for `/etc/chef/validation.pem`, the validation\_key.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a 401 and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this `node_name` and writes this to `/etc/chef/client.pem`
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate



Monday, 30 June 14

Take a minute to talk about how chef does authentication:

- \* We digitally sign each request with a private key
- \* We verify that signature
- \* We protect against replay attacks as well
- \* There is no central store for Private Keys – meaning you have to keep them safe. It also means no attacker can compromise your security credentials centrally.

Lets take a detailed look at the authentication cycle. When Chef client starts, it first looks for `/etc/chef/client.pem`.

1. If the `client.pem` exists, chef will use it to sign requests with the chef server API.
2. If the `client.pem` does not exist, chef looks for `/etc/chef/validation.pem`, the validation\_key.
3. If the validation key does not exist, then chef exits, or if the key does not match for the validation API client, chef reports a `401` and exits.
4. If the validation key exists, and it matches for the validation API client, chef requests a new API client for this `node_name` and writes this to `/etc/chef/client.pem`
5. Chef will use the newly created client to sign requests with the chef server API for all further requests.

If key lost: must delete node & regenerate

# Review Questions

- What are the steps in a Chef Client run?
- How does a new machine get a private key with which to authenticate requests?
- If you have the right credentials in place, why else might you not be able to authenticate?

Monday, 30 June 14

- )
- )
- ) Chef API requests are time sensitive. Check your NTP drift.

# Introducing the Node object

Attributes & Search

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Explain what the Node object represents in Chef
  - List the Nodes in an organization
  - Show details about a Node
  - Describe what Node Attributes are
  - Retrieve a node attribute directly, and via search

# What is the Node object

- A node is any physical, virtual, or cloud machines that is configured to be maintained by a Chef
- When you are writing Recipes, the Node object is always available to you.

Monday, 30 June 14

The node object is the global context in which recipes, attribute files, etc are operating in

# Exercise: List nodes

```
$ knife node list
```

```
node1
```

# Exercise: List clients

```
$ knife client list
```

```
ORGNAME-validator  
node1
```

Monday, 30 June 14

Reinforce that the node has both an object for itself as well as the API client.

# Each node must have a unique name

- Every node must have a unique name within an organization
- Chef defaults to the *Fully Qualified Domain Name* of the server, i.e. in the format `server.domain.com`
- We overrode it to "node1" to make typing easier

Monday, 30 June 14

Simply using the likes of 'node1' is not recommended in real life, but should be something more descriptive and meaningful

Note that many of the students, if they are using a pre-made virtual machine, have the same hostname – this is one of the reasons we have them using separate organizations

# Exercise: Show node details

```
$ knife node show node1
```

```
Node Name:      node1
Environment:    _default
FQDN:          ip-10-154-155-107.ec2.internal
IP:            54.242.35.165
Run List:
Roles:
Recipes:
Platform:      ubuntu 12.04
Tags:
```

# What is the Node object

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

# Exercise: Run Ohai on node

```
opscode@node1:~$ sudo ohai | less
```

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

    },
    "python": {
      "version": "2.7.3",
      "builddate": "Apr 10 2013, 06:20:15"
    },
    "perl": {
      "version": "5.14.2",
      "archname": "x86_64-linux-gnu-thread-multi"
    }
  },
  "kernel": {
```

Monday, 30 June 14

Hammer home that ohai collects a ton of data.

We're using 'less' here as it allows backward movement in the file as well as forward movement, but does not have to read the entire input file before starting, so it starts up faster than text editors like vi with large input files

We are going to get all the way through to the many, many ways you set attributes on a node attribute, but we do it throughout the course – steer people away from a long discussion of precedence or merge order.

They can also run it on their workstations just to see how it differs.

# Exercise: Show all the node attributes

```
$ knife node show node1 -l
```

```
Node Name:    node1
Environment:  _default
FQDN:        ip-10-154-155-107.ec2.internal
IP:          54.242.35.165
Run List:
Roles:
Recipes:
Platform:    ubuntu 12.04
Tags:
Attributes:
tags:

Default Attributes:

Override Attributes:

Automatic Attributes (Ohai Data):
block_device:
  loop0:
    removable: 0
    size:      0
```

# Exercise: Show the raw node object

```
$ knife node show node1 -Fj
```

```
{  
  "name": "node1",  
  "chef_environment": "_default",  
  "run_list": [],  
  "normal": {"tags": []}  
}
```

Monday, 30 June 14

Also yaml formatter too with -Fy if people want to try that.

Dont need to worry about Tags for now. More information on Tags here [http://docs.opscode.com/essentials\\_cookbook\\_recipes.html#use-tags](http://docs.opscode.com/essentials_cookbook_recipes.html#use-tags)

# Exercise: Show only the fqdn attribute

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

```
node1:  
  fqdn: ip-10-154-155-107.ec2.internal
```

# Exercise: Use search to find the same data

```
$ knife search node "*:*" -a fqdn
```

```
1 items found
```

```
node1:
```

```
fqdn: ip-10-154-155-107.ec2.internal
```

# Review Questions

- What is the Node object?
- What is a Node Attribute?
- How do you display all the attributes of a Node?
- Can you search for the **cpu** attribute of your node?

# Chef Resources and Recipes

Writing an Apache cookbook

v1.2.3



Monday, 30 June 14

# Writing an Apache cookbook

Packages, Cookbook Files, and Services

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Describe what a **cookbook** is
  - Create a new cookbook
  - Explain what a **recipe** is
  - Describe how to use the **package**, **service**, and **cookbook\_file** resources
  - **Upload a cookbook** to the Chef Server
  - Explain what a **run list** is, and how to set it for a node via **knife**
  - Read the output of a chef-client run

# What is a cookbook?

- A cookbook is like a "package" for Chef recipes.
- It contains all the recipes, files, templates, libraries, etc. required to configure a portion of your infrastructure
- Typically they map 1:1 to a piece of software or functionality.

Monday, 30 June 14

Example: "apache", "mysql", "mongodb", "tomcat", "jboss" – all of these things are software style cookbooks, and they would contain all the different ways you work with each. (A "mysql client", "mysql server", "mysql slave", etc.)

Functional cookbooks might be something like "users", "security", etc.

We are going to write a lot of cookbooks – so you're going to get comfortable with them.

# The Problem and the Success Criteria

---

- **The Problem:** We need a web server configured to serve up our home page.
- **Success Criteria:** We can see the homepage in a web browser.

# Required steps

- Install Apache
- Start the service, and make sure it will start when the machine boots
- Write out the home page

Monday, 30 June 14

we ought not to recommend the community Apache cookbook blindly, we should simply mention that it exists and that there are other ideas out there in the community for Apache cookbooks.

# Exercise: Create a new Cookbook

```
$ knife cookbook create apache
```

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

# Exercise: Explore the cookbook

```
$ ls -la cookbooks/apache
```

```
total 24
drwxr-xr-x 13 opscode opscode 442 Mar 24 21:25 .
drwxr-xr-x  5 opscode opscode 170 Mar 24 21:25 ..
-rw-r--r--  1 opscode opscode 412 Mar 24 21:25 CHANGELOG.md
-rw-r--r--  1 opscode opscode 1447 Mar 24 21:25 README.md
drwxr-xr-x  2 opscode opscode  68 Mar 24 21:25 attributes
drwxr-xr-x  2 opscode opscode  68 Mar 24 21:25 definitions
drwxr-xr-x  3 opscode opscode 102 Mar 24 21:25 files
drwxr-xr-x  2 opscode opscode  68 Mar 24 21:25 libraries
-rw-r--r--  1 opscode opscode 276 Mar 24 21:25 metadata.rb
drwxr-xr-x  2 opscode opscode  68 Mar 24 21:25 providers
drwxr-xr-x  3 opscode opscode 102 Mar 24 21:25 recipes
drwxr-xr-x  2 opscode opscode  68 Mar 24 21:25 resources
drwxr-xr-x  3 opscode opscode 102 Mar 24 21:25 templates
```

# Exercise: Open the default recipe in your editor



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

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
#
```

# Recipe Naming

- The "default.rb" recipe for a given cookbook is referred to by the name of the cookbook (*apache*)
- If we added another recipe to this cookbook named "mod\_ssl.rb", we would refer to it as *apache::mod\_ssl*

# Exercise: Add package resource to install Apache



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

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
#
```

```
package "apache2" do  
  action :install  
end
```

**SAVE FILE!**

Monday, 30 June 14

Three concepts while they are typing:

1: Strings in ruby are enclosed in double quotes. (Can use singles)

2: :install is a Symbol – symbols in ruby are strings that live in only one place in memory. (No matter how many times you type :install, it refers to the same place.) In our material, anywhere we use a symbol, it is required – you can't replace it with a string.

3: Idiomatic ruby uses two SPACES for indentation, not four, and no tabs.

# Chef Resources

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

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

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

Monday, 30 June 14

Let's take a moment to discuss resources. This is the structure of a Chef resource.

Point out that resource parameters live between the "DO..END".

# Chef Resources

- Have a **type**

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

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

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

# Chef Resources

- Have a **type**
- Have a **name**

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

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

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

# Chef Resources

- Have a **type**
- Have a **name**
- Have **parameters**

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

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

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

# Chef Resources

- Have a **type**
- Have a **name**
- Have **parameters**
- Take **action** to put the resource into the desired state

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

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

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

# Chef Resources

- Have a **type**
- Have a **name**
- Have **parameters**
- Take **action** to put the resource into the desired state
- Can send **notifications** to other resources

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

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

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

# So the resource we just wrote...

```
package "apache2" do
  action :install
end
```

# So the resource we just wrote...

- Is a **package** resource

```
package "apache2" do
  action :install
end
```

# So the resource we just wrote...

- Is a **package** resource
- Whose **name** is *apache2*

```
package "apache2" do
  action :install
end
```

# So the resource we just wrote...

- Is a **package** resource
- Whose **name** is *apache2*
- With an install **action**

```
package "apache2" do
  action :install
end
```

# Notice we didn't say how to install the package

- Resources are **declarative** - that means we say *what* we want to have happen, rather than *how*
- Chef uses the **platform** the node is running to determine the correct **provider** for a resource

Monday, 30 June 14

Declarative resources **inspect** the system state, and do whatever they need to do to make the system conform with the declared state.

Chef automatically knows that we should use "apt" on ubuntu, "yum" on red hat by looking up the correct PROVIDER based on the platform.

## Exercise: Add a service resource to ensure service is started & enabled at boot



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

```
...
# All rights reserved - Do Not Redistribute
#
package "apache2" do
  action :install
end

service "apache2" do
  action [ :enable, :start ]
end
```

**SAVE FILE!**

# So the resource we just wrote...

```
service "apache2" do
  action [ :enable, :start ]
end
```

Monday, 30 June 14

You can have more than one action, and they are evaluated in order. So we first will make sure this service is running, and that it is enabled at boot

# So the resource we just wrote...

- Is a **service** resource

```
service "apache2" do
  action [ :enable, :start ]
end
```

# So the resource we just wrote...

- Is a **service** resource
- Whose **name** is *apache2*

```
service "apache2" do
  action [ :enable, :start ]
end
```

# So the resource we just wrote...

- Is a **service** resource
- Whose **name** is *apache2*
- With two **actions**: **start** and **enable**

```
service "apache2" do
  action [ :enable, :start ]
end
```

# Order Matters

- Resources are executed in order

1st

2nd

3rd

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

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

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

Monday, 30 June 14

Order matters because the operating system was designed to be managed by hand

Think about adding a user, where the user ids are dynamically generated – if they aren't added in the same order, you can't guarantee that the system will be the same.

# Exercise: Add a cookbook\_file resource to copy the homepage in place

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

```
...
service "apache2" do
  action [ :enable, :start ]
end

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

**SAVE FILE!**

Monday, 30 June 14

We are leaving the action off intentionally, in case anyone asks.

# So the resource we just wrote...

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

Monday, 30 June 14

Hammer home this stuff, because later in the deck, we are going to force them to write resources without the syntax on the right at all.

# So the resource we just wrote...

- Is a **cookbook\_file** resource

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

Monday, 30 June 14

Hammer home this stuff, because later in the deck, we are going to force them to write resources without the syntax on the right at all.

# So the resource we just wrote...

- Is a **cookbook\_file** resource
- Whose **name** is */var/www/index.html*

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

Monday, 30 June 14

Hammer home this stuff, because later in the deck, we are going to force them to write resources without the syntax on the right at all.

# So the resource we just wrote...

- Is a **cookbook\_file** resource
- Whose **name** is */var/www/index.html*
- With two **parameters**:
  - **source** of *index.html*
  - **mode** of "0644"

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

Monday, 30 June 14

Hammer home this stuff, because later in the deck, we are going to force them to write resources without the syntax on the right at all.

# Best Practice: Omit the action if it is the default

- Has no action!
- If you omit the action in Chef, we default to the most common positive action. In this case, it is the :create action.

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

# Full contents of the apache recipe

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
  
#  
  
package "apache2" do  
  action :install  
end  
  
service "apache2" do  
  action [ :enable, :start ]  
end  
  
cookbook_file "/var/www/index.html" do  
  source "index.html"  
  mode "0644"  
end
```

Monday, 30 June 14

Note: do/end, no squiggly brackets -- idiomatic leave off unnecessary bits

# Question!

- Using what we have learned so far, can we make the recipe shorter?

Monday, 30 June 14

Answer: omit the action on the package resource.

Don't have them do it – just point it out. :)

# Best Practice: Omit the default action

- If the only action you need from a resource is the default action - omit it from the recipe

Monday, 30 June 14

In Ruby, it is common to not type characters you don't need.

In Chef, we carry that forward – since the rule is that we default to the most common positive action, we don't type it if we don't have to.

The rule of least surprise!

# Exercise: Add index.html to cookbook's files/default directory



**OPEN IN EDITOR:** cookbooks/apache/files/default/index.html

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

**SAVE FILE!**

Monday, 30 June 14

Reflect on the fact that, yeah, we're not a class about HTML – but we are doing things the hard way, and part of that is getting used to putting files in the right places in cookbooks, and typing them in. If they've never written HTML before, congratulate them on their first web site. :)

# What's with the 'default' subdirectory?

- Chef allows you to select the most appropriate file (or template) within a cookbook according to the platform of the node it is being executed on
  - host node name (e.g. foo.bar.com)
  - platform-version (e.g. redhat-6.2.1)
  - platform-version\_components (e.g. redhat-6.2, redhat-6)
  - platform (e.g. redhat)
  - default
- 99% of the time, you will just use **default**

# Exercise: Upload the cookbook

```
$ knife cookbook upload apache
```

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

Monday, 30 June 14

This checks for syntax errors, bundles the cookbook up, and uploads it to the chef server for distribution to the chef clients. If they have syntax errors in the recipe, it will get caught here.

If you get a ‘ruby’ error on Windows, you need to make the Chef repo live in a path without spaces in it. (Seriously.)

# Exercise: Add apache recipe to test node's run list

```
$ knife node run_list add node1 "recipe[apache]"
```

```
node1:  
  run_list: recipe[apache]
```

# The Run List

- The Run List is the ordered set of recipes and roles that the Chef Client will execute on a node
  - Recipes are specified by "**recipe[*name*]**"
  - Roles are specified by "**role[*name*]**"

Monday, 30 June 14

We talk about Roles sometime in day two, for now , just get the reference out

# Exercise: Run chef-client on your test node

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

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
Converging 15 resources
Recipe: apache::default
* package[apache2] action install
  - install version 2.2.22-1ubuntu1 of package apache2

* service[apache2] action enable
  - enable service service[apache2]

* service[apache2] action start (up to date)
* execute[a2dissite default] action run
  - execute a2dissite default

* template[/etc/apache2/sites-available/clowns] action create
  - create new file /etc/apache2/sites-available/clowns
  - update content in file /etc/apache2/sites-available/clowns from none to acb2ac
    --- /etc/apache2/sites-available/clowns 2013-11-05 16:17:06.444688887 +0000
    +++ /tmp/chef-rendered-template20131105-3310-188e3so 2013-11-05 16:17:06.448688888 +0000
    @@ -1 +1,17 @@
    +
    +<VirtualHost *:80>
    +  ServerAdmin webmaster@localhost
```

Monday, 30 June 14

Typical errors:

- \* The file is not in the 'default' subdirectory
- \* The name of the package or service is wrong
- \* The VM has an out of date apt-cache. run 'apt-get update' if the package apache2 finds a version, but cannot install it.

# Exercise: Verify homepage works

- Open a web browser
- Type in the URL for your test node



Monday, 30 June 14

If it's not working at all – check the firewall rules. You can disable them.

If you can't reach your test node from your workstation, use 'curl <http://localhost>' from the command line on the test node.

# Congratulate yourself!

- You have just written your first Chef cookbook!
- (clap!)

Monday, 30 June 14

Note that they have just used packages, services, and templates: three resources which are the trinity of config mgmt!

# Reading the output of a chef-client run

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
Converging 15 resources
Recipe: apache::default
 * package[apache2] action install
  - install version 2.2.22-1ubuntu1 of package apache2
```

- The run list is shown
- The expanded Run List is the complete list, after nested roles are expanded

Monday, 30 June 14

We tell you the nodes run list, as set on the object itself  
We tell you what the nodes run list \*expands\* to, which comes in to play when roles are nested  
We cover expansion in much more depth later.

# Reading the output of a chef-client run

```
resolving cookbooks for run list: ["apache"]
[2013-06-25T04:20:24+00:00] INFO: Loading cookbooks [apache]
Synchronizing Cookbooks:
[2013-06-25T04:20:24+00:00] INFO: Storing updated cookbooks/apache/recipes/default.rb in the
cache.
[2013-06-25T04:20:24+00:00] INFO: Storing updated cookbooks/apache/CHANGELOG.md in the
cache.
[2013-06-25T04:20:25+00:00] INFO: Storing updated cookbooks/apache/metadata.rb in the cache.
[2013-06-25T04:20:25+00:00] INFO: Storing updated cookbooks/apache/README.md in the cache.
 - apache
Compiling Cookbooks...
```

- Load the cookbooks in the order specified by the run list
- Download any files that are missing from the server

# Reading the output of a chef-client run

```
Converging 3 resources
Recipe: apache::default
 * package[apache2] action install[2013-06-25T04:20:25+00:00] INFO:
Processing package[apache2] action install (apache::default line 9)
  - install version 2.2.22-1ubuntu1 of package apache2
```

- Check to see if the package apache2 is installed
- It was not, so version 2.2.22-1ubuntu1 of the package was installed (yours may be different)

Monday, 30 June 14

Talk about the Processing lines – this means we are evaluating this resource for the "install" action. You will see these lines for every resource.

# Reading the output of a chef-client run

```
* service[apache2] action enable[2013-06-25T04:20:40+00:00] INFO:  
Processing service[apache2] action enable (apache::default line 13)  
(up to date)  
* service[apache2] action start[2013-06-25T04:20:40+00:00] INFO: Processing  
service[apache2] action start (apache::default line 13)  
(up to date)
```

- Check to see if apache2 is already enabled to run at boot - it is, take no further action
- Check to see if apache2 is already started - it is, take no further action

Monday, 30 June 14

Take this moment to talk about the differences between operating systems. Ubuntu policy says applications should be installed, configured, and started on package installation. Red Hat says they should be installed and configured, but not started.

Putting "start" into the recipe is an Ubuntu systems quirk.

This is **why you bring your systems knowledge, and we bring the framework = unicorns!**

# Idempotence

- Actions on resources in Chef are designed to be **idempotent**
- This means they can be applied multiple times but the end result is still the same - like multiplying by 1 in mathematics!
- Chef is a "desired state configuration" system - if a resource is already configured, no action is taken. This is called *convergence*

Monday, 30 June 14

Idempotence, in math, means an operation can be applied multiple times without changing the result.  $1 \times 1 = 1$  is an idempotent operation of multiplication.

Chef is a "desired state configuration" system and that if the current state == desired state, it won't do anything.

# Reading the output of a chef-client run

```
* cookbook_file[/var/www/index.html] action create[2013-06-25T04:20:40+00:00] INFO: Processing
cookbook_file[/var/www/index.html] action create (apache::default line 17)
[2013-06-25T04:20:41+00:00] INFO: cookbook_file[/var/www/index.html] backed up to /var/chef/backup/var/
www/index.html.chef-20130625042041
[2013-06-25T04:20:41+00:00] INFO: cookbook_file[/var/www/index.html] mode changed to 644
[2013-06-25T04:20:41+00:00] INFO: cookbook_file[/var/www/index.html] created file /var/www/index.html

- create a new cookbook_file /var/www/index.html
  --- /var/www/index.html 2013-06-25 04:20:37.036043860 +0000
  +++ /var/chef/cache/cookbooks/apache/files/default/index.html 2013-06-25 04:20:40.904043861
+0000
@@ -1,4 +1,5 @@
-<html><body><h1>It works!</h1>
-<p>This is the default web page for this server.</p>
-<p>The web server software is running but no content has been added, yet.</p>
-</body></html>
...

```

- Check for an index.html file
- There is already one in place, backup the file
- Set permissions on the file
- A diff of the written file is shown with the modified lines called out

Monday, 30 June 14

Make note that we back up files we change

If a student asks about what happens if they have local changes, have them modify index.html, and run it again – and we put it back

If nobody asks, bring it up, and have someone do it. Once you manage something with Chef, it is managed – period.

If anyone fights with you, the story is easy – if you want to be able to make any kind of sane statement about whether your servers are correct, you can't have managed and un-managed changes. You might fix a single machine by hand, but the final fix goes back into Chef.

# Reading the output of a chef-client run

```
[2013-06-25T04:20:41+00:00] INFO: Chef Run complete in 17.432572377 seconds
[2013-06-25T04:20:41+00:00] INFO: Running report handlers
[2013-06-25T04:20:41+00:00] INFO: Report handlers complete
Chef Client finished, 2 resources updated
```

- Time to complete the Chef run is displayed
- Report and exception handlers are now run

Monday, 30 June 14

We don't really get to reports and exception handlers – the gist is, you use them to send statistics about the run to all sorts of places – email, splunk, etc.

# Exercise: Re-run Chef Client

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

```
Converging 3 resources
Recipe: apache::default
  * package[apache2] action install[2013-06-25T04:49:38+00:00] INFO: Processing package[apache2] action install
(apache::default line 9)
  (up to date)
  * service[apache2] action enable[2013-06-25T04:49:39+00:00] INFO: Processing service[apache2] action enable
(apache::default line 13)
  (up to date)
  * service[apache2] action start[2013-06-25T04:49:39+00:00] INFO: Processing service[apache2] action start
(apache::default line 13)
  (up to date)
  * cookbook_file[/var/www/index.html] action create[2013-06-25T04:49:39+00:00] INFO: Processing cookbook_file[/var/www/
index.html] action create (apache::default line 17)
  (up to date)
[2013-06-25T04:49:39+00:00] INFO: Chef Run complete in 0.670331523 seconds
[2013-06-25T04:49:39+00:00] INFO: Removing cookbooks/apache/files/default/index.html from the cache; it is no longer
needed by chef-client.
[2013-06-25T04:49:39+00:00] INFO: Running report handlers
[2013-06-25T04:49:39+00:00] INFO: Report handlers complete
Chef Client finished, 0 resources updated
```

Monday, 30 June 14

Note that it didn't do anything – idempotency at its best.

# Review Questions

- What is a cookbook?
- How do you create a new cookbook?
- What is a recipe?
- What is a resource?
- How do you upload a cookbook to the Chef Server?
- What is a run list?
- What do the "Processing" lines in the chef-client output mean?

# Attributes, Templates, and Cookbook Dependencies

Writing an MOTD Cookbook

v1.2.3



Monday, 30 June 14

# Writing an MOTD Cookbook

Cookbook Attributes, Cookbook Dependencies, Attribute Precedence, and ERB Templates

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Describe Cookbook Attribute files
  - Use ERB Templates in Chef
  - Explain Attribute Precedence
  - Describe Cookbook Metadata
  - Specify cookbook dependencies
  - Perform the cookbook creation, upload, and test loop

# The Problem and the Success Criteria

---

- **The Problem:** We need to add a message that appears at login that states:
  - "This server is property of COMPANY"
  - "This server is in-scope for PCI compliance" if the server is, in fact, in scope.
- **Success Criteria:** We see the message when we log in to the test node

# We have a small problem...

- We don't have an attribute for 'Company', nor one that reflects whether it is in or out of scope for PCI Compliance

# Possible Solutions

- Well factored cookbooks only contain the information relevant to their domain
- We know we will likely have other things related to PCI (security settings, for example)
- So the best thing to do is create a PCI cookbook, and add our attribute there

Monday, 30 June 14

Tiresome because we would wind up evaluating each one, and there are probably rules

We could add node attributes for Company name and PCI Compliance, but that will become very tiresome at scale

# Exercise: Create a cookbook named ‘motd’

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

# Exercise: Create a cookbook named ‘motd’

```
$ knife cookbook create motd
```

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

# Exercise: Create a default.rb attribute file



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

```
default[ 'company' ] = "Opscode"
```

**SAVE FILE!**

- Creates a new Node attribute: `node[ 'company' ]`
- Sets the values to the string "Opscode"
- Note: there is no space between 'default' and '[' in attributes files!

Monday, 30 June 14

We introduce new syntax here – the first is our node attribute syntax, which lets you create:

- \* A top level attribute named 'company'
- \* Whose value is the string 'Opscode'

We will introduce the PCI compliance attribute a little later

If there are ruby people in the room, make note of the fact that we just auto-vivified a hash (we didn't have to create the interstitial hash for 'in\_scope' – chef did that for us.)

Talk about truth/false. The bareword 'false' means false, as does 'nil'. True is any other value, including the literal bareword 'true'. Unlike some languages, in ruby, 0 is **not** false.

Reiterate there is no space before the [ in attributes!

# Exercise: Open the default recipe in your editor



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

```
#  
# Cookbook Name:: motd  
# Recipe:: default  
  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
  
#  
# All rights reserved - Do Not Redistribute  
#
```

# What resource should we use?

- We could try and use a `cookbook_file` here, and rely on the file copy rules. Create a file per server, basically.
- Obviously, that's dramatically inefficient.
- Instead, we will render a **template** - a file that is a mixture of the contents we want, and embedded Ruby code

# Exercise: Add template resource for /etc/motd.tail

- Use a **template** resource
- The **name** is "`/etc/motd.tail`"
- The resource has two parameters
  - **source** is "`motd.tail.erb`"
  - **mode** is "`0644`"

Monday, 30 June 14

Talk about the hard way again!

We are going to WAIT for the big reveal. Students will walk out of here confident that they can make their own recipes.

See if they can do it without a syntax hint – if they need help, ask another student, the teacher, or use the apache recipe we wrote earlier as a reference.

(On Debian systems, the system message of the day is rebuilt at each startup, in order to display an accurate information. So `/etc/motd.tail` is the file to edit permanent changes to the message of the day)

# The template[/etc/motd.tail] resource



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

```
#  
# Cookbook Name:: motd  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
template "/etc/motd.tail" do  
  source "motd.tail.erb"  
  mode "0644"  
end
```

**SAVE FILE!**

Monday, 30 June 14

The big reveal! Watch for how many got it right.

Congrats, you just learned some Ruby!

# Exercise: Open motd.tail.erb in your Editor



**OPEN IN EDITOR:** cookbooks/motd/templates/default/motd.tail.erb

```
This server is property of <%= node['company'] %>
<% if node['pci']['in_scope'] -%>
This server is in-scope for PCI compliance
<% end -%>
```

**SAVE FILE!**

- "erb" stands for "Embedded Ruby"

Monday, 30 June 14

This is the "source" file we referenced in our template resource.

It is the same templating language used in Ruby on Rails, and should be familiar to users of PHP, Perl's Mason, etc.

We will add the attribute 'node['pci']['in\_scope']' shortly

# Exercise: Open motd.tail.erb in your Editor



**OPEN IN EDITOR:** cookbooks/motd/templates/default/motd.tail.erb

```
This server is property of <%= node['company'] %>
<% if node['pci']['in_scope'] -%>
This server is in-scope for PCI compliance
<% end -%>
```

- To embed a value within an ERB template:
  - Start with <%=
  - Write your Ruby expression - most commonly a node attribute
  - End with %>

# Exercise: Open motd.tail.erb in your Editor



**OPEN IN EDITOR:** cookbooks/motd/templates/default/motd.tail.erb

```
This server is property of <%= node['company'] %>
<% if node['pci']['in_scope'] -%>
This server is in-scope for PCI compliance
<% end -%>
```

- You can use any Ruby construct in a template
  - Starting with <% will evaluate the expression, but not insert the result
  - Ending with -%> will not insert a line in the resulting file

Monday, 30 June 14

We introduce the "if" conditional here. We will include the next line if 'node['pci']['in\_scope']' is true. Ask the class whether we are going to include it or not. Remind them that in ruby:

\* false or nil is false  
\* anything else is true

# Templates Are Used For Almost All Configuration Files

- Templates are very flexible ways to create your configuration files
- Coupled with Chef's attribute precedence rules, you can create very effective, data-driven cookbooks

Monday, 30 June 14

Once again: the holy trinity – package, service, template

## Best Practice: Recipes contain the pattern, attributes supply the details

- Recipes contain the pattern for how to do something. ("How we deploy tomcat")
- Attributes contain the details. ("What port do we run tomcat on?")

Monday, 30 June 14

We repeat this a lot in this deck. Drive it home.

Best practice = abstraction for portability. If you couple desired attributes into your recipes, disaster will befall you later!

# Exercise: Upload the motd cookbook

```
Uploading motd  
Uploaded 1 cookbook.
```

[ 0 . 1 . 0 ]

Monday, 30 June 14  
Testing for syntax  
Uploading files to the Chef Server

# Exercise: Upload the motd cookbook

```
$ knife cookbook upload motd
```

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

Monday, 30 June 14

Testing for syntax

Uploading files to the Chef Server

# Exercise: Create a cookbook named ‘pci’

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

Monday, 30 June 14

Note: knife cmds no longer automatically appear in the deck presentation from here on out. Encourage students to help each other. The "big reveal" will happen after they've had ample time to figure it out.

# Exercise: Create a cookbook named ‘pci’

```
$ knife cookbook create pci
```

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

Monday, 30 June 14

Note: knife cmds no longer automatically appear in the deck presentation from here on out. Encourage students to help each other. The "big reveal" will happen after they've had ample time to figure it out.

# Exercise: Create a default.rb attribute file

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

```
default['pci']['in_scope'] = false
```

**SAVE FILE!**

- Creates a new Node attribute: node['pci']['in\_scope']
- Sets the value to the Ruby false literal

Monday, 30 June 14

We introduce new syntax here – the first is our node attribute syntax, which lets you create:

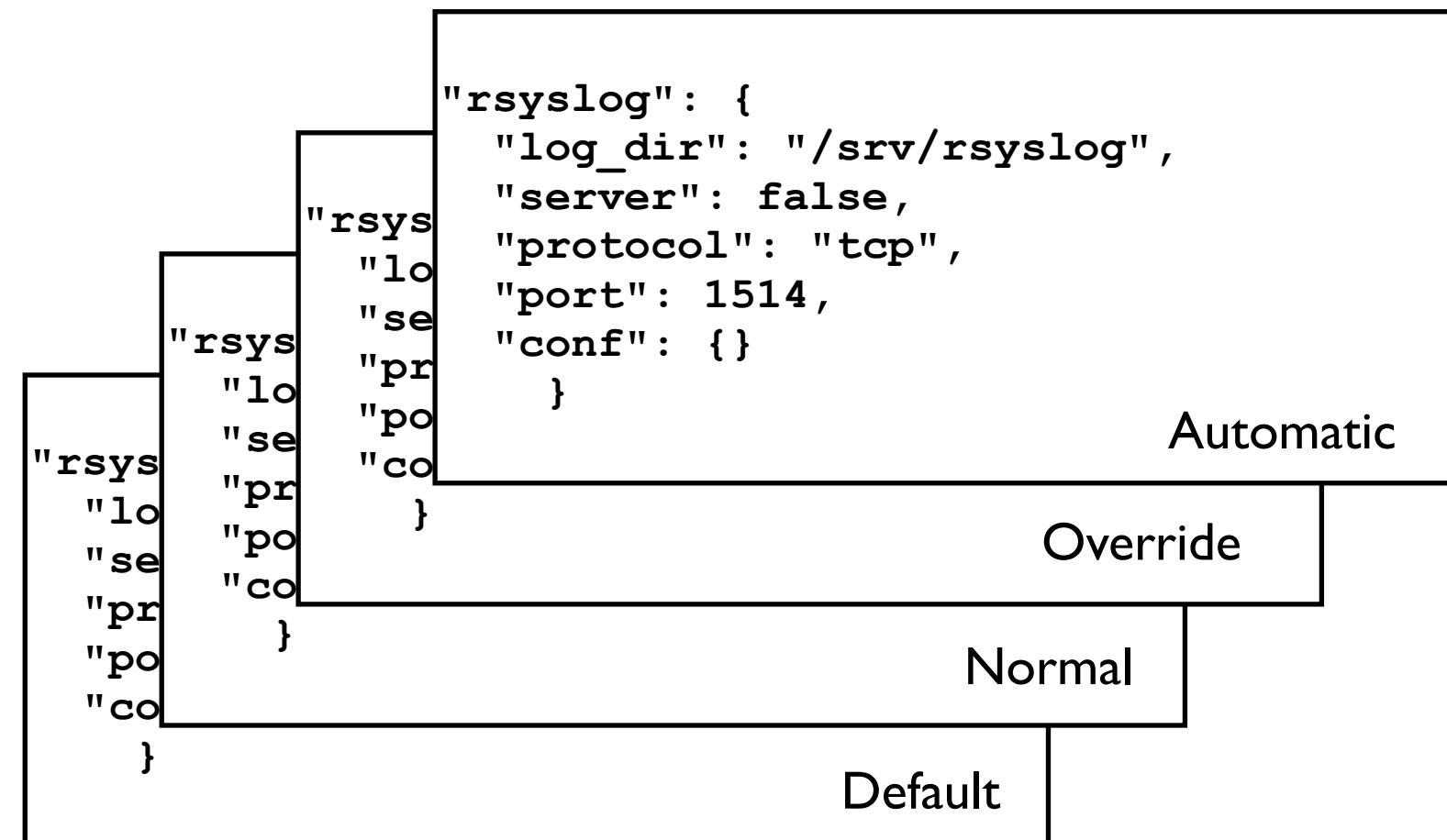
- \* A top level attribute named 'pci'
- \* Whose value is a hash
- \* With a key named 'in\_scope'
- \* Whose value is the ruby literal 'false'

If there are ruby people in the room, make note of the fact that we just auto-vivified a hash (we didn't have to create the interstitial hash for 'in\_scope' – chef did that for us.)

Talk about truth/false. The bareword 'false' means false, as does 'nil'. True is any other value, including the literal bareword 'true'. Unlike some languages, in ruby, 0 is **not** false.

# Node Attributes have four levels of precedence

- **Automatic** attributes are those discovered by Ohai
- **Override** attributes are the strongest way to set an attribute - use sparingly
- **Normal** attributes are those set directly on a Node object
- **Default** attributes are typically set in Cookbooks, Roles and Environments



Monday, 30 June 14

We explain the fullness of precedence throughout the deck. We also hit hard on the fact that you should, in almost all cases, be using default attributes. The only exception is environments, where we encourage you to use overrides, for reasons that will become clear in the environments section.

# Best Practice: Always use ‘default’ attributes in your cookbooks

- When setting an attribute in a cookbook, it should (almost) always be a **default** attribute
- There are exceptions, but they are rare

Monday, 30 June 14

We actually will describe, in depth, why this is – but we'll use object lessons throughout the class to do it. Have folks sit tight.

# Best Practice: Make sure cookbooks have default values

- If a cookbook needs an attribute to exist, it should
  1. Define a default value for it in an attribute file, or
  2. Depend on another cookbook that does
- **Never rely on an attribute being created manually**

Monday, 30 June 14  
This is important.

This is especially important in community cookbooks. When sharing code, if you depend on it make sure it is defined somewhere.

# Exercise: Upload the PCI cookbook

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

Monday, 30 June 14

Reiterate what happens here:

- \* Syntax check
- \* Upload to Chef Server

# Exercise: Upload the PCI cookbook

```
$ knife cookbook upload pci
```

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

Monday, 30 June 14

Reiterate what happens here:

- \* Syntax check
- \* Upload to Chef Server

# Exercise: Add the motd recipe to your test node's run list

```
node1:  
  run_list:  
    - recipe[apache]  
    - recipe[motd]
```

Monday, 30 June 14

Could use the -a switch to place the recipe after a specific recipe  
\$ knife node run\_list add node1 'recipe[motd]' -a 'recipe[apache]'

We are **leaving out the pci cookbook on purpose**, as we are going to cause a failure due to the missing attribute, and fix with cookbook dependencies

If someone spots them, congratulate them on their good eye, and keep going.

# Exercise: Add the motd recipe to your test node's run list

```
$ knife node run_list add node1 "recipe[motd]"
```

```
node1:  
  run_list:  
    - recipe[apache]  
    - recipe[motd]
```

Monday, 30 June 14

Could use the -a switch to place the recipe after a specific recipe  
\$ knife node run\_list add node1 'recipe[motd]' -a 'recipe[apache]'

We are **leaving out the pci cookbook on purpose**, as we are going to cause a failure due to the missing attribute, and fix with cookbook dependencies

If someone spots them, congratulate them on their good eye, and keep going.

# Exercise: Add the motd recipe to your test node's run list

```
$ knife node show node1
```

```
Node Name:      node1
Environment:    _default
FQDN:          ip-10-154-155-107.ec2.internal
IP:            54.242.35.165
Run List:       recipe[apache], recipe[motd]
Roles:
Recipes:
Platform:      ubuntu 10.04
Tags:
```

Monday, 30 June 14

JSON syntax rules for this class: use double quotes around strings, no trailing commas. Stick to that and you will survive Chef Bootcamp! No trailing commas in JSON! Most common syntax error.

We are **leaving out the pci cookbook on purpose**, as we are going to cause a failure due to the missing attribute, and fix with cookbook dependencies

If someone spots them, congratulate them on their good eye, and keep going.

# Exercise: Re-run the Chef Client

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

# Exercise: Re-run the Chef Client

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

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache", "motd"]
Synchronizing Cookbooks:
  - apache
  - motd
Compiling Cookbooks...
Converging 16 resources
Recipe: apache::default
  * package[apache2] action install (up to date)
  * service[apache2] action enable
    - enable service service[apache2]

  * service[apache2] action start (up to date)
  * execute[a2dissite default] action run (skipped due to only_if)
...
  * template[/srv/apache/bears/index.html] action create (up to date)
  * template[/etc/apache2/sites-available/ponies] action create (up to date)
  * execute[a2ensite ponies] action run (skipped due to not_if)
  * directory[/srv/apache/ponies] action create (up to date)
  * template[/srv/apache/ponies/index.html] action create (up to date)
Recipe: motd::default
  * template[/etc/motd.tail] action create
    - create new file /etc/motd.tail
=====
Error executing action `create` on resource 'template[/etc/motd.tail]'
```

Monday, 30 June 14

What went wrong!

# Exercise: Re-run the Chef Client

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

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache", "motd"]
Synchronizing Cookbooks:
  - apache
  - motd
Compiling Cookbooks...
Converging 16 resources
Recipe: apache::default
  * package[apache2]          action up_to_date
  * service[apache2]           action enable
  - enable service[apache2]
  * service[apache2]           action to_date
  * execute[a2dissite]         action run
  - run execute[a2dissite]
...
  * template[/srv/apache/ponies/index.html] action create (up to date)
  * template[/etc/apache/sites-available/ponies] action create (up to date)
  * execute[a2ensite]           action run
  - run execute[a2ensite]
  * directory[/srv/apache/ponies] action create (up to date)
  * template[/srv/apache/ponies/index.html] action create (up to date)
Recipe: motd::default
  * template[/etc/motd.tail]    action create
  - create new file /etc/motd.tail
=====
Error executing action `create` on resource 'template[/etc/motd.tail]'
```

FAIL!

Monday, 30 June 14

What went wrong!

# You probably see this at the bottom of your screen...

```
Template Context:
```

```
-----  
on line #3  
1: This server is the property of <%= node['company'] %>  
2: This Server is <%= node['hostname'] %>  
3: <% if node['pci']['in_scope'] -%>  
4: This server is in-scope for PCI compliance  
5: <% end -%>
```

```
[2013-11-05T16:26:25+00:00] ERROR: Running exception handlers  
[2013-11-05T16:26:25+00:00] ERROR: Exception handlers complete  
[2013-11-05T16:26:25+00:00] FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out  
Chef Client failed. 1 resources updated  
[2013-11-05T16:26:25+00:00] ERROR:
```

```
Chef::Mixin::Template::TemplateError (undefined method `[]' for nil:NilClass) on line #3:
```

Monday, 30 June 14

undefined method `[]' for nil:NilClass

Ruby is object oriented and functional, and everything you do has a return code. So we do:

```
* node (this is the node object)  
* ['pci'] (this is the method [], with an argument 'pci')  
** this returns 'nil', because there is no 'pci' attribute on the node, because we haven't evaluated the pci cookbook  
* ['in_scope'] gets called on 'nil', because that was the last return code
```

Hence - 'undefined method [] for nil:NilClass'. When you see this, it most often means you tried to look up a Node attribute that does not exist.

# Stack Traces

- A stack trace tells you where in a program an error occurred
- They can (obviously) be very detailed
- They can also be intensely useful, as they supply the data you need to find a problem

# Scroll up

- In this case, Chef actually knows exactly what went wrong.
- Scroll up to find out.

# Scroll up

- In this case, Chef actually knows exactly what went wrong.
- Scroll up to find out.

```
=====
Error executing action `create` on resource 'template[/etc/motd.tail]'
=====
Chef::Mixin::Template::TemplateError
-----
undefined method `[]' for nil:NilClass

Resource Declaration:
-----
# In /var/chef/cache/cookbooks/motd/recipes/default.rb

10: template "/etc/motd.tail" do
11:   source "motd.tail.erb"
12:   mode "0644"
13: end
```

# We do not have the attribute we are using in the conditional

```
INFO: Run List is [recipe[apache], recipe[motd]]  
INFO: Run List expands to [apache, motd]  
INFO: Starting Chef Run for node1.local  
INFO: Running start handlers  
INFO: Start handlers complete.  
resolving cookbooks for run list: ["apache", "motd"]  
INFO: Loading cookbooks [apache, motd]
```

# We do not have the attribute we are using in the conditional

```
INFO: Run List is [recipe[apache], recipe[motd]]  
INFO: Run List expands to [apache, motd]  
INFO: Starting Chef Run for node1.local  
INFO: Running start handlers  
INFO: Start handlers complete.  
resolving cookbooks for run list: ["apache", "motd"]  
INFO: Loading cookbooks [apache, motd]
```

- Can anyone guess why?

# We do not have the attribute we are using in the conditional

```
INFO: Run List is [recipe[apache], recipe[motd]]  
INFO: Run List expands to [apache, motd]  
INFO: Starting Chef Run for node1.local  
INFO: Running start handlers  
INFO: Start handlers complete.  
resolving cookbooks for run list: ["apache", "motd"]  
INFO: Loading cookbooks [apache, motd]
```

- Can anyone guess why?
- We did not load the PCI cookbook!

# Exercise: Add a dependency on the PCI cookbook to the MOTD cookbook



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

```
maintainer          "YOUR_COMPANY_NAME"
maintainer_email    "YOUR_EMAIL"
license             "All rights reserved"
description         "Installs/Configures motd"
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version            "0.1.0"
```

# Exercise: Add a dependency on the PCI cookbook to the MOTD cookbook



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

```
maintainer          "YOUR_COMPANY_NAME"
maintainer_email    "YOUR_EMAIL"
license             "All rights reserved"
description         "Installs/Configures motd"
long_description   IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version            "0.1.0"
depends "pci"
```

# Cookbook Metadata



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

```
maintainer          "YOUR_COMPANY_NAME"
maintainer_email    "YOUR_EMAIL"
license             "All rights reserved"
description         "Installs/Configures motd"
long_description   IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version             "0.1.0"
depends             "pci"
```

# Cookbook Metadata



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

```
maintainer          "YOUR_COMPANY_NAME"
maintainer_email    "YOUR_EMAIL"
license             "All rights reserved"
description         "Installs/Configures motd"
long_description   IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version             "0.1.0"
depends             "pci"
```

**SAVE FILE!**

- Cookbooks that depend on other cookbooks will cause the dependent cookbook to be downloaded to the client, and evaluated

## Cookbook Attributes are applied for all downloaded cookbooks!

- Cookbooks downloaded as dependencies will have their attribute files evaluated
- Even if there is no recipe from the cookbook in the run-list

# Exercise: Upload the motd cookbook

```
Uploading motd  
Uploaded 1 cookbook.
```

[ 0 . 1 . 0 ]

# Exercise: Upload the motd cookbook

```
$ knife cookbook upload motd
```

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

# Exercise: Re-run the Chef Client

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

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache", "motd"]
Synchronizing Cookbooks:
  - apache
  - motd
  - pci
Compiling Cookbooks...
Converging 16 resources
Recipe: apache::default
  * package[apache2] action install (up to date)
  * service[apache2] action enable
    - enable service service[apache2]

  * service[apache2] action start (up to date)
  * execute[a2dissite default] action run (skipped due to only_if)
...
  * directory[/srv/apache/ponies] action create (up to date)
  * template[/srv/apache/ponies/index.html] action create (up to date)
Recipe: motd::default
  * template[/etc/motd.tail] action create
    - update content in file /etc/motd.tail from e3b0c4 to f2120e
      --- /etc/motd.tail 2013-11-05 16:26:25.580688879 +0000
      +++ /tmp/chef-rendered-template20131105-4625-170b8xu 2013-11-05 16:39:23.204688879 +0000
      @@ -1 +1,3 @@
      +This server is the property of Opscode
    - change mode from '0644' to '0664'

Chef Client finished, 2 resources updated
```

Monday, 30 June 14

Notice we downloaded the pci cookbook, and we rendered motd.tail

# Exercise: Re-run the Chef Client

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

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache", "motd"]
Synchronizing Cookbooks:
  - apache
  - motd
  - pci
Compiling Cookbooks...
Converging 16 resources...
Recipe: apache::default
  * package[apache2] action install
  * service[apache2] action enable
    - enable service
  * service[apache2] action start
  * execute[a2dissite default] action run
    - run
...
  * directory[/srv/apache] action create
  * template[/srv/apache/index.html] action create
    - create
    +Apache2 index page
    +Content
      - content
        +This server is the property of Opscode
  - change mode from '0644' to '0664'

Recipe: motd::default
  * template[/etc/motd.tail] action create
    - create
    +Content
      - content
        +This server is the property of Opscode
  - change mode from '0644' to '0664'

Chef Client finished, 2 resources updated
```

WIN!

Monday, 30 June 14

Notice we downloaded the pci cookbook, and we rendered motd.tail

# Exercise: Check your work

```
opscode@node1:~$ cat /etc/motd.tail
```

This server is property of Opscode

# Exercise: Show your test node's pci attribute

```
$ knife search node "pci:*" -a pci
```

```
1 items found
```

```
id:      node1
pci:
in_scope:  false
```

Monday, 30 June 14

Think about what just happened – if you needed a report about being in/out of scope, you can generate it trivially with Chef's search features.

In this example, a simple primitive has been turned into a powerful reporting tool.

# Exercise: Set attribute to true

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

```
default['pci']['in_scope'] = true
```

**SAVE FILE!**

- Set the attribute to true

# Exercise: Upload the PCI cookbook

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

Monday, 30 June 14

Reiterate what happens here:

- \* Syntax check
- \* Upload to Chef Server

# Exercise: Upload the PCI cookbook

```
$ knife cookbook upload pci
```

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

Monday, 30 June 14

Reiterate what happens here:

- \* Syntax check
- \* Upload to Chef Server

# Exercise: Re-run the Chef Client

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

```
Starting Chef Client, version 11.8.0
resolving cookbooks for run list: ["apache", "motd"]
Synchronizing Cookbooks:
  - apache
  - motd
  - pci
Compiling Cookbooks...
Converging 16 resources
Recipe: apache::default
  * package[apache2] action install (up to date)
  * service[apache2] action enable
    - enable service service[apache2]

  * service[apache2] action start (up to date)
  * execute[a2dissite default] action run (skipped due to only_if)
...
  * directory[/srv/apache/ponies] action create (up to date)
  * template[/srv/apache/ponies/index.html] action create (up to date)
Recipe: motd::default
  * template[/etc/motd.tail] action create
    - update content in file /etc/motd.tail from e3b0c4 to f2120e
      --- /etc/motd.tail 2013-11-05 16:26:25.580688879 +0000
      +++ /tmp/chef-rendered-template20131105-4625-170b8xu 2013-11-05 16:39:23.204688879 +0000
      @@ -1 +1,3 @@
      +This server is the property of Opscode
      +This server is in-scope for PCI compliance
    - change mode from '0644' to '0664'

Chef Client finished, 2 resources updated
```

Monday, 30 June 14

Notice we downloaded the pci cookbook, and we rendered motd.tail

# Exercise: Check your work

```
opscode@node1:~$ cat /etc/motd.tail
```

This server is property of Opscode  
This server is in-scope for PCI compliance

# Exercise: Show your test node's pci attribute

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

```
1 items found

id: node1
pci:
in_scope: true
```

# Congratulations!

- You now know the 3 most important resources in the history of configuration management
  - Package
  - Template
  - Service

Monday, 30 June 14

These 3 are the holy trinity, and they are the most important because almost all work you do to configure an application uses them. Install the service, render its configuration, and restart/reload the service.

# Review Questions

- What goes in a cookbook's attribute files?
- What are the 4 different levels of precedence?
- When do you need to specify a cookbook dependency?
- What does <%= mean, and where will you encounter it?
- What are the 3 most important resources in configuration management?

# Template Variables, Notifications, and Controlling Idempotency

Refactoring the Apache Cookbook

v1.2.3



Monday, 30 June 14

This section has a significant jump in complexity from the last two. Make sure you take a break before you start it, and get a cup of coffee.

AKA "I really hate that cookbook I wrote six months ago"

# Refactoring the Apache Cookbook

Execute, Not If/Only If, Directories, Notifications, Template Variables, and the Chef Docs Site

v1.2.3



Monday, 30 June 14

This section has a significant jump in complexity from the last two. Make sure you take a break before you start it, and get a cup of coffee.

AKA "I really hate that cookbook I wrote six months ago"

# Lesson Objectives

- After completing the lesson, you will be able to
  - Use the **execute** resource
  - Control idempotence manually with **not\_if** and **only\_if**
  - Navigate the Resources page on [docs.opscode.com](http://docs.opscode.com)
  - Describe the Directory resource
  - Use resource notifications
  - Explain what Template Variables are, and how to use them
  - Use Ruby variables, loops, and string expansion

# The Problem and the Success Criteria

---

- **The Problem:** We need to deploy multiple custom home pages running on different ports
- **Success Criteria:** Be able to view our custom home page

Monday, 30 June 14

Warn students not to skip ahead to saving their cookbook in this module. They don't always have to follow along exactly. But here, if they save early, they will see inconsistencies in later modules. Hold off on uploading your cookbook until we get there together.

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



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

```
maintainer          "YOUR_COMPANY_NAME"
maintainer_email    "YOUR_EMAIL"
license             "All rights reserved"
description         "Installs/Configures apache"
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version            "0.2.0"
```

**SAVE FILE!**

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

Monday, 30 June 14

Major refactoring, so we should bump the cookbook version # first.

We want students to do this first, so they don't wreck their known-good-state 0.1.0 apache cookbook by uploading in the middle.

Also, we are going to use this version bump to show off environments and illustrate that Chef requires you to declare \*all\* your desired state, even contra state. This will happen later. :)

# Exercise: Create a default.rb attribute file



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

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

**SAVE FILE!**

- We add information about the sites we need to deploy
  - One about Clowns, running on port 80
  - One about Bears, running on port 81

# Exercise: Open the default apache recipe in your editor



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

```
#  
# Cookbook Name:: apache  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
#  
package "apache2" do  
  action :install  
end  
  
service "apache2" do  
  action [:enable, :start]  
end  
  
cookbook_file "/var/www/index.html" do  
  source "index.html"  
  mode "0644"  
end
```

**SAVE FILE!**

Monday, 30 June 14

A lot is about to change here – it's worth warning folks that we're going to do a significant refactor.

Also, we are going to force a likely bug. The refactored recipe will have the apache configuration rendered after the service – which means that, if the user makes a mistake in writing the template, they will wind up in a state that cannot be recovered from with chef (because the service apache2 should be running, and the broken config means it cannot start.)

The fix is to move the service "apache2" resource to the bottom of the recipe, which, if they encounter it, you should encourage them to do.

Use it as an object lesson about why order matters, and a best practice illustration that starting services should always happen at the end of a recipe, not at the beginning.

# Exercise: Use execute resource to disable the default Apache virtual host

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

```
service "apache2" do
  action [:enable, :start]
end

execute "a2dissite default" do
  only_if do
    File.symlink?("/etc/apache2/sites-enabled/000-default")
  end
  notifies :restart, "service[apache2]"
end

cookbook_file "/var/www/index.html" do
```

## SAVE FILE!

- Runs the command "a2dissite default", but only if the symlink exists
- If the action succeeds, restart Apache

Monday, 30 June 14

a2dissite and a2ensite are the debian/ubuntu utilities for managing apache virtual hosts, not part of Chef.

We encounter 3 new pieces of syntax here:

\* do..end being used as a block. depending on the crowd, explain anonymous functions and lambdas, or just explain that it means "execute this code and return the result".

\* File – this is a class name

\* File.symlink? is a class method, which returns true/false based on whether the symlink exists. In idiomatic Ruby, methods that end with a ? return true or false.

The next 5 or 6 slides go into greater detail.

# Execute resources are generally not idempotent

- Chef will stop your run if a resource fails
- Most command line utilities are not idempotent - they assume a human being is interacting with, and understands, the state of the system
- The result is - it's up to you to make execute resources idempotent

Monday, 30 June 14

If you have to color outside the lines, it's up to you to make stuff work.

# Enter the `not_if` and `only_if` metaparameters

```
only_if do
  File.symlink?("/etc/apache2/sites-enabled/000-default")
end
```

- The `only_if` parameter causes the resources actions to be taken only if its argument returns true
- The `not_if` parameter is the opposite of `only_if` - the actions are taken only if its argument returns false

Monday, 30 June 14

`only_if` and `not_if` are part of Chef, not part of Ruby

We don't use "If" and "Unless" because they are reserved for the language that is hosting the DSL

# Best Practice: The Chef Docs Site

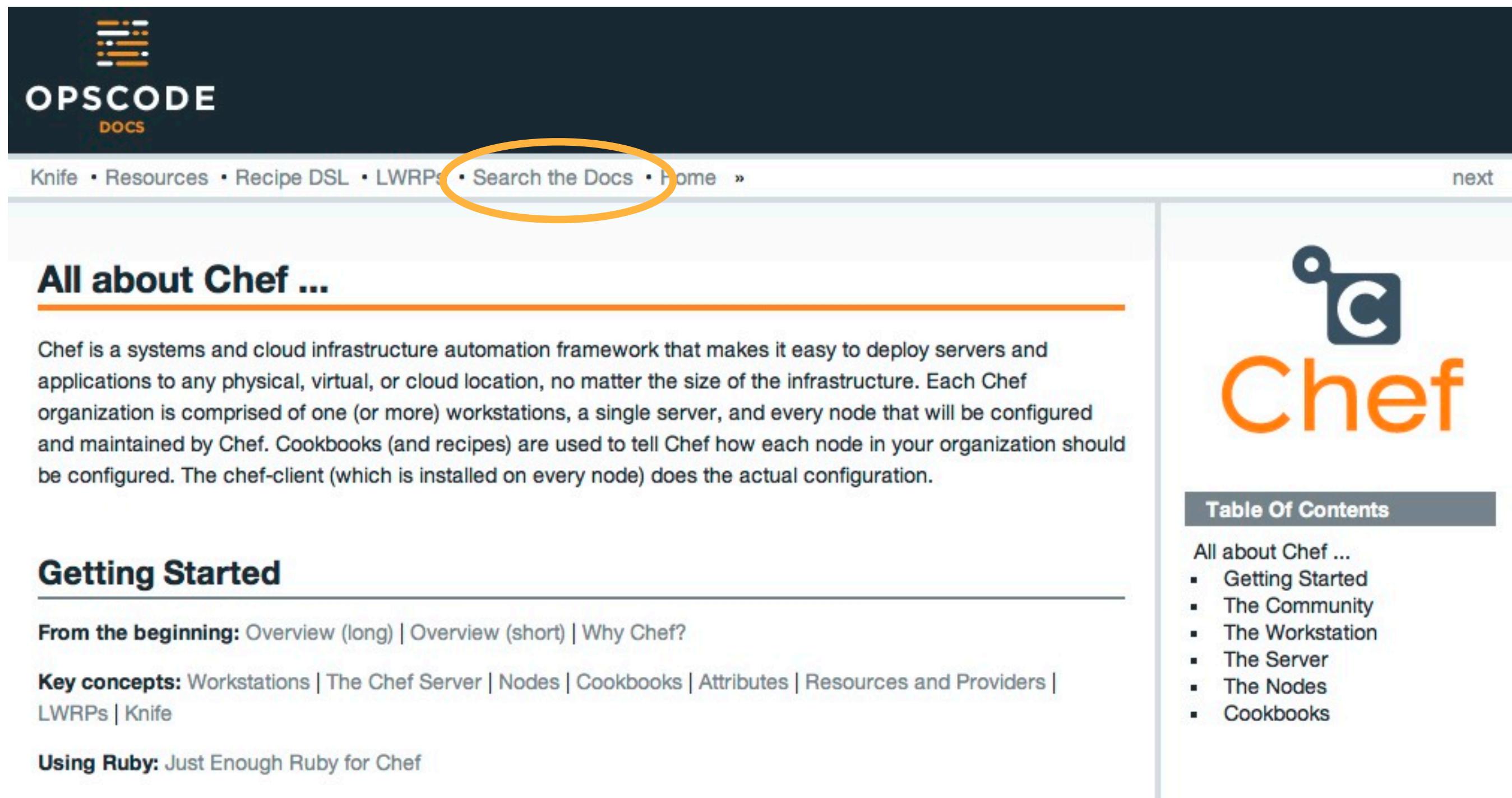
- The Chef Docs Site is the home for all of the documentation about Chef.
  - It is very comprehensive
  - It has a page on every topic
- <http://docs.opscode.com>
- Let's use the docs to learn more about **not\_if** and **only\_if**

Monday, 30 June 14

How the heck are you supposed to know the valid parameters for these?

This class is teaching you how people who are good with chef use chef. Therefore... docs!

# Exercise: Search for more information about Resources



The screenshot shows a portion of the Opscode Chef documentation website. At the top, there's a dark header with the "OPSCODE DOCS" logo. Below the header, a navigation bar includes links for "Knife", "Resources", "Recipe DSL", "LWRPs", "Search the Docs" (which is circled in orange), and "Home". To the right of the navigation bar, there are "next" and "previous" links. The main content area has a title "All about Chef ...". Below the title is a paragraph of text describing Chef as a systems and cloud infrastructure automation framework. Underneath the text, there's a section titled "Getting Started" with links to "From the beginning", "Key concepts", and "Using Ruby". On the right side of the page, there's a sidebar with the "Chef" logo and a "Table Of Contents" section containing a list of topics.

OPS CODE  
DOCS

Knife · Resources · Recipe DSL · LWRPs · **Search the Docs** · Home »

next

## All about Chef ...

Chef is a systems and cloud infrastructure automation framework that makes it easy to deploy servers and applications to any physical, virtual, or cloud location, no matter the size of the infrastructure. Each Chef organization is comprised of one (or more) workstations, a single server, and every node that will be configured and maintained by Chef. Cookbooks (and recipes) are used to tell Chef how each node in your organization should be configured. The chef-client (which is installed on every node) does the actual configuration.

### Getting Started

**From the beginning:** Overview (long) | Overview (short) | Why Chef?

**Key concepts:** Workstations | The Chef Server | Nodes | Cookbooks | Attributes | Resources and Providers | LWRPs | Knife

**Using Ruby:** Just Enough Ruby for Chef

### Table Of Contents

- All about Chef ...
  - Getting Started
  - The Community
  - The Workstation
  - The Server
  - The Nodes
  - Cookbooks

# Exercise: Search for more information about Resources

## Search the Documentation for Chef

From here you can search the documentation for Chef. Enter your search words into the box below and click the search button. The search will query all of the documentation for Chef. When you click a search result to view it, a new window will be opened.

resources

All **Chef Documentation** Cookbooks

About 817 results (0.20 seconds) Sort by: Relevance

**[About Resources and Providers — Chef Docs](#)**  
[docs.opscode.com/resource.html](http://docs.opscode.com/resource.html)  
If you want to see all of the information about Chef **resources** in a single document, see:  
<http://docs.opscode.com/chef/resources.html>. (This document also ...  
Labeled Chef ...

**[Resources and Providers Reference — Chef](#)**  
[docs.opscode.com/chef/resources.html](http://docs.opscode.com/chef/resources.html)  
A **resource** is a key part of a recipe. A **resource** defines the actions that can be taken, such as when a package should be installed, whether a service should be ...  
Labeled Chef ...



- Find "Resources and Providers Reference"

# The Resources Page

## Resources and Providers Reference

A resource is a key part of a recipe. A resource defines the actions that can be taken, such as when a package should be installed, whether a service should be enabled or restarted, which groups, users, or groups of users should be created, where to put a collection of files, what the name of a new directory should be, and so on. During a chef-client run, each resource is identified and then associated with a provider. The provider then does the work to complete the action defined by the resource. Each resource is processed in the same order as they appear in a recipe. The chef-client ensures that the same actions are taken the same way everywhere and that actions produce the same result every time. A resource is implemented within a recipe using Ruby.

Where a resource represents a piece of the system (and its desired state), a provider defines the steps that are needed to bring that piece of the system from its current state into the desired state. These steps are de-coupled from the request itself. The request is made in a recipe and is defined by a lightweight resource. The steps are then defined by a lightweight provider.

The `Chef::Platform` class maps providers to platforms (and platform versions). Ohai, as part of every chef-client run, verifies the `platform` and `platform_version` attributes on each node. The chef-client then uses those values to identify the correct provider, build an instance of that provider, identify the current state of the resource, do the specified action, and then mark the resource as updated (if changes were made). For example, given the following resource:

```
directory "/tmp/folder" do
  owner "root"
  group "root"
  mode 0755
  action :create
end
```



### Table Of Contents

- Resources and Providers Reference
  - Common Functionality for all Resources
    - Actions
      - Examples
    - Attributes
      - Examples
    - Guards
      - Attributes
      - Arguments
      - `not_if` Examples
      - `only_if` Examples
    - Lazy Attribute Evaluation
  - Notifications
    - Notifications Timers
    - Notifies Syntax
      - Examples
    - Subscribes Syntax

278

Monday, 30 June 14

Let them know that this is how professionals work – they have their editor open, a command line, and the docs open all the time.

The rest of the deck tells them to look up more information in the Docs, and at this stage I start answering questions with "where can you look that up?"

Note: The docs has an easier way to run the nested check we just talked about. We did it to have the conversation.

# Notifications

```
notifies :restart, "service[apache2]"
```

- Resource Notifications in Chef are used to trigger an action on a resource when the current resources actions are successful.
- "If we delete the site, restart apache"
- The first argument is an action, and the second argument is the string representation of a given resource
- Like not\_if and only\_if, **notifies** is a resource metaparameter - any resource can notify any other

Monday, 30 June 14

Direct the students to read more about notifications on the wiki (notifies resource)

Important to talk about the fact that we batch notifications up, and run them at the end

The `:immediately` flag is a good question to see if they know how to use the wiki – ask them if they can figure out how to not wait for the end of the run for a notification to fire

# Exercise: Iterate over each apache site



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

```
execute "a2dissite default" do
  only_if do
    File.symlink?("/etc/apache2/sites-enabled/000-default")
  end
  notifies :restart, "service[apache2]"
end

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

- **Delete the cookbook\_file resource**

# Exercise: Iterate over each apache site



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

```
execute "a2dissite default" do
  only_if do
    File.symlink?("/etc/apache2/sites-enabled/000-default")
  end
  notifies :restart, "service[apache2]"
end

node['apache']['sites'].each do |site_name, site_data|
  document_root = "/srv/apache/#{site_name}"
```

**SAVE FILE!**

- **Delete the cookbook\_file resource**
- `node['apache']['sites']` is a ruby hash, with keys and values

# Exercise: Iterate over each apache site

```
node['apache']['sites'].each do |site_name, site_data|
  document_root = "/srv/apache/#{site_name}"
```

- Calling .each loops over each site

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

- First pass
  - site\_name = 'clowns'
  - site\_data = { "port" => 80 }
- Second pass
  - site\_name = 'bears'
  - site\_data = { "port" => 81 }

Monday, 30 June 14

Line 25, using the hash we wrote earlier, ".each" is (more or less) a for loop. For every item, iterate the function. Works just like a for loop. Ruby does have a "for" loop, but it is rarely used. Instead, we use "iterators" to yield values to blocks.

In this case, we are walking all the sites we defined earlier, and assigning two pieces of data between the "|" (arms) – site\_name for the key, and site\_data for the value.

We are breaking up arrays and hashes.

# Exercise: Iterate over each apache site

```
node['apache']['sites'].each do |site_name, site_data|
  document_root = "/srv/apache/#{site_name}"
```

- Create a variable called `document_root`
- `#{}{site_name}` means "insert the value of `site_name` here"
- First pass
  - The value is the string `"/srv/apache/clowns"`
- Second pass
  - The value is the string `"/srv/apache/bears"`

Monday, 30 June 14

More syntax. Variables within blocks use the `=` for assignment, and are locally scoped.

`#{}{..}` in a string

# Exercise: Add a template for Apache virtual host configuration

```
node['apache']['sites'].each do |site_name, site_data|
  document_root = "/srv/apache/#{site_name}"

  template "/etc/apache2/sites-available/#{site_name}" do
    source "custom.erb"
    mode "0644"
    variables(
      :document_root => document_root,
      :port => site_data['port']
    )
    notifies :restart, "service[apache2]"
  end
end
```

Monday, 30 June 14

New resource parameter for templates, in the "variables" parameter. Explained in future slides.

# Template Variables

- Not all data you might need in a template is necessarily node attributes
- The **variables** parameter lets you pass in custom data for use in a template

Monday, 30 June 14

We will show how to use these variables when we write the template in a few slides

# Exercise: Add an execute resource to enable new virtual host

```
template "/etc/apache2/sites-available/#{site_name}" do
  source "custom.erb"
  mode "0644"
  variables(
    :document_root => document_root,
    :port => site_data['port']
  )
  notifies :restart, "service[apache2]"
end

execute "a2ensite #{site_name}" do
  not_if do
    File.symlink?("/etc/apache2/sites-enabled/#{site_name}")
  end
  notifies :restart, "service[apache2]"
end
```

Monday, 30 June 14

Nothing new here, other than the use of `not_if` rather than `only_if`

## Exercise: Add a directory resource to create the document\_root

- Use a **directory** resource
- The name is `document_root`
- The resource has two parameters
  - **mode** is "0755"
  - **recursive** is true
- Use the Resources page on the Docs Site to read more about what **recursive** does.

Monday, 30 June 14

Make them use the wiki, and let them sweat a bit. The next slide has the answer.

If students want to check their syntax, encourage them to use "`knife cookbook test [cookbook]`" rather than uploading their changes just yet. Typically, they can apply updates as often as needed in uploading cookbooks. But if we do that now, our later examples will be off.

# The directory resource

```
execute "a2ensite #{site_name}" do
  not_if do
    File.symlink?("/etc/apache2/sites-enabled/#{site_name}")
  end
  notifies :restart, "service[apache2]"
end
```

```
directory document_root do
  mode "0755"
  recursive true
end
```

## Exercise: Add a template resource for the virtual host's index.html

```
directory document_root do
  mode "0755"
  recursive true
end

template "#{document_root}/index.html" do
  source "index.html.erb"
  mode "0644"
  variables(
    :site_name => site_name,
    :port => site_data['port']
  )
end
end
```

# Don't forget the last "end"

```
template "#{document_root}/index.html" do
  source "index.html.erb"
  mode "0644"
  variables(
    :site_name => site_name,
    :port => site_data['port']
  )
end
end
```

# Don't forget the last "end"

```
template "#{document_root}/index.html" do
  source "index.html.erb"
  mode "0644"
  variables(
    :site_name => site_name,
    :port => site_data['port']
  )
end
end
```

See the correct, whole file at <https://gist.github.com/2866378>

# Exercise: Add custom.erb to your templates directory



**OPEN IN EDITOR:** cookbooks/apache/templates/default/custom.erb

- Note the two **template variables** are prefixed with an @ symbol
- Our first conditional if!

```
<% if @port != 80 -%>
  Listen <%= @port %>
<% end -%>

<VirtualHost *:<%= @port %>>
  ServerAdmin webmaster@localhost

  DocumentRoot <%= @document_root %>
  <Directory />
    Options FollowSymLinks
    AllowOverride None
  </Directory>
  <Directory <%= @document_root %>>
    Options Indexes FollowSymLinks Multiviews
    AllowOverride None
    Order allow,deny
    allow from all
  </Directory>
</VirtualHost>
```

**SAVE FILE!**

Monday, 30 June 14

New syntax here:

\* Ruby has "instance" variables, which are bound to the instance of an object. When we render templates and pass in the variables parameter, we are adding instance variables to the context of the template render. Hence, we prefix them with an "@" symbol. The node, on the other hand, is supplied as part of the context itself, and so requires no @ symbol.

\* Conditional if – ruby has the pretty standard set of operators for checking equality.

Re-affirm the ERB syntax, of -%> on the end and <%= on the front

# Exercise: Add custom.erb to your templates directory



**OPEN IN EDITOR:** cookbooks/apache/templates/default/custom.erb

- Note the two **template variables** are prefixed with an @ symbol
- Our first conditional if!
- If you are feeling hardcore, type it.
- <https://gist.github.com/2866454>

```
<% if @port != 80 -%>
  Listen <%= @port %>
<% end -%>

<VirtualHost *:<%= @port %>>
  ServerAdmin webmaster@localhost

  DocumentRoot <%= @document_root %>
  <Directory />
    Options FollowSymLinks
    AllowOverride None
  </Directory>
  <Directory <%= @document_root %>>
    Options Indexes FollowSymLinks Multiviews
    AllowOverride None
    Order allow,deny
    allow from all
  </Directory>
</VirtualHost>
```

**SAVE FILE!**

Monday, 30 June 14

New syntax here:

\* Ruby has "instance" variables, which are bound to the instance of an object. When we render templates and pass in the variables parameter, we are adding instance variables to the context of the template render. Hence, we prefix them with an "@" symbol. The node, on the other hand, is supplied as part of the context itself, and so requires no @ symbol.

\* Conditional if – ruby has the pretty standard set of operators for checking equality.

Re-affirm the ERB syntax, of -%> on the end and <%= on the front

# Exercise: Add index.html.erb to your templates directory



**OPEN IN EDITOR:** cookbooks/apache/templates/default/index.html.erb

```
<html>
  <body>
    <h1>Welcome to <%= node['company'] %></h1>
    <h2>We love <%= @site_name %></h2>
    <%= node['ipaddress'] %>:<%= @port %>
  </body>
</html>
```

**SAVE FILE!**

- Note the two **template variables** are prefixed with an @ symbol

Monday, 30 June 14

When we render templates and pass in the variables parameter, we are adding instance variables to the context of the template render. Hence, we prefix them with an "@" symbol. The node, on the other hand, is supplied as part of the context itself, and so requires no @ symbol.

# Exercise: Add index.html.erb to your templates directory



**OPEN IN EDITOR:** cookbooks/apache/templates/default/index.html.erb

```
<html>
  <body>
    <h1>Welcome to <%= node['company'] %></h1>
    <h2>We love <%= @site_name %></h2>
    <%= node['ipaddress'] %>:<%= @port %>
  </body>
</html>
```

**SAVE FILE!**

- Note the two **template variables** are prefixed with an @ symbol
- <https://gist.github.com/2866421>

Monday, 30 June 14

When we render templates and pass in the variables parameter, we are adding instance variables to the context of the template render. Hence, we prefix them with an "@" symbol. The node, on the other hand, is supplied as part of the context itself, and so requires no @ symbol.

# Exercise: Upload the Apache cookbook

```
Uploading apache  
Uploaded 1 cookbook.
```

[ 0 . 2 . 0 ]

# Exercise: Upload the Apache cookbook

```
$ knife cookbook upload apache
```

Uploading apache

[ 0 . 2 . 0 ]

Uploaded 1 cookbook.

# Exercise: Re-run the Chef Client

```
Compiling Cookbooks...
Converging 12 resources
Recipe: apache::default
  * package[apache2] action install
INFO: Processing package[apache2] action install (apache::default line 10)
    (up to date)
  * service[apache2] action start
INFO: Processing service[apache2] action start (apache::default line 14)
    (up to date)
  * service[apache2] action enable
INFO: Processing service[apache2] action enable (apache::default line 14)
    (up to date)
  * execute[a2dissite default] action run
INFO: Processing execute[a2dissite default] action run (apache::default line 18)
Site default disabled.
To activate the new configuration, you need to run:
  service apache2 reload
INFO: execute[a2dissite default] ran successfully

  - execute a2dissite default
```

Monday, 30 June 14

We ran a resource with a notification, which...

# Exercise: Re-run the Chef Client

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

```
Compiling Cookbooks...
Converging 12 resources
Recipe: apache::default
  * package[apache2] action install
INFO: Processing package[apache2] action install (apache::default line 10)
    (up to date)
  * service[apache2] action start
INFO: Processing service[apache2] action start (apache::default line 14)
    (up to date)
  * service[apache2] action enable
INFO: Processing service[apache2] action enable (apache::default line 14)
    (up to date)
  * execute[a2dissite default] action run
INFO: Processing execute[a2dissite default] action run (apache::default line 18)
Site default disabled.
To activate the new configuration, you need to run:
  service apache2 reload
INFO: execute[a2dissite default] ran successfully

  - execute a2dissite default
```

Monday, 30 June 14

We ran a resource with a notification, which...

# Exercise: Re-run the Chef Client

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

```
INFO: execute[a2dissite default] sending restart action to service[apache2]
(delayed)
Recipe: apache::default
  * service[apache2] action restart
INFO: Processing service[apache2] action restart (apache::default line 14)
INFO: service[apache2] restarted

  - restart service service[apache2]
```

```
INFO: Chef Run complete in 5.559923708 seconds
INFO: Running report handlers
INFO: Report handlers complete
Chef Client finished, 10 resources updated
```

Monday, 30 June 14

Triggered the restart

Again, service starts, are typically the last resource executed.

# Exercise: Verify the two sites are working!



Welcome to opscode  
We love clowns

10.4.25.155:80



Welcome to opscode  
We love bears

10.4.25.155:81

Monday, 30 June 14

Again, if they have trouble getting to them with a web browser, whip out curl on the command line

```
curl http://localhost:80
curl http://localhost:81
```

## Best Practice: Recipes contain the pattern, attributes supply the details

- Recipes contain the pattern for how to do something. ("How we deploy apache virtual hosts")
- Attributes contain the details. ("What virtual hosts should we deploy?")

Monday, 30 June 14

This is a re-iteration from earlier, but it's one of the most important things to understand about Chef.

# Review Questions

- How do you control the idempotence of an Execute resource?
- Where can you learn the details about all the core resources in Chef?
- What is a notification?
- What is a template variable?
- What does `#{foo}` do in a Ruby string?

# Recipe Inclusion, Data Bags, and Search

Writing a Users cookbook

v1.2.3



Monday, 30 June 14

# Writing a Users cookbook

Users, Groups, Data Bags, Recipe Inclusion and Search

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Explain what Data Bags are, and how they are used
  - Use the User and Group resources
  - Use `include_recipe`
  - Describe the role Search plays in recipes

# The Problem and the Success Criteria

- **The Problem:** Employees should have local user accounts created on servers, along with custom groups
- **Success Criteria:** We can add new employees and groups to servers dynamically

# Where should we store the user data?

- As we've seen, we could start by storing information about users as Node Attributes
- This is sort of a bummer, because we would be duplicating a lot of information - every user in the company would be stored in every Node object!
- Additionally, it would be very hard to integrate such a solution with another source of truth about users

# Introducing Data Bags

- A data bag is a **container** for **items** that represent information about your infrastructure that is not tied to a single node
- Examples
  - Users
  - Groups
  - Application Release Information

Monday, 30 June 14

These "containers" take JSON. Anything you can do in JSON you can do in a databag.

# Make a data\_bags directory

```
$ mkdir data_bags
```

(No output)

# Exercise: Create a data bag named users

```
$ mkdir data_bags/users  
$ knife data_bag create users
```

```
Created data_bag[users]
```

# Exercise: Create a user item in the users data bag



**OPEN IN EDITOR:** data\_bags/users/bobo.json

```
{  
  "id": "bobo",  
  "comment": "Bobo T. Clown",  
  "uid": 2000,  
  "gid": 0,  
  "home": "/home/bobo",  
  "shell": "/bin/bash"  
}
```

**SAVE FILE!**

Monday, 30 June 14

New syntax – integers in JSON have no quotes, and will wind up as integers in Ruby. If you quoted integers they'd be strings.  
Bobo is a Clown, that I have been using as my test user for identity management systems for years.

# Exercise: Create the data bag item

```
$ knife data_bag from file users bobo.json
```

```
Updated data_bag_item[users::bobo]
```

# Exercise: Create another user in the users data bag



**OPEN IN EDITOR:**

```
{  
  "id": "frank",  
  "comment": "Frank Belson",  
  "uid": 2001,  
  "gid": 0,  
  "home": "/home/frank",  
  "shell": "/bin/bash"  
}
```

**SAVE FILE!**

Monday, 30 June 14

Frank Belson is the sergeant of the police precinct in the Spenser novels by Robert B. Parker (later: Spenser for Hire TV series)

# Exercise: Create another user in the users data bag



**OPEN IN EDITOR:** data\_bags/users/frank.json

```
{  
  "id": "frank",  
  "comment": "Frank Belson",  
  "uid": 2001,  
  "gid": 0,  
  "home": "/home/frank",  
  "shell": "/bin/bash"  
}
```

**SAVE FILE!**

Monday, 30 June 14

Frank Belson is the sergeant of the police precinct in the Spenser novels by Robert B. Parker (later: Spenser for Hire TV series)

# Exercise: Create the data bag item

```
Updated data_bag_item[users::frank]
```

# Exercise: Create the data bag item

```
$ knife data_bag from file users frank.json
```

```
Updated data_bag_item[users::frank]
```

# Exercise: Show all the items in users data bag

```
2 items found

chef_type: data_bag_item
comment: Frank Belson
data_bag: users
gid: 0
home: /home/frank
id: frank
shell: /bin/bash
uid: 2001

chef_type: data_bag_item
comment: Bobo T. Clown
data_bag: users
gid: 0
home: /home/bobo
id: bobo
shell: /bin/bash
uid: 2000
```

Monday, 30 June 14

What's important here is that more than just the nodes get indexed for search – it is everything in Chef!

Search Index Names: role

node

client

environment

(data bags are indexed by data bag's name)

# Exercise: Show all the items in users data bag

```
$ knife search users "*:*"
```

```
2 items found

chef_type: data_bag_item
comment: Frank Belson
data_bag: users
gid: 0
home: /home/frank
id: frank
shell: /bin/bash
uid: 2001

chef_type: data_bag_item
comment: Bobo T. Clown
data_bag: users
gid: 0
home: /home/bobo
id: bobo
shell: /bin/bash
uid: 2000
```

Monday, 30 June 14

What's important here is that more than just the nodes get indexed for search – it is everything in Chef!

Search Index Names: role

node

client

environment

(data bags are indexed by data bag's name)

# Exercise: Find Bobo's shell in Chef

```
1 items found

data_bag_item_users_bobo:
  shell: /bin/bash
```

Monday, 30 June 14

Encourage students to try searches: knife help search

You can use boolean values e.g. knife search INDEX 'field:value OR field:value'

Could have them count the shell's in use, like this:-

```
knife search users "*:*" -a shell |grep shell | uniq -c
```

Note: if the search text is a string you'll need to put it in double quotes ("")

```
knife search users "comment:\\"Bobo T. Clown\\\" -a shell
```

Note we have to escape the " here within the search string. On a Mac/\*nix machine they could do this.

```
knife search users 'comment:"Bobo T. Clown"' -a shell
```

But the single quote ('') wouldn't work in a Windows powershell terminal

# Exercise: Find Bobo's shell in Chef

```
$ knife search users "id:bobo" -a shell
```

```
1 items found

data_bag_item_users_bobo:
  shell: /bin/bash
```

Monday, 30 June 14

Encourage students to try searches: knife help search

You can use boolean values e.g. knife search INDEX 'field:value OR field:value'

Could have them count the shell's in use, like this:-

```
knife search users "*:*" -a shell |grep shell | uniq -c
```

Note: if the search text is a string you'll need to put it in double quotes ("")

```
knife search users "comment:\\"Bobo T. Clown\\\" -a shell
```

Note we have to escape the " here within the search string. On a Mac/\*nix machine they could do this.

```
knife search users 'comment:"Bobo T. Clown"' -a shell
```

But the single quote ('') wouldn't work in a Windows powershell terminal

# Exercise: Create a data bag named groups

```
Created data_bag[groups]
```

# Exercise: Create a data bag named groups

```
$ mkdir data_bags/groups  
$ knife data_bag create groups
```

```
Created data_bag[groups]
```

# Exercise: Create a group item in the group data bag



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

```
{  
  "id": "clowns",  
  "gid": 3000,  
  "members": [ "bobo", "frank" ]  
}
```

**SAVE FILE!**

Monday, 30 June 14

JSON syntax again. Arrays in JSON have brackets.

Quotes around strings.

No trailing commas.

# Exercise: Create the data bag item

```
Updated data_bag_item[groups::clowns]
```

# Exercise: Create the data bag item

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

```
Updated data_bag_item[groups::clowns]
```

# Exercise: Show all the groups in Chef

```
1 items found

chef_type:    data_bag_item
data_bag:      groups
gid:          3000
id:           clowns
members:
  bobo
  frank
```

# Exercise: Show all the groups in Chef

```
$ knife search groups "*:*"
```

```
1 items found
```

```
chef_type:    data_bag_item
data_bag:      groups
gid:          3000
id:           clowns
members:
  bobo
  frank
```

# Exercise: Create a cookbook named ‘users’

- \*\* Creating cookbook users
- \*\* Creating README for cookbook: users
- \*\* Creating CHANGELOG for cookbook: users
- \*\* Creating metadata for cookbook: users

# Exercise: Create a cookbook named ‘users’

```
$ knife cookbook create users
```

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

# Exercise: Open the default recipe in your editor



**OPEN IN EDITOR:**

```
#  
# Cookbook Name:: users  
# Recipe:: default  
  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
#
```

**SAVE FILE!**

# Exercise: Open the default recipe in your editor



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

```
#  
# Cookbook Name:: users  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
#
```

**SAVE FILE!**

# Exercise: Open the default recipe in your editor

```
search( :users, "*:*").each do |user_data|
  user user_data['id'] do
    comment user_data['comment']
    uid user_data['uid']
    gid user_data['gid']
    home user_data['home']
    shell user_data['shell']
  end
end

include_recipe "users::groups"
```

- We use the same search we just tried with Knife in the recipe
- Each item is bound to `user_data`
- Use the Chef Docs for information about the `user` resource

Monday, 30 June 14

The search syntax is new here, but we've been doing lots of searching throughout. Relate that the objects bound to `user_data` during each iteration are the same as what we got in the search results.

# Exercise: Open the default recipe in your editor

```
search( :users, "*:*").each do |user_data|
  user user_data['id'] do
    comment user_data['comment']
    uid user_data['uid']
    gid user_data['gid']
    home user_data['home']
    shell user_data['shell']
  end
end

include_recipe "users::groups"
```

- **include\_recipe** ensures another recipes resources are complete before we continue
- Chef will only include each recipe once
- **include\_attribute** does the same, but for attribute files

Monday, 30 June 14

Note: we have not created the users::groups recipe yet

"include\_recipe" happens in order. When we get to that line, we check to see if this recipe has already been executed. If it has not been run yet, we run it now.

We run this after user resource to add users to the group they belong to. If opposite order, we'd try to add users that don't exist yet to our new group.

## Best Practice: Use `include_recipe` and `include_attribute` liberally

- If there is a pre-requisite for your recipe that resides in another recipe (the JVM existing for your Java application, for example)
  - Always use `include_recipe` to include it specifically, even if you put it in a run list
- The same goes for `include_attribute`

Monday, 30 June 14

The reason is that the recipes themselves stand alone as declarations of intent – no hidden dependencies that only get expressed in the order of the run list, and are easy to mess up.

You want a recipe to read as standalone. Basically, you're declaring intent. You declare that this is a dependency. This is a documentation trail. If you only declare this somewhere else, you get spaghetti code references and it gets convoluted.

Purists will say only use includes once. Pragmatists will say, use these liberally. For day to day infrastructure, it's better to be a pragmatist.

# Exercise: Open the users::group recipe in your editor



**OPEN IN EDITOR:** cookbooks/users/recipes/groups.rb

```
search( :groups, "*:*" ).each do |group_data|
  group group_data['id'] do
    gid group_data['gid']
    members group_data['members']
  end
end
```

**SAVE FILE!**

- This file follows the same pattern as the default users recipe
- Use the Chef Docs for information about the **group** resource

# Exercise: Upload the users cookbook

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

# Exercise: Upload the users cookbook

```
$ knife cookbook upload users
```

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

# Exercise: Add the users recipe to your test node's run list

```
{  
  "name": "node1",  
  "chef_environment": "_default",  
  "normal": {  
    "company": "opscode",  
    "pci": {  
      "in_scope": true  
    },  
    "tags": [ ],  
  },  
  "run_list": [  
    "recipe[apache]",  
    "recipe[motd]",  
    "recipe[users]"  
  ]  
}
```

- Add `recipe[users]` to the `run_list`, **save and close**.
  - **Do** add a comma after `recipe[motd]`
  - **Don't** add a comma after `recipe[users]`

Monday, 30 June 14

OLD SLIDE BEFORE 'knife node edit' WAS REMOVED

knife node show [node] (to see without edit)

-Fj to see it in JSON

Make sure everyone has done this. Only about to get worse since we're going to converge.

# Exercise: Add the users recipe to your test node's run list

```
node1:  
  run_list:  
    - recipe[apache]  
    - recipe[motd]  
    - recipe[users]
```

Monday, 30 June 14

knife node show [node] (to see without edit)  
-Fj to see it in JSON

Make sure everyone has done this. Only about to get worse since we're going to converge.

# Exercise: Add the users recipe to your test node's run list

```
$ knife node run_list add node1 'recipe[users]'
```

```
node1:  
  run_list:  
    - recipe[apache]  
    - recipe[motd]  
    - recipe[users]
```

Monday, 30 June 14

knife node show [node] (to see without edit)  
-Fj to see it in JSON

Make sure everyone has done this. Only about to get worse since we're going to converge.

# Exercise: Re-run the Chef Client

```
* template[/etc/apache2/sites-available/bears] action create (up to date)
* execute[a2ensite bears] action run (skipped due to not_if)
* directory[/srv/apache/bears] action create (up to date)
* template[/srv/apache/bears/index.html] action create (up to date)
Recipe: motd::default
* template[/etc/motd.tail] action create (up to date)
Recipe: users::default
* user[frank] action create
- create user user[frank]

* user[bobo] action create
- create user user[bobo]
```

Monday, 30 June 14

This is the old slide for 11.6.2 – leaving in (but hidden) for now in case there's an issue with the following slide and need to backtrack....

If you get errors, a common cause is that arguments to adduser are not valid – fat-fingered shells, for example.

This is another chance to talk about how knowledge of the system you are automating is vital to success – it's easy to debug that error if you know how adduser behaves.

# Exercise: Re-run the Chef Client

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

```
* template[/etc/apache2/sites-available/bears] action create (up to date)
* execute[a2ensite bears] action run (skipped due to not_if)
* directory[/srv/apache/bears] action create (up to date)
* template[/srv/apache/bears/index.html] action create (up to date)
Recipe: motd::default
* template[/etc/motd.tail] action create (up to date)
Recipe: users::default
* user[frank] action create
- create user user[frank]

* user[bobo] action create
- create user user[bobo]
```

Monday, 30 June 14

This is the old slide for 11.6.2 – leaving in (but hidden) for now in case there's an issue with the following slide and need to backtrack....

If you get errors, a common cause is that arguments to adduser are not valid – fat-fingered shells, for example.

This is another chance to talk about how knowledge of the system you are automating is vital to success – it's easy to debug that error if you know how adduser behaves.

# Exercise: Re-run the Chef Client

```
* directory[/srv/apache/ponies] action create[2013-11-06T11:05:13+00:00] INFO: Processing directory[/srv/apache/ponies] action create (apache::default line 57)
  (up to date)
    * template[/srv/apache/ponies/index.html] action create[2013-11-06T11:05:13+00:00] INFO: Processing template[/srv/apache/ponies/index.html] action create (apache::default line 62)
      (up to date)
Recipe: motd::default
  * template[/etc/motd.tail] action create[2013-11-06T11:05:13+00:00] INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)
    (up to date)
Recipe: users::default
  * user[frank] action create[2013-11-06T11:05:13+00:00] INFO: Processing user[frank] action create (users::default line 10)
[2013-11-06T11:05:13+00:00] INFO: user[frank] created
  - create user user[frank]

  * user[bobo] action create[2013-11-06T11:05:13+00:00] INFO: Processing user[bobo] action create (users::default line 10)
[2013-11-06T11:05:13+00:00] INFO: user[bobo] created
  - create user user[bobo]
```

Monday, 30 June 14

If you get errors, a common cause is that arguments to adduser are not valid – fat-fingered shells, for example.

This is another chance to talk about how knowledge of the system you are automating is vital to success – it's easy to debug that error if you know how adduser behaves.

# Exercise: Re-run the Chef Client

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

```
* directory[/srv/apache/ponies] action create[2013-11-06T11:05:13+00:00] INFO: Processing directory[/srv/apache/ponies] action create (apache::default line 57)
  (up to date)
    * template[/srv/apache/ponies/index.html] action create[2013-11-06T11:05:13+00:00] INFO: Processing template[/srv/apache/ponies/index.html] action create (apache::default line 62)
      (up to date)
Recipe: motd::default
  * template[/etc/motd.tail] action create[2013-11-06T11:05:13+00:00] INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)
    (up to date)
Recipe: users::default
  * user[frank] action create[2013-11-06T11:05:13+00:00] INFO: Processing user[frank] action create (users::default line 10)
[2013-11-06T11:05:13+00:00] INFO: user[frank] created
  - create user user[frank]

  * user[bobo] action create[2013-11-06T11:05:13+00:00] INFO: Processing user[bobo] action create (users::default line 10)
[2013-11-06T11:05:13+00:00] INFO: user[bobo] created
  - create user user[bobo]
```

Monday, 30 June 14

If you get errors, a common cause is that arguments to adduser are not valid – fat-fingered shells, for example.

This is another chance to talk about how knowledge of the system you are automating is vital to success – it's easy to debug that error if you know how adduser behaves.

# Exercise: Verify the users and groups exist

```
opscode@node1:~$ cat /etc/passwd
```

```
frank:x:2001:0:Frank Belson:/home/frank:/bin/bash
bobo:x:2000:0:Bobo T. Clown:/home/bobo:/bin/bash
```

```
opscode@node1:~$ cat /etc/group
```

```
clowns:x:3000:bobo,frank
```

Monday, 30 June 14

We did **not** create home directories for these users, because it's not the default adduser behavior.

The docs can show you how to get it done (manage\_home)

# Let's review real quick..

- We just created a centralized user and group repository, from scratch
  - (That's kind of like what LDAP and Active Directory do, only they are, well, fancier.)
- Between Data Bags and Node Attribute precedence, Chef provides a plethora of ways to inform the patterns you use to configure your infrastructure

Monday, 30 June 14

What happened here is epic. This doesn't do all the same things as AD. But we've done something fairly close using ~40 lines of ruby and JSON. The idea is that primitives driven by data are very powerful. All you need is an abstract notion of data about stuff you're managing, then apply the same pattern. This is the stuff!

# Exercise: Create home directories

- Why do the users not have home directories?
- Use Chef to give them home directories

Monday, 30 June 14

This is currently a bug in Chef 11.6 -- does not work (user does not get modified)

# Review Questions

- What are Data Bags?
- How are they used?
- What does the User resource do?
- What is `include_recipe`, and why is it useful?
- How does search work inside a recipe?
- What other applications do you see for search?
- How could we have used Data Bags in the refactored Apache recipe?
- Where would you go to find out more?

# Roles

Role-based Attributes and Merge Order Precedence

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Explain what Roles are, and how they are used to provide clarity
  - Discuss the Role Ruby DSL
  - Show a Role with Knife
  - Merge order affects the precedence hierarchy
  - Describe nested Roles

# What is a Role?

- So far, we've been just adding recipes directly to a single node
- But that's not how your infrastructure works - think about how you refer to servers
  - "It's a **web server**"
  - "It's a **database server**"
  - "It's a **monitoring server**"

# What is a Role?

- Roles allow you to conveniently encapsulate the run lists and attributes required for a server to "be" what you already think it is
- In practice, Roles make it **easy to configure many nodes identically** without repeating yourself each time

# Best Practice: Roles live in your chef-repo

- Like Data Bags, you have options with how to create a Role
- The best practice is that all of your Roles live in the roles directory of your chef-repo
- They can be created via the API and Knife, but it's nice to be able to see them evolve in your source control history

Monday, 30 June 14

Source control history can act as a documentation trail. Helpful if you need to unentangle your infrastructure.

# Exercise: Create the webserver role



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

- A Role has a:
  - name
  - description
  - run\_list

```
name "webserver"
description "Web Server"
run_list "recipe[apache]"
default_attributes( {
  "apache" => {
    "sites" => {
      "admin" => {
        "port" => 82
      }
    }
  }
})
```

**SAVE FILE!**

# Exercise: Create the webserver role



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

- You can set default node attributes within a role.

```
name "webserver"
description "Web Server"
run_list "recipe[apache]"
default_attributes( {
  "apache" => {
    "sites" => {
      "admin" => {
        "port" => 82
      }
    }
  }
})
```

**SAVE FILE!**

# Exercise: Create the role

Updated Role webserver!

# Exercise: Create the role

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

Updated Role webserver!

# Exercise: Show the role with knife

```
chef_type:          role
default_attributes:
  apache:
    sites:
      admin:
        port: 82
  description:      Web Server
  env_run_lists:
  json_class:       Chef::Role
  name:             webserver
  override_attributes:
  run_list:
    recipe[apache]
```

# Exercise: Show the role with knife

```
$ knife role show webserver
```

```
chef_type:          role
default_attributes:
  apache:
    sites:
      admin:
        port:  82
  description:       Web Server
  env_run_lists:
  json_class:        Chef::Role
  name:              webserver
  override_attributes:
  run_list:
    recipe[apache]
```

# Exercise: Search for roles with recipe[apache] in their run list

```
1 items found

chef_type:          role
default_attributes:
  apache:
    sites:
      admin:
        port: 82
  description:      Web Server
  env_run_lists:
  json_class:       Chef::Role
  name:             webserver
  override_attributes:
  run_list:
    recipe[apache]
```

Monday, 30 June 14

This is a hard one to do without the key above, but it's worth it. Make them try and figure it out from the docs's "search" page and the hint that it is "solr syntax".

Talk about the escaping of square brackets when searching on a run list – this applies to nodes too!

We could also search for port:82, and see what roles set a key named port to the value 82  
knife search role "port:82"

# Exercise: Search for roles with recipe[apache] in their run list

```
$ knife search role "run_list:recipe\[apache\]"
```

```
1 items found

chef_type:          role
default_attributes:
  apache:
    sites:
      admin:
        port: 82
  description:      Web Server
  env_run_lists:
  json_class:       Chef::Role
  name:             webserver
  override_attributes:
  run_list:
    recipe[apache]
```

Monday, 30 June 14

This is a hard one to do without the key above, but it's worth it. Make them try and figure it out from the docs's "search" page and the hint that it is "solr syntax".

Talk about the escaping of square brackets when searching on a run list – this applies to nodes too!

We could also search for port:82, and see what roles set a key named port to the value 82  
knife search role "port:82"

# Exercise: Replace recipe[apache] with role[webserver] in run list

```
{  
  "name": "node1",  
  "chef_environment": "_default",  
  "normal": {  
    "company": "opscode",  
    "pci": {  
      "in_scope": true  
    },  
    "tags": [  
    ]  
  },  
  "run_list": [  
    "role[webserver]",  
    "recipe[motd]",  
    "recipe[users]"  
  ]  
}
```

- Replace `recipe[ apache ]` with `role[ webserver ]`, **save** and **close**.

# Exercise: Replace `recipe[apache]` with `role[webserver]` in run list

```
node1:  
  run_list:  
    - role[webserver]  
    - recipe[motd]  
    - recipe[users]
```

Monday, 30 June 14

There are some problems with these commands in Windows environment, so using the UI (next slide) instead – Trello issue here <http://goo.gl/bh8NSb>

## Exercise: Replace recipe[apache] with role[webserver] in run list

```
$ knife node run_list add node1 "role[webserver]" -a "recipe[apache]"
$ knife node run_list remove node1 "recipe[apache]"
```

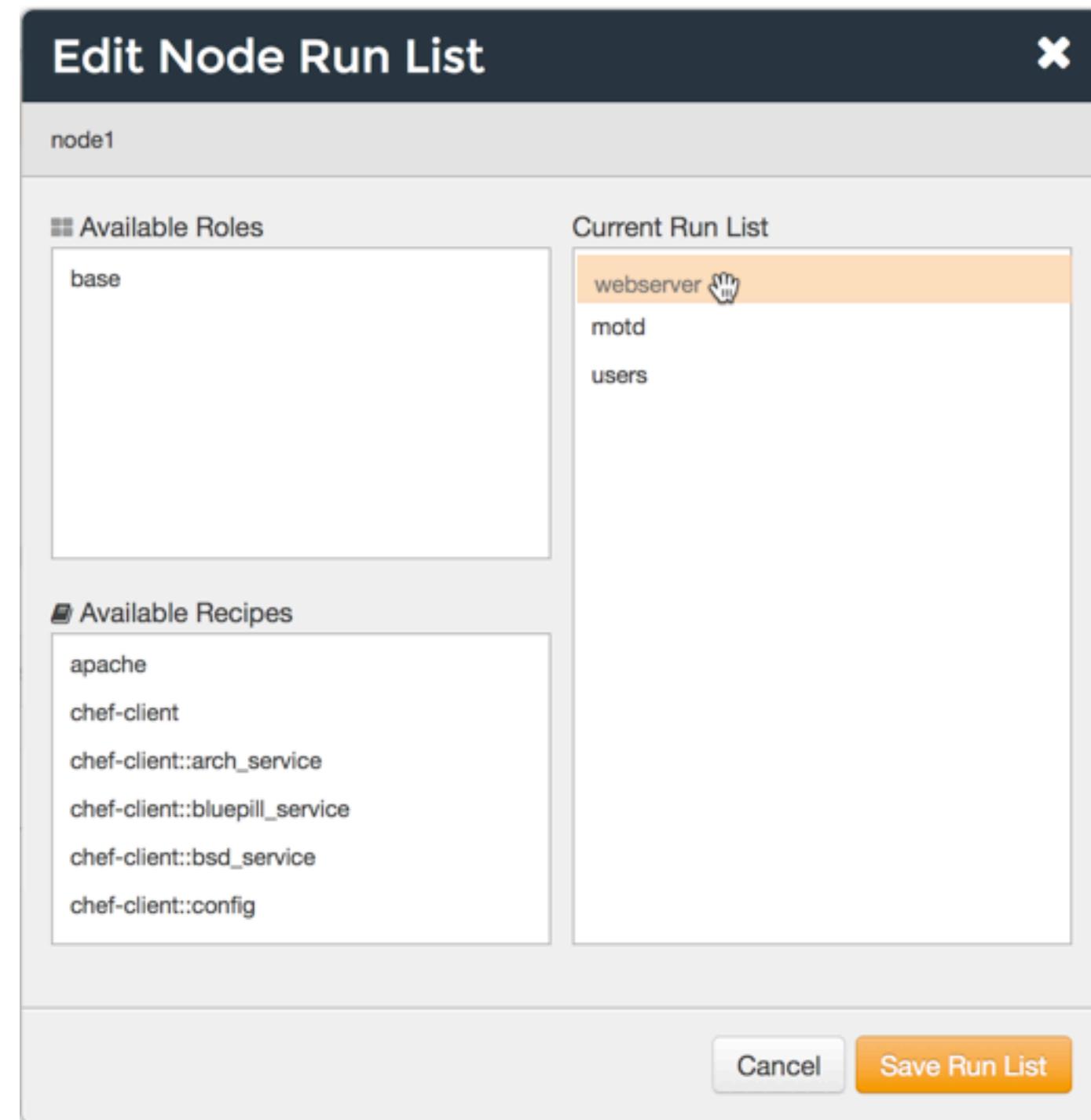
```
node1:
  run_list:
    role[webserver]
    recipe[motd]
    recipe[users]
```

Monday, 30 June 14

There are some problems with these commands in Windows environment, so using the UI (next slide) instead – Trello issue here <http://goo.gl/bh8NSb>

# Exercise: Replace recipe[apache] with role[webserver] in run list

- Click the ‘Nodes’ tab then select node ‘node1’
- Click ‘Edit Run List’ from left navigation bar
- Drag ‘Apache’ over from ‘Current Run List’ to ‘Available Recipes’
- Drag ‘webserver’ over from ‘Available Roles’ to the top of ‘Current Run List’
- Click ‘Save Run List’



Monday, 30 June 14

Here will use the UI to add the Role ‘Webserver’ to the top of the Run List. (Real reason, we dont want to show ‘knife node edit’, and there is an issue using ‘knife node run\_list add node1 ‘role[webserver]’ –a ‘recipe[apache]’ in Windows)

Click the Node tab then select node ‘node1’.

Note (30/10/13): This only works for the ‘\_default’ environment. Using teh others I get "Coming soon"

Image shows ‘target3’, but this will need changed anyhow when we rename our node ‘node1’

# Exercise: Re-run the Chef Client

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

```
INFO: *** Chef 11.8.0 ***
INFO: Chef-client pid: 5634
INFO: Run List is [role[webserver], recipe[motd], recipe[users]]
INFO: Run List expands to [apache, motd, users]
```

Monday, 30 June 14

Talk about run list expansion – we looked at the role, grabbed it's runlist, and stuck it in place.

We are coming back to run list expansion when we finish the base role

# Exercise: Re-run the Chef Client

```
INFO: Processing template[/etc/apache2/sites-available/clowns] action create (apache::default line 28)
INFO: Processing execute[a2ensite clowns] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/clowns] action create (apache::default line 45)
INFO: Processing template[/srv/apache/clowns/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/apache2/sites-available/bears] action create (apache::default line 28)
INFO: Processing execute[a2ensite bears] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/bears] action create (apache::default line 45)
INFO: Processing template[/srv/apache/bears/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/apache2/sites-available/admin] action create (apache::default line 28)
INFO: template[/etc/apache2/sites-available/admin] updated content
INFO: template[/etc/apache2/sites-available/admin] mode changed to 644
INFO: Processing execute[a2ensite admin] action run (apache::default line 38)
Enabling site admin.

To activate the new configuration, you need to run:
  service apache2 reload
INFO: execute[a2ensite admin] ran successfully
INFO: execute[a2ensite admin] not queuing delayed action restart on service[apache2] (delayed), as
it's already been queued
INFO: Processing directory[/srv/apache/admin] action create (apache::default line 45)
INFO: directory[/srv/apache/admin] created directory /srv/apache/admin
INFO: directory[/srv/apache/admin] mode changed to 755
INFO: Processing template[/srv/apache/admin/index.html] action create (apache::default line 50)
INFO: template[/srv/apache/admin/index.html] updated content
INFO: template[/srv/apache/admin/index.html] mode changed to 644
```

Monday, 30 June 14

We added the admin site to the existing set

# Node Attributes that are hashes are merged

- The apache cookbooks attribute file contains:

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

- While our role has...

```
default_attributes( {
  "apache" => {
    "sites" => {
      "admin" => {
        "port" => 82
      }
    }
  }
})
```

Monday, 30 June 14

And the results are merged together.

We will get to merge order in just a few more slides...

## Exercise: Display the apache.sites attribute on all nodes with webserver role

```
1 items found
```

```
node1:  
  apache.sites:  
    admin:  
      port: 82  
    bears:  
      port: 81  
    clowns:  
      port: 80
```

Monday, 30 June 14

This is the first time we use the "dot" notation with -a in knife – talk about using it to show deeply nested attributes easily.

It is also in contrast to how you **search** for that field, which is to join on an `_`. The inconsistency is lame, but it is because `.` is a stopword in the search query parser, and the index itself. So we used `_` in the search, and when I did knife's attribute lookup, I switched it to a `.` without thinking about it. Sad but true.

Talk about the 'role' field, and how commonly used in search it is

## Exercise: Display the apache.sites attribute on all nodes with webserver role

```
$ knife search node "role:webserver" -a apache.sites
```

```
1 items found
```

```
node1:  
  apache.sites:  
    admin:  
      port: 82  
    bears:  
      port: 81  
    clowns:  
      port: 80
```

Monday, 30 June 14

This is the first time we use the "dot" notation with -a in knife – talk about using it to show deeply nested attributes easily.

It is also in contrast to how you **search** for that field, which is to join on an `_`. The inconsistency is lame, but it is because `.` is a stopword in the search query parser, and the index itself. So we used `_` in the search, and when I did knife's attribute lookup, I switched it to a `.` without thinking about it. Sad but true.

Talk about the 'role' field, and how commonly used in search it is

# Exercise: Edit the webserver role



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

- Do not forget the **comma** after the admin site
- Change the value of the bears site to be 8081

```
default_attributes( {
  "apache" => {
    "sites" => {
      "admin" => {
        "port" => 82
      },
      "bears" => {
        "port" => 8081
      }
    }
  }
})
```

**SAVE FILE!**

Monday, 30 June 14

We are going to show that merge order has replacement for values that appear in two places...

# Exercise: Create the role

Updated Role webserver!

# Exercise: Create the role

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

Updated Role webserver!

# Exercise: Re-run the Chef Client

```
[2012-10-23T03:28:34+00:00] INFO: Processing template[/etc/apache2/sites-available/bears]
action create (apache::default line 28)
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] backed up to /
var/chef/backup/etc/apache2/sites-available/bears.chef-20121023032834
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] updated content
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] owner changed to
0
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] group changed to
0
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] mode changed to
644
[2012-10-23T03:28:34+00:00] INFO: Processing execute[a2ensite bears] action run
(apache::default line 38)
[2012-10-23T03:28:34+00:00] INFO: Processing directory[/srv/apache/bears] action create
(apache::default line 45)
```

# Exercise: Re-run the Chef Client

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

```
[2012-10-23T03:28:34+00:00] INFO: Processing template[/etc/apache2/sites-available/bears]
action create (apache::default line 28)
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] backed up to /
var/chef/backup/etc/apache2/sites-available/bears.chef-20121023032834
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] updated content
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] owner changed to
0
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] group changed to
0
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] mode changed to
644
[2012-10-23T03:28:34+00:00] INFO: Processing execute[a2ensite bears] action run
(apache::default line 38)
[2012-10-23T03:28:34+00:00] INFO: Processing directory[/srv/apache/bears] action create
(apache::default line 45)
```

## Exercise: Display the apache sites attribute on all nodes with the webserver role

```
1 items found

node1:
  apache.sites:
    admin:
      port: 82
    bears:
      port: 8081
    clowns:
      port: 80
```

## Exercise: Display the apache sites attribute on all nodes with the webserver role

```
$ knife search node 'role:webserver' -a apache.sites
```

```
1 items found
```

```
node1:  
  apache.sites:  
    admin:  
      port: 82  
    bears:  
      port: 8081  
    clowns:  
      port: 80
```

**When you combine merge  
order and precedence  
rules, you get this:**

# Merge Order and Precedence

	Attribute Files	Node / Recipe	Environment	Role
default	1	2	3	4
force_default	5	6		
normal	7	8		
override	9	10	12	11
force_override	13	14		
automatic			15	

Monday, 30 June 14

And of course, automatic attributes go to 11 (spinal tap joke, folks)!

The 9 is not a typo – environment overrides bind higher than roles at override, but not at default. This is why, in the next section, we recommend that environment overrides are what you should use most often at that level.

Also, this shows why we say you should use defaults all the time! Merge order does the right thing for you, and leaves you so many levels of precedence to work with if you get stuck.

Take away => use defaults all the time. You will know you're in the situation you need overrides when you're in it.

# Best Practice: Roles get default attributes

- While it is awesome that you can use overrides, in practice there is little need
- If you always set **default** node attributes in your cookbook attribute files
- You can almost **always** set default node attributes in your role, and let merge order do the rest

# Best Practice: Have "base" roles

- In addition to obvious roles, such as "webserver", it is a common practice to group any functionality that "goes together" in a role
- The most common example here is a **base** role, where you include all the recipes that should be run on every node

# Exercise: Create the base role



**OPEN IN EDITOR:**

```
name "base"
description "Base Server Role"
run_list "recipe[motd]", "recipe[users]"
```

**SAVE FILE!**

Monday, 30 June 14

Note to students: Don't run ahead of this step and remove these out of your node definition. We will make use of those attributes again shortly.

# Exercise: Create the base role



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

```
name "base"
description "Base Server Role"
run_list "recipe[motd]", "recipe[users]"
```

**SAVE FILE!**

Monday, 30 June 14

Note to students: Don't run ahead of this step and remove these out of your node definition. We will make use of those attributes again shortly.

# Exercise: Create the role

Updated Role base!

# Exercise: Create the role

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

Updated Role base!

# Exercise: Add the base role to the webserver role's run list



**OPEN IN EDITOR:**

```
name "webserver"
description "Web Server"
run_list "role[base]", "recipe[apache]"
default_attributes( {
  "apache" => {
```

- Put `role[base]` at the front of the `run_list`

**SAVE FILE!**

Monday, 30 June 14

Again, make sure they **do not edit the node** and remove the motd and users recipes from the node object -we need that for our next object lesson

# Exercise: Add the base role to the webserver role's run list



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

```
name "webserver"
description "Web Server"
run_list "role[base]", "recipe[apache]"
default_attributes(
  "apache" => {
```

- Put `role[base]` at the front of the `run_list`

**SAVE FILE!**

Monday, 30 June 14

Again, make sure they **do not edit the node** and remove the motd and users recipes from the node object -we need that for our next object lesson

# Exercise: Update the role

Updated Role webserver!

# Exercise: Update the role

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

Updated Role webserver!

# Exercise: Re-run the Chef Client

```
INFO: *** Chef 11.8.0 ***
Chef-client pid: 5634
INFO: Run List is [role[webserver], recipe[motd], recipe[users]]
INFO: Run List expands to [motd, users, apache]
```

Monday, 30 June 14

Talk about run-list expansion – even though motd and users are explicitly after role[webserver] in the list, due to our nested role in the webserver, we are going to execute them first, because they expanded from the role[webserver].

also, you can drive home that chef only executes recipes the first time through, so it is fine to just be as explicit as you want.

# Exercise: Re-run the Chef Client

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

```
INFO: *** Chef 11.8.0 ***
Chef-client pid: 5634
INFO: Run List is [role[webserver], recipe[motd], recipe[users]]
INFO: Run List expands to [motd, users, apache]
```

Monday, 30 June 14

Talk about run-list expansion – even though motd and users are explicitly after role[webserver] in the list, due to our nested role in the webserver, we are going to execute them first, because they expanded from the role[webserver].

also, you can drive home that chef only executes recipes the first time through, so it is fine to just be as explicit as you want.

# Best Practice: Be explicit about what you need or expect

- Chef will only execute a recipe the first time it appears in the run list
- So be explicit about your needs and expectations - either by nesting roles or using include\_recipe

# Exercise: Set the run list to just role[webserver]

- Remove all the entries in the run list other than role[webserver]

Monday, 30 June 14

This one has no example! Oh noes!

Instead of thinking about the world as a dependency tree, roles allow you to think about the world as arrays of functionality.

# Review Questions

- What is a Role?
- What makes for a "good" role?
- How do you search for roles with a given recipe in their run list?
- How many times will Chef execute a recipe in the same run?

# Environments

Cookbook Version Constraints and Override Attributes

v1.2.3



Monday, 30 June 14

# Lesson Objectives

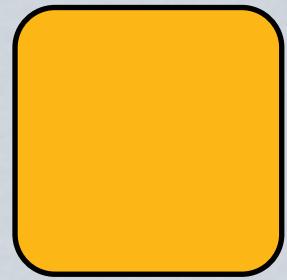
- After completing the lesson, you will be able to
  - Describe what an Environment is, and how it is different from an Organization
  - Set cookbook version constraints
  - Explain when to set attributes in an environment

Monday, 30 June 14

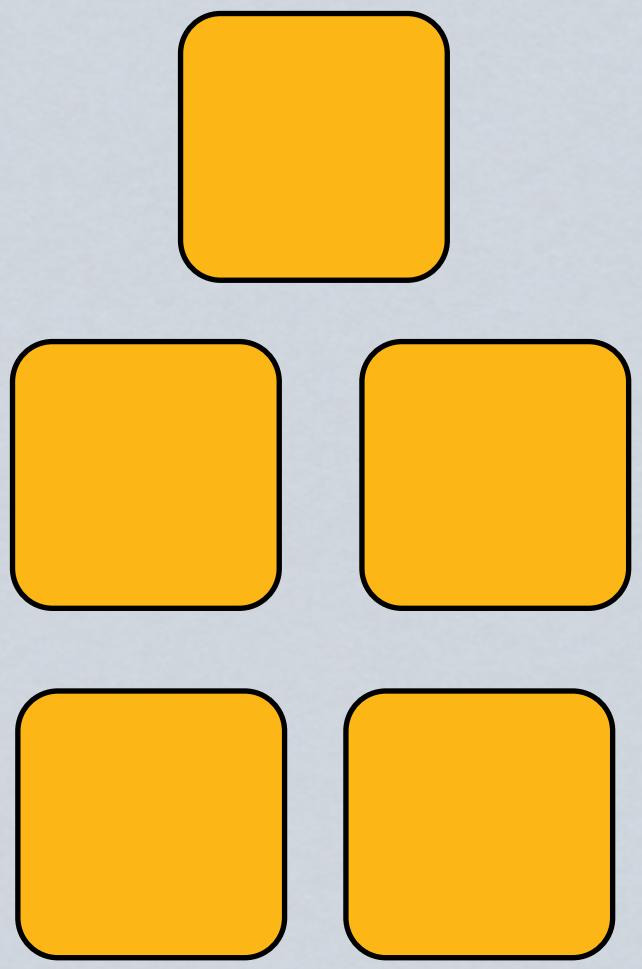
Note: when this module is finished, we will have learned every core primitive

# Environments

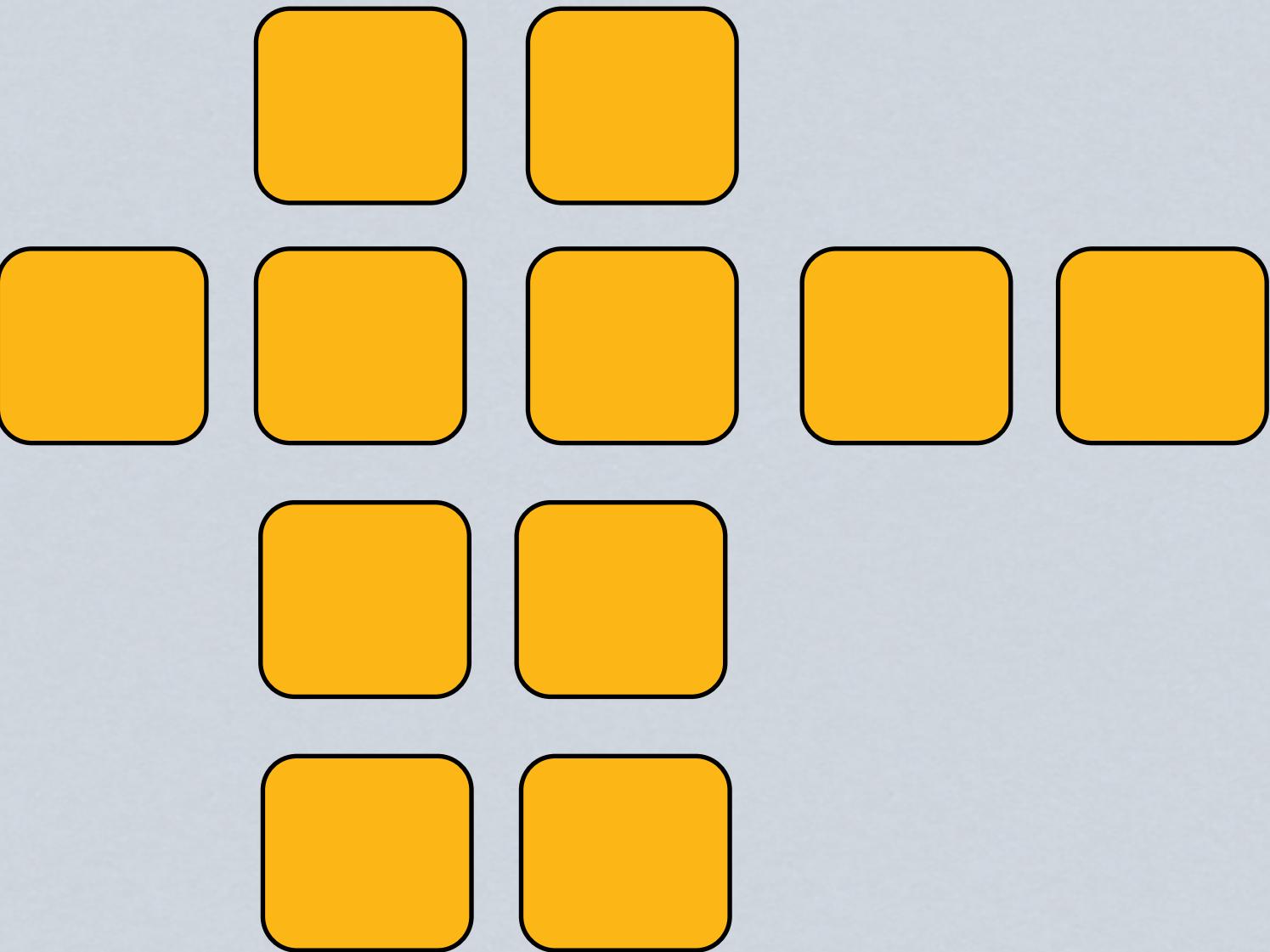
Development



Staging



Production



Organization

# Environments

- Every Organization starts with a single environment
- Environments reflect your patterns and workflow
  - Development
  - Test
  - Staging
  - Production
  - etc.

# Environments Define Policy

- Each environment may include attributes necessary for configuring the infrastructure in that environment
  - Production needs certain Yum repos
  - QA needs different Yum repos
  - The version of the Chef cookbooks to be used

# Environment Best Practice

- Best Practice: If you need to share cookbooks or roles, you likely want an Environment rather than an organization
- Environments allow for isolating resources within a single organization

## Exercise: Use knife to show the available cookbook versions

```
$ knife cookbook show apache
```

```
apache      0.2.0      0.1.0
```

Monday, 30 June 14

Remember the version bump from earlier? This was why.

# Exercise: List current environments

```
_default
```

- The `_default` environment is read-only, and sets no policy at all

Monday, 30 June 14

Every organization starts out with a single environment that is named `_default`, which ensures that at least one environment is always available to the server. The `_default` environment cannot be modified in any way. Nodes, roles, run-lists, cookbooks (and cookbook versions), and attributes specific to an organization can only be associated with a custom environment.

# Exercise: List current environments

```
$ knife environment list
```

```
_default
```

- The `_default` environment is read-only, and sets no policy at all

Monday, 30 June 14

Every organization starts out with a single environment that is named `_default`, which ensures that at least one environment is always available to the server. The `_default` environment cannot be modified in any way. Nodes, roles, run-lists, cookbooks (and cookbook versions), and attributes specific to an organization can only be associated with a custom environment.

# Make an environments directory

```
$ mkdir environments
```

(No output)

# Exercise: Create a dev environment



**OPEN IN EDITOR:** environments/dev.rb

```
name "dev"  
description "For developers!"  
cookbook "apache", "= 0.2.0"
```

**SAVE FILE!**

- Environments have **names**
- Environments have a **description**
- Environments *can* have one or more **cookbook** constraints

Monday, 30 June 14

Again, these can be defined either via Ruby or JSON. In a minute, we'll see one of the benefits of defining via Ruby

# Cookbook Version Constraints

- = Equal to
- There are other options but equality is the recommended practice.
- Learn more at [http://docs.opscode.com/chef/essentials\\_cookbook\\_versions.html](http://docs.opscode.com/chef/essentials_cookbook_versions.html)

Monday, 30 June 14

Specifying "`>= 2.6.5`" is an optimistic version constraint. All versions greater than the one specified, including major releases (e.g. 3.0.0) are allowed.

Conversely, specifying "`~> 2.6`" is pessimistic about future major revisions and "`~> 2.6.5`" is pessimistic about future minor revisions.

`"~> 2.6"` matches cookbooks `>= 2.6.0 AND < 3.0.0`

`"~> 2.6.5"` matches cookbooks `>= 2.6.5 AND < 2.7.0`

# Best Practice: Use only the = operator

- When you need more complex version constraints, you'll be glad they are there
- But do not be tempted to build complicated schemes tying cookbook development branches to complex version constraints
  - That way leads madness :)

Monday, 30 June 14

The moment you need more operators is not common. Reality is, dev branches fork off, some die, environments never entirely consolidate. It's easy to lose track and manage a ton of complexity.

Note: any slide bullet point that has smiley means it must be TRUE! :)

# Exercise: Create the dev environment

Updated Environment dev

# Exercise: Create the dev environment

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

```
Updated Environment dev
```

# Exercise: Show your dev environment

```
chef_type:          environment
cookbook_versions:
  apache:  = 0.2.0
default_attributes:
description:       For developers!
json_class:        Chef::Environment
name:              dev
override_attributes:
```

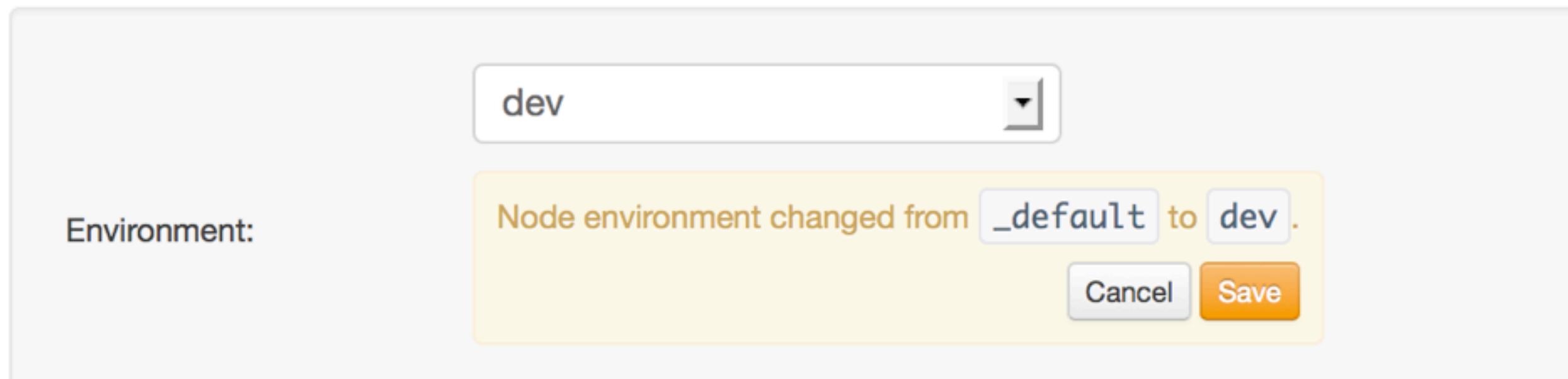
# Exercise: Show your dev environment

```
$ knife environment show dev
```

```
chef_type:          environment
cookbook_versions:
  apache:  = 0.2.0
default_attributes:
description:       For developers!
json_class:        Chef::Environment
name:              dev
override_attributes:
```

# Exercise: Change your node's environment to "dev"

- Click the ‘Nodes’ tab then select node ‘node1’
- Select dev from the ‘Environments’ drop-down list
- Click ‘Save’



Monday, 30 June 14

Click the Node tab then select node ‘node1’

Might look at knife filp plugin – <https://github.com/jonlives/knife-fip>

# Exercise: Re-run the Chef Client

```
INFO: Chef Run complete in 1.587776095 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

Monday, 30 June 14

We removed PCI data, so we did modify the MOTD

# Exercise: Re-run the Chef Client

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

```
INFO: Chef Run complete in 1.587776095 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

Monday, 30 June 14

We removed PCI data, so we did modify the MOTD

# Exercise: Create a production environment



**OPEN IN EDITOR:** environments/production.rb

- Make sure the apache cookbook is set to version 0.1.0
- Set an override attribute for being in scope - no matter what, you are in scope

```
name "production"
description "For Prods!"
cookbook "apache", "= 0.1.0"
override_attributes( {
  "pci" => {
    "in_scope" => true
  }
})
```

**SAVE FILE!**

# Exercise: Create the production environment

Updated Environment production

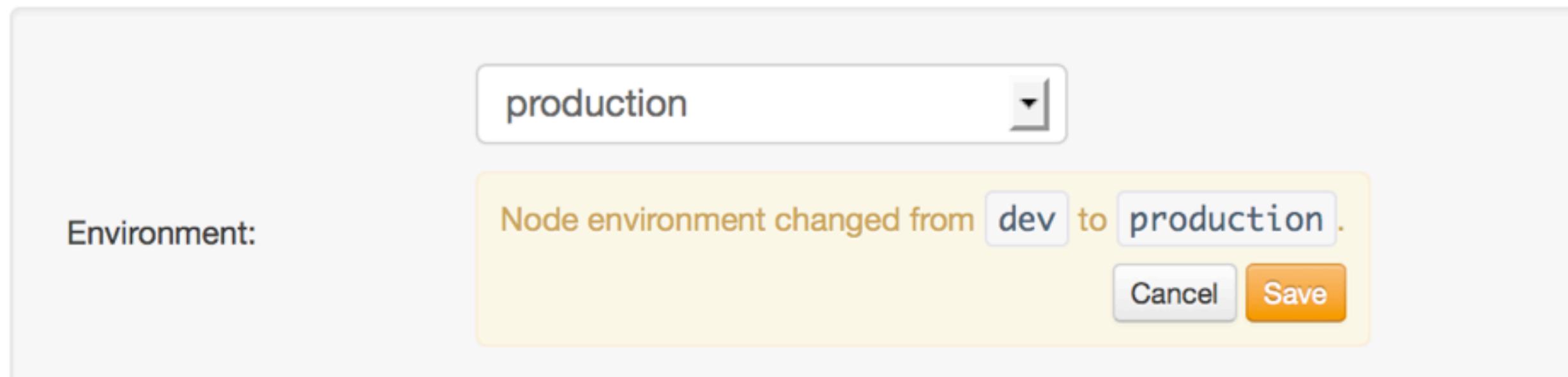
# Exercise: Create the production environment

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

Updated Environment production

# Exercise: Change your node's environment to "production"

- Click the 'Nodes' tab then select node 'node1'
- Select production from the 'Environments' drop-down list
- Click 'Save'



Monday, 30 June 14

Click the Node tab then select node 'node1'

# Exercise: Re-run the Chef Client

```
INFO: Loading cookbooks [apache, motd, pci, users]
Synchronizing Cookbooks:
...
Recipe: motd::default
  * template[/etc/motd.tail] action create
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)

  * cookbook_file[/var/www/index.html] action create
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 17)
...
INFO: Chef Run complete in 1.903297305 seconds
INFO: Removing cookbooks/apache/templates/default/custom.erb from the cache; it is no longer needed by chef-client.
INFO: Removing cookbooks/apache/templates/default/index.html.erb from the cache; it is no longer needed by chef-client.
INFO: Removing cookbooks/apache/attributes/default.rb from the cache; it is no longer needed by chef-client.
INFO: Running report handlers
INFO: Report handlers complete
Chef Client finished, 2 resources updated
```

Monday, 30 June 14

We went back in time! Whoa!

Point out that we downloaded the old files, we removed the ones that we had that were newer, and we applied the old policy

If students have cookbook errors, the last uploaded cookbook at the time of the version roll probably had syntax errors in it.

# Exercise: Re-run the Chef Client

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

```
INFO: Loading cookbooks [apache, motd, pci, users]
Synchronizing Cookbooks:
...
Recipe: motd::default
  * template[/etc/motd.tail] action create
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)

  * cookbook_file[/var/www/index.html] action create
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 17)
...
INFO: Chef Run complete in 1.903297305 seconds
INFO: Removing cookbooks/apache/templates/default/custom.erb from the cache; it is no longer needed by chef-client.
INFO: Removing cookbooks/apache/templates/default/index.html.erb from the cache; it is no longer needed by chef-client.
INFO: Removing cookbooks/apache/attributes/default.rb from the cache; it is no longer needed by chef-client.
INFO: Running report handlers
INFO: Report handlers complete
Chef Client finished, 2 resources updated
```

Monday, 30 June 14

We went back in time! Whoa!

Point out that we downloaded the old files, we removed the ones that we had that were newer, and we applied the old policy

If students have cookbook errors, the last uploaded cookbook at the time of the version roll probably had syntax errors in it.

# Rollbacks and Desired State Best Practice

- Chef is not magic - it manages state for **declared** resources
- We just rolled back to an earlier version of the apache cookbook
- While the recipe applied fine, investigating the system will reveal Apache is still configured as it was in the 0.2.0 cookbook
- A better way to ensure a smooth rollback: write contra-resources to clean up, and have a new version of the cookbook.

# Review Questions

- What is an Environment?
- How is it different from an Organization?
- What kind of node attributes do you typically set from an Environment?

Monday, 30 June 14

We need to add:

- \* Showing how to use `chef_environment` in a recipe
- \* How to write data-driven recipes that can do contra-state

# Using Community Cookbooks

Open Source: Saving you time!

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Use the Opscode Chef Community site to find, preview and download cookbooks
  - Use knife to work with the Community Site API
  - Download, extract, examine and implement cookbooks from the Community site

# The easy way...

- We've been writing some cookbooks so far...
- Hundreds already exist for a large number of use cases and purposes. Many (but only a fraction) are maintained by Opscode.
- Think of it like RubyGems.org, CPAN.org, or other focused plugin-style distribution sites.

# Exercise: Find and preview cookbooks on the community site

The screenshot shows the Opscode Community website. At the top, there's a navigation bar with links for 'Community Login', 'Sign Up', 'Recover Password', social media icons for Facebook and Twitter, and a 'Reader' button. Below the navigation is the Opscode logo and a search bar with 'Search Cookbooks' and 'Advanced Search' options.

The main content area features a large 'Download & Install Chef' section with a 'Chef' logo and a large orange arrow pointing down. To its right is a list of links: 'Quick Start', 'Other ways to install Chef', 'Learn the basics of Chef', 'Learn about Chef's Architecture', and 'Plugins for Chef, Knife, and Ohai'. Below this are four tabs: 'Download & Install Chef' (selected), 'How to Contribute Code', 'Chef Documentation', and 'Ways to Get Help'.

On the left, under 'Chef Cookbooks', there's a table of popular cookbooks with their names, ratings (represented by stars), and download counts:

Cookbook	Rating	Downloads
mysql	★★★★★	14575
apache2	★★★★★	11704
nginx	★★★★★	10061
apt	★★★★★	8061
application	★★★★★	7355
java	★★★★★	7099

A link 'See all cookbooks' is also present. On the right side, there are two boxes: 'Blog Posts' which links to a blog post about a webinar, and 'Opscode Events' which lists '2 Day Chef Fundamentals - Boston' and '2 Day Chef Fundamentals - Austin'.

Monday, 30 June 14

Let's go look around on the community site. This is the main portal, and more features will come, explore on your own after class.

\*click\* – go to the cookbooks section.

# Exercise: Find and preview cookbooks on the community site

The screenshot shows a web browser window for the Opscode Community site at [community.opscode.com](http://community.opscode.com). The page features a dark header with the Opscode logo and navigation links for Community Login, Sign Up, Recover Password, Facebook, Twitter, and Reader. A large button labeled "Cookbooks!" is highlighted with a dashed box and an arrow pointing to it. Below the header, there's a search bar for "Search Cookbooks" and an Advanced Search link. The main content area includes sections for "Download & Install Chef" (with a Chef logo graphic), "Chef Cookbooks" (listing mysql, apache2, nginx, apt, application, and java with their download counts and star ratings), "Helpful Links" (including Plugins for Chef, Knife, and Ohai, Source code on github, Opscode's bug tracker, #chef on irc.freenode.net, Presentations on Chef, and Community MVPs), and "Blog Posts" (with a summary of a recent webinar and a link to the blog). There are also sections for "Opscode Events" (listing 2 Day Chef Fundamentals - Boston and Austin).

Monday, 30 June 14

Let's go look around on the community site. This is the main portal, and more features will come, explore on your own after class.

\*click\* – go to the cookbooks section.

# Exercise: Search for a chef-client cookbook

The screenshot shows the main page of the Opscode Community Cookbooks website. At the top, there's a navigation bar with links for 'Community Login', 'Sign Up', 'Recover Password', social media icons for Facebook and Twitter, and a 'Reader' link. Below the navigation is a search bar with 'Search Cookbooks' and 'Advanced Search' buttons. The main content area is titled 'All Categories' and features a list of popular cookbooks with their download counts and ratings. To the right, there are two callout boxes: one for 'Need Help?' and another for 'REST API'.

**All Categories**

[See all Cookbooks](#)

Category	Cookbook	Downloads
Databases	mysql	14575
	postgresql	5750
	database	3710
Process Management	runit	4352
	god	543
	supervisord	425
Programming Languages	java	7099
	php	4392
	python	3321
Applications	application	
Web Servers	apache2	11704
	nginx	10061
	passenger_apache2	1788
Monitoring & Trending	zabbix	4533
	nagios	4272
	newrelic	3049
Package Management	apt	8061
	yumrepo	3576
	yum	3199
Networking	ntp	

[Add a New Cookbook](#)

**Need Help?**  
If you have questions, or you're stuck, we're here to help.  
[Get Help](#)

**REST API**  
Access all cookbooks on this site programmatically.  
[Learn More](#)

Monday, 30 June 14

The cookbooks main page.

Perform a search using the search box. Simply type in 'chef-client' and hit enter.

# Exercise: Search for a chef-client cookbook

The screenshot shows a web browser window for the Opscode Community Cookbooks page at [community.opscode.com/cookbooks](http://community.opscode.com/cookbooks). A large dashed box highlights the search bar with the placeholder "Search for: chef-client". An arrow points from this box to the search bar. Below the search bar, there's a "Search Cookbooks" button and an "Advanced Search" link. The main content area displays various cookbooks categorized by type, such as Databases, Web Servers, Process Management, Monitoring & Trending, Programming Languages, Package Management, Applications, and Networking. Each category lists several cookbooks with their names, download counts, and star ratings. To the right of the main content, there are two boxes: one for "Need Help?" with a "Get Help" link, and another for "REST API" with a "Learn More" link.

Opscode Community

Community Login | Sign Up | Recover Password f t

Cookbooks | Users | Chat | Opscode.com >

Search for: chef-client

All Categories

See all Cookbooks

Databases

- mysql (14575 downloads)
- apache2 (11704 downloads)
- nginx (10061 downloads)

See all

Process Management

- runit (4352 downloads)
- god (543 downloads)
- supervisord (425 downloads)

See all

Programming Languages

- java (7099 downloads)
- php (4392 downloads)
- python (3321 downloads)

See all

Applications

- application

Web Servers

- apache2 (11704 downloads)
- nginx (10061 downloads)
- passenger\_apache2 (1768 downloads)

See all

Monitoring & Trending

- zabbix (4533 downloads)
- nagios (4272 downloads)
- newrelic (3049 downloads)

See all

Package Management

- apt (8061 downloads)
- yumrepo (3576 downloads)
- yum (3199 downloads)

See all

Networking

- ntp

Add a New Cookbook

Need Help?

If you have questions, or you're stuck, we're here to help.

Get Help

REST API

Access all cookbooks on this site programmatically.

Learn More

Monday, 30 June 14

The cookbooks main page.

Perform a search using the search box. Simply type in 'chef-client' and hit enter.

# Search Results...

The screenshot shows a web browser window titled "Search Results for chef-client – Opscode Community". The URL in the address bar is "community.opscode.com/search?query=chef-client&scope=cookbook". The page features the Opscode logo and navigation links for "Community Login", "Sign Up", "Recover Password", "Cookbooks", "Users", "Chat", and "Opscode.com". A search bar at the top right contains the query "chef-client". Below the search bar, the text "Search results for 'chef-client'" is displayed.

**chef-client** 2469 downloads 84 followers

Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), mac\_os\_x\_server (>= 0.0.0), openbsd (>= 0.0.0), oracle (>= 0.0.0), redhat (>= 0.0.0), suse (>= 0.0.0), ubuntu (>= 0.0.0), windows (>= 0.0.0)

★★★★★ 6 ratings Category: Utilities

Manages client.rb configuration and chef-client service

---

**chef-client\_syslog** 44 downloads 0 followers

★★★★★ 0 ratings Category: Utilities

chef-client log to syslog

---

**chef-client-cron** 100 downloads 0 followers

Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), openbsd (>= 0.0.0), redhat (>= 0.0.0), ubuntu (>= 0.0.0)

★★★★★ 0 ratings Category: Utilities

Manages aspects of only chef-client

---

**forkable\_client** 41 downloads 3 followers

★★★★★ 0 ratings Category: Utilities

Provides forked chef-client runs

Monday, 30 June 14

Here's the results, there are other cookbooks, for purposes of this class we want the first one.

# Search Results...

The screenshot shows a web browser window titled "Search Results for chef-client – Opscode Community". The search bar at the top contains the query "chef-client". Below the search bar, there are links for "Community Login", "Sign Up", "Recover Password", and social media icons for Facebook and Twitter. A navigation bar includes "Cookbooks", "Users", "Chat", and "Opscode.com". The main content area displays search results for "chef-client".

**chef-client** ← **We're probably looking for this one**

Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), mac\_os\_x\_server (>= 0.0.0), openbsd (>= 0.0.0), oracle (>= 0.0.0), redhat (>= 0.0.0), suse (>= 0.0.0), ubuntu (>= 0.0.0), windows (>= 0.0.0)

★★★★★ 6 ratings Category: Utilities

Manages client.rb configuration and chef-client service

69 downloads 84 followers

**chef-client\_syslog**

★★★★★ 0 ratings Category: Utilities

chef-client log to syslog

44 downloads 0 followers

**chef-client-cron**

Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), openbsd (>= 0.0.0), redhat (>= 0.0.0), ubuntu (>= 0.0.0)

★★★★★ 0 ratings Category: Utilities

Manages aspects of only chef-client

100 downloads 0 followers

**forkable\_client**

★★★★★ 0 ratings Category: Utilities

Provides forked chef-client runs

41 downloads 3 followers

Monday, 30 June 14

Here's the results, there are other cookbooks, for purposes of this class we want the first one.

# Viewing a cookbook

The screenshot shows a web browser window displaying the Opscode Community website. The URL in the address bar is [community.opscode.com/cookbooks/chef-client](http://community.opscode.com/cookbooks/chef-client). The page title is "Chef Cookbook: chef-client – Opscode Community". The main content area shows the "chef-client [3.0.4]" cookbook. Key details include:

- Rating: ★★★★☆ 6 ratings
- Downloads: 2469 times
- Last updated: Jun 12, 2013
- Description: Manages client.rb configuration and chef-client service
- Category: Utilities
- Homepage: <http://github.com/opscode-cookbooks/chef-client>
- License: Apache 2.0
- Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), mac\_os\_x\_server (>= 0.0.0), openbsd (>= 0.0.0), oracle (>= 0.0.0), redhat (>= 0.0.0), suse (>= 0.0.0), ubuntu (>= 0.0.0), windows (>= 0.0.0)
- Created: Dec 16, 2010
- Last updated: Jun 12, 2013
- Versions: 3.0.4, 3.0.2, 3.0.0, 2.2.4, 2.2.2, 2.2.0, 2.1.10, 2.1.8, 2.1.6, 2.1.4, 2.1.2, 2.1.0, 2.0.2, 2.0.0, 1.2.0, 1.1.4, 1.1.2, 1.1.0, 1.0.4, 1.0.2, 1.0.0, 0.99.5, 0.99.4, 0.99.3, 0.99.2, 0.99.1, 0.99.0

On the right side, there are orange buttons for "Download" and "Follow". Below them, sections for "Maintainer" (Opscode logo) and "Collaborators" (two user icons) are shown. A "84 Followers" section displays a grid of 16 small user profile pictures.

Monday, 30 June 14

A cookbook displays a lot of information. Some of it comes from the metadata (version, license, platforms), some comes from the upload process or maintaining the cookbook page (maintainer, collaborators, category, home page)

The source code can be browsed on the site directly so you can preview a cookbook before using it.

If the cookbook has a README, the site will display it. Markdown READMEs will be formatted in HTML.

# Viewing a cookbook

The screenshot shows a web browser displaying the Opscode Community website at [community.opscode.com/cookbooks/chef-client](http://community.opscode.com/cookbooks/chef-client). The page title is "Chef Cookbook: chef-client – Opscode Community". The main content area shows the "chef-client [3.0.4]" cookbook. Key details include:

- Rating: ★★★★☆ 6 ratings
- Downloads: 2469 times
- Last updated: Jun 12, 2013
- Description: Manages client.rb configuration and chef-client service
- Category: Utilities
- Homepage: <http://github.com/opscode-cookbooks/chef-client>
- License: Apache 2.0
- Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), mac\_os\_x\_server (>= 0.0.0), openbsd (>= 0.0.0), oracle (>= 0.0.0), redhat (>= 0.0.0), suse (>= 0.0.0), ubuntu (>= 0.0.0), windows (>= 0.0.0)
- Created: Dec 16, 2010
- Last updated: Jun 12, 2013
- Versions: 3.0.4, 3.0.2, 3.0.0, 2.2.4, 2.2.2, 2.2.0, 2.1.0, 2.0.4, 2.0.2, 2.0.0, 1.2.0, 1.1.0, 1.1.2, 1.1.0, 1.0.2, 1.0.0

A dashed orange box highlights the text "README displayed on the page (if it has one)" over the version list. An orange arrow points from the left margin to the "Description" section below.

**Description**  
This cookbook is used to configure a system as a Chef Client.

**Requirements**  
Chef 0.10.10 or greater and Ohai 0.6.12 or greater are required due to the use of `platform_family`.

**Platforms**  
The following platforms are tested directly under test-kitchen; see `.kitchen.yml` and `TESTING.md` for details.  
Ubuntu 10.04, 12.04

**Maintainer**  
opscode

**Collaborators**  
Two user icons.

**84 Followers**  
A grid of 16 small user icons.

Monday, 30 June 14

A cookbook displays a lot of information. Some of it comes from the metadata (version, license, platforms), some comes from the upload process or maintaining the cookbook page (maintainer, collaborators, category, home page)

The source code can be browsed on the site directly so you can preview a cookbook before using it.

If the cookbook has a README, the site will display it. Markdown READMEs will be formatted in HTML.

# Viewing a cookbook

The screenshot shows a web browser displaying the Opscode Community website at [community.opscode.com/cookbooks/chef-client](http://community.opscode.com/cookbooks/chef-client). The page title is "Chef Cookbook: chef-client – Opscode Community". The main content area shows the "chef-client [3.0.4]" cookbook. Key details include:

- Version: 3.0.4
- Ratings: ★★★★☆ (6 ratings)
- Downloads: 2469 times
- Last updated: Jun 12, 2013
- Description: Manages client.rb configuration and chef-client service
- Category: Utilities
- Homepage: <http://github.com/opscode-cookbooks/chef-client>
- License: Apache 2.0
- Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac\_os\_x (>= 0.0.0), mac\_os\_x\_server (>= 0.0.0), openbsd (>= 0.0.0), oracle (>= 0.0.0), redhat (>= 0.0.0), suse (>= 0.0.0), ubuntu (>= 0.0.0), windows (>= 0.0.0)
- Created: Dec 16, 2010
- Last updated: Jun 12, 2013
- Versions: 3.0.4, 3.0.2, 3.0.0, 2.2.4, 2.2.2, 2.2.0, 2.1.0, 2.0.0, 2.0.2, 2.0.0, 1.2.0, 1.1.0, 1.1.2, 1.1.0, 1.0.2, 1.0.0, 0.99.5, 0.99.4, 0.99.3, 0.99.2, 0.99.1, 0.99.0

On the right side, there are buttons for "Download" and "Follow". A dashed orange box highlights the "Source Code" tab, which is currently selected. Another dashed orange box highlights the "README displayed on the page (if it has one)" section, which contains the text: "This cookbook is used to configure a system as a Chef Client". An arrow points from the "Description" section to this README text.

Monday, 30 June 14

A cookbook displays a lot of information. Some of it comes from the metadata (version, license, platforms), some comes from the upload process or maintaining the cookbook page (maintainer, collaborators, category, home page)

The source code can be browsed on the site directly so you can preview a cookbook before using it.

If the cookbook has a README, the site will display it. Markdown READMEs will be formatted in HTML.

# You can download cookbooks directly from the site...

- You can download cookbooks directly from the community site, but:
  - It doesn't put them in your Chef Repository
  - It isn't fast if you know what you're looking for (click, click...)
  - It isn't necessarily fast if you **don't** know what you're looking for.
  - You're already using knife for managing cookbooks and other things in your Chef Repository.

# Introducing Knife Cookbook Site plugin

- Knife includes a "cookbook site" plugin with some sub-commands:
  - search
  - show
  - download
  - ... and more!

# Download and use chef-client cookbook

v1.2.3

Monday, 30 June 14

# Exercise: Use knife to search the community site

```
chef:
  cookbook:          http://cookbooks.opscode.com/api/v1/cookbooks/chef
  cookbook_description: Installs and configures Chef for chef-client and chef-server
  cookbook_maintainer: opscode
  cookbook_name:      chef

chef-client:
  cookbook:          http://cookbooks.opscode.com/api/v1/cookbooks/chef-client
  cookbook_description: Manages client.rb configuration and chef-client service
  cookbook_maintainer: opscode
  cookbook_name:      chef-client

chef-client-cron:
  cookbook:          http://cookbooks.opscode.com/api/v1/cookbooks/chef-client-cron
  cookbook_description: Manages aspects of only chef-client
  cookbook_maintainer: bryanwb
  cookbook_name:      chef-client-cron
```

Monday, 30 June 14

The search command simply takes a single parameter as the query. It doesn't require the field:pattern format like Chef server search. It will return all the results.

# Exercise: Use knife to search the community site

```
$ knife cookbook site search chef-client
```

```
chef:
  cookbook:          http://cookbooks.opscode.com/api/v1/cookbooks/chef
  cookbook_description: Installs and configures Chef for chef-client and chef-server
  cookbook_maintainer: opscode
  cookbook_name:      chef

chef-client:
  cookbook:          http://cookbooks.opscode.com/api/v1/cookbooks/chef-client
  cookbook_description: Manages client.rb configuration and chef-client service
  cookbook_maintainer: opscode
  cookbook_name:      chef-client

chef-client-cron:
  cookbook:          http://cookbooks.opscode.com/api/v1/cookbooks/chef-client-cron
  cookbook_description: Manages aspects of only chef-client
  cookbook_maintainer: bryanwb
  cookbook_name:      chef-client-cron
```

Monday, 30 June 14

The search command simply takes a single parameter as the query. It doesn't require the field:pattern format like Chef server search. It will return all the results.

# Exercise: Use knife to show the chef-client cookbook on the community site

```
average_rating: 4.85714
category: Utilities
created_at: 2010-12-16T23:00:45Z
description: Manages client.rb configuration and chef-client service
external_url: github.com/opscode-cookbooks/chef-client
latest_version: http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_1_0
maintainer: opscode
name: chef-client
updated_at: 2013-10-01T03:14:41Z
versions:
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_1_0
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_0_6
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_0_4
```

# Exercise: Use knife to show the chef-client cookbook on the community site

```
$ knife cookbook site show chef-client
```

```
average_rating: 4.85714
category: Utilities
created_at: 2010-12-16T23:00:45Z
description: Manages client.rb configuration and chef-client service
external_url: github.com/opscode-cookbooks/chef-client
latest_version: http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_1_0
maintainer: opscode
name: chef-client
updated_at: 2013-10-01T03:14:41Z
versions:
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_1_0
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_0_6
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3_0_4
```

# Exercise: Download the chef-client cookbook

Downloading chef-client from the cookbooks site at version 3.1.0 to /Users/YOU/chef-repo/chef-client-3.1.0.tar.gz

Cookbook saved: /Users/YOU/chef-repo/chef-client-3.1.0.tar.gz

Monday, 30 June 14

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Download the chef-client cookbook

```
$ knife cookbook site download chef-client
```

Downloading chef-client from the cookbooks site at version 3.1.0 to /Users/YOU/chef-repo/chef-client-3.1.0.tar.gz

Cookbook saved: /Users/YOU/chef-repo/chef-client-3.1.0.tar.gz

Monday, 30 June 14

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Extract chef-client cookbook tarball

```
x chef-client/
x chef-client/attributes/
x chef-client/CHANGELOG.md
x chef-client/CONTRIBUTING
x chef-client/LICENSE
x chef-client/metadata.json
x chef-client/metadata.rb
x chef-client/README.md
x chef-client/recipes/
x chef-client/templates/
x chef-client/templates/arch/
x chef-client/templates/default/
x chef-client/templates/windows/
x chef-client/templates/default/debian/
x chef-client/templates/default/redhat/
x chef-client/templates/default/solaris/
x chef-client/templates/arch/conf.d/
x chef-client/templates/arch/rc.d/
x chef-client/recipes/config.rb
x chef-client/recipes/cron.rb
x chef-client/recipes/default.rb
x chef-client/recipes/delete_validation.rb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Extract chef-client cookbook tarball

```
$ tar -zxvf chef-client*.tar.gz -C cookbooks/
```

```
x chef-client/
x chef-client/attributes/
x chef-client/CHANGELOG.md
x chef-client/CONTRIBUTING
x chef-client/LICENSE
x chef-client/metadata.json
x chef-client/metadata.rb
x chef-client/README.md
x chef-client/recipes/
x chef-client/templates/
x chef-client/templates/arch/
x chef-client/templates/default/
x chef-client/templates/windows/
x chef-client/templates/default/debian/
x chef-client/templates/default/redhat/
x chef-client/templates/default/solaris/
x chef-client/templates/arch/conf.d/
x chef-client/templates/arch/rc.d/
x chef-client/recipes/config.rb
x chef-client/recipes/cron.rb
x chef-client/recipes/default.rb
x chef-client/recipes/delete_validation.rb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# What we just did...

- Cookbooks are distributed as a versioned .tar.gz archive.
- The latest version is downloaded by default (you can specify the version).
- Extract the cookbook into the "cookbooks" directory with tar.
- Next, let's examine the contents.

Monday, 30 June 14

If asked: There is a version here that uses git under the hood, and does a lot of branching. We aren't using it, because we aren't teaching you version control

# Best practice: well written cookbooks have a README!

- Documentation for cookbooks doesn't need to be extensive, but a README should describe some important aspects of a cookbook:
  - Expectations (cookbooks, platform, data)
  - Recipes and their purpose
  - LWRPs, Libraries, etc.
  - Usage notes
- Read the README first!

Monday, 30 June 14

A cookbook might have certain expectations, such as a Chef version requirement, other cookbooks (and their interaction), or the platforms it is tested on (though may work on other platforms w/o modification), and any data that is expected, such as a data bag or kinds of search it does.

Recipes can be read, but explanation what each is for in a complicated cookbook is helpful.

Other cookbook components, how to use them, or what they're for, like LWRP, libs.

Run list usage, attribute setting and example LWRPs are often under usage.

# Best Practice: This runs as root!

- So, you just downloaded source code from the internet.
- As root.
- To load in the magic machine that:
  - **Makes your computers run code**
- Read the *entire* cookbook first!

Monday, 30 June 14

Again, cookbooks are not vetted.

Opscode does maintain their cookbooks and there's community policing going on for others.  
But just to be clear: always assume that cookbooks are not vetted!

# Examining the chef-client cookbook

- We're going to use two recipes on the node from the chef-client cookbook.
  - `delete_validation`
  - `service` (via default)

# Exercise: View the chef-client::delete\_validation recipe



**OPEN IN EDITOR:**

```
unless chef server?
  file Chef::Config[:validation_key] do
    action :delete
    backup false
    only_if { ::File.exists?(Chef::Config[:client_key]) }
  end
end
```

**SAVE FILE!**

Monday, 30 June 14

This recipe deletes the org validator file if client.pem exists.

We have a couple of sanity checks.

In open source chef server land, we recommend using the 'chef-server' recipe on the server itself, so we don't want to delete the validation key there. We don't keep backups, so we don't have old copies on the node.

We also make sure that the configured client key is there.

# Exercise: View the chef-client::delete\_validation recipe



**OPEN IN EDITOR:** cookbooks/chef-client/recipes/delete\_validation.rb

```
unless chef server?
  file Chef::Config[:validation_key] do
    action :delete
    backup false
    only_if { ::File.exists?(Chef::Config[:client_key]) }
  end
end
```

**SAVE FILE!**

Monday, 30 June 14

This recipe deletes the org validator file if client.pem exists.

We have a couple of sanity checks.

In open source chef server land, we recommend using the 'chef-server' recipe on the server itself, so we don't want to delete the validation key there. We don't keep backups, so we don't have old copies on the node.

We also make sure that the configured client key is there.

# Exercise: Add chef-client::delete\_validation to your base role



**OPEN IN EDITOR:**

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

**SAVE FILE!**

- Add the **delete\_validation** recipe

# Exercise: Add chef-client::delete\_validation to your base role



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

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

**SAVE FILE!**

- Add the **delete\_validation** recipe

## Best Practice: Delete the validation certificate when it isn't required

- Once Chef enters the actual run, synchronizing cookbooks, it has registered its own API client with the validation certificate
- That certificate is no longer required. We do this first because in case the run fails for another reason, we know at least the validation certificate is gone

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



**OPEN IN EDITOR:**

```
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
#  
  
include_recipe "chef-client::service"
```

**SAVE FILE!**

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



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

```
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
#  
  
include_recipe "chef-client::service"
```

**SAVE FILE!**

## Best Practice: Sane defaults do "pretty much" what you expect

- The main point of the "chef-client" cookbook is managing the "chef-client" program. It is designed that it can run as a daemonized service.
- The least surprising thing for most users is that the default recipe starts the service.
- You can manage the service in a number of ways, see the cookbook's README.md.

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



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

- The recipe supports a number of **service** providers and styles.
- It works on a lot of **platforms**.
- Everything is controllable through **attributes**.

```
supported_init_styles = [
  'arch',
  'bluepill',
  'bsd',
  'daemontools',
  'init',
  'launchd',
  'runit',
  'smf',
  'upstart',
  'winsw'
]
init_style = node[ "chef_client" ][ "init_style" ]

# Services moved to recipes
if supported_init_styles.include? init_style
  include_recipe "chef-client::#{init_style}_service"
else
  log "Could not determine service init style, manual intervention required
to start up the chef-client service."
end
```

Monday, 30 June 14

There's a lot to this recipe. We're not going to cover it in detail, you can read it on your own.

There are several init styles that can be selected by setting the node attribute in the recipe. "init" is the default, and what we're going to use. Also available: smf, upstart, arch, runit, bluepill, daemontools, winsw. "cron" is a separate recipe since it is not technically running as a "service".

## Best Practice: Well-written cookbooks change behavior based on attributes

- Ideally, you don't have to modify the contents of a cookbook to use it for your specific use case.
- Look at the attributes directory for things you can override through roles to affect behavior of the cookbook.
- Of course, well written cookbooks have sane defaults, and a README to describe all this.

# Exercise: Upload the chef-client cookbook

```
$ knife cookbook upload chef-client
```

Monday, 30 June 14

Need to grab the cron cookbook too

# Exercise: Upload the chef-client cookbook

```
$ knife cookbook upload chef-client
```

```
Uploading chef-client [3.1.0]
ERROR: Cookbook chef-client depends on cookbook 'cron' version '>= 1.2.0',
ERROR: which is not currently being uploaded and cannot be found on the
server.
```

Monday, 30 June 14

Need to grab the cron cookbook too

# Exercise: Download the cron cookbook

```
Downloading cron from the cookbooks site at version  
1.2.8 to /Users/YOU/chef-repo/cron-1.2.8.tar.gz  
Cookbook saved: /Users/YOU/chef-repo/  
cron-1.2.8.tar.gz
```

# Exercise: Download the cron cookbook

```
$ knife cookbook site download cron
```

```
Downloading cron from the cookbooks site at version  
1.2.8 to /Users/YOU/chef-repo/cron-1.2.8.tar.gz  
Cookbook saved: /Users/YOU/chef-repo/  
cron-1.2.8.tar.gz
```

# Exercise: Extract cron cookbook tarball

```
x cron/
x cron/CHANGELOG.md
x cron/README.md
x cron/metadata.json
x cron/metadata.rb
x cron/providers
x cron/providers/d.rb
x cron/recipes
x cron/recipes/default.rb
x cron/recipes/test.rb
x cron/resources
x cron/resources/d.rb
x cron/templates
x cron/templates/default
x cron/templates/default/cron.d.erb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Extract cron cookbook tarball

```
$ tar -zxvf cron*.tar.gz -C cookbooks/
```

```
x cron/
x cron/CHANGELOG.md
x cron/README.md
x cron/metadata.json
x cron/metadata.rb
x cron/providers
x cron/providers/d.rb
x cron/recipes
x cron/recipes/default.rb
x cron/recipes/test.rb
x cron/resources
x cron/resources/d.rb
x cron/templates
x cron/templates/default
x cron/templates/default/cron.d.erb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Upload the cron cookbook

```
Uploading cron  
Uploaded 1 cookbook.
```

[ 1.2.8 ]

# Exercise: Upload the cron cookbook

```
$ knife cookbook upload cron
```

```
Uploading cron [1.2.8]
Uploaded 1 cookbook.
```

# Exercise: Upload the chef-client cookbook

```
Uploading chef-client [3.1.0]
ERROR: Cookbook chef-client depends on
cookbook 'logrotate' version '>= 1.2.0',
ERROR: which is not currently being
uploaded and cannot be found on the
server.
```

# Exercise: Upload the chef-client cookbook

```
$ knife cookbook upload chef-client
```

```
Uploading chef-client [3.1.0]
ERROR: Cookbook chef-client depends on
cookbook 'logrotate' version '>= 1.2.0',
ERROR: which is not currently being
uploaded and cannot be found on the
server.
```

# Exercise: Download the logrotate cookbook

```
Downloading logrotate from the cookbooks site at
version 1.4.0 to /Users/johnfitzpatrick/
cheftraining/chef-repo/logrotate-1.4.0.tar.gz
Cookbook saved: /Users/YOU/chef-repo/
logrotate-1.4.0.tar.gz
```

# Exercise: Download the logrotate cookbook

```
$ knife cookbook site download logrotate
```

```
Downloading logrotate from the cookbooks site at
version 1.4.0 to /Users/johnfitzpatrick/
cheftraining/chef-repo/logrotate-1.4.0.tar.gz
Cookbook saved: /Users/YOU/chef-repo/
logrotate-1.4.0.tar.gz
```

# Exercise: Extract logrotate cookbook tarball

```
x logrotate/
x logrotate/CHANGELOG.md
x logrotate/README.md
x logrotate/attributes
x logrotate/attributes/default.rb
x logrotate/definitions
x logrotate/definitions/logrotate_app.rb
x logrotate/libraries
x logrotate/libraries/logrotate_config.rb
x logrotate/metadata.json
x logrotate/metadata.rb
x logrotate/recipes
x logrotate/recipes/default.rb
x logrotate/recipes/global.rb
x logrotate/templates
x logrotate/templates/default
x logrotate/templates/default/logrotate-global.erb
x logrotate/templates/default/logrotate.erb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Extract logrotate cookbook tarball

```
$ tar -zxvf logrotate*.tar.gz -C cookbooks/
```

```
x logrotate/
x logrotate/CHANGELOG.md
x logrotate/README.md
x logrotate/attributes
x logrotate/attributes/default.rb
x logrotate/definitions
x logrotate/definitions/logrotate_app.rb
x logrotate/libraries
x logrotate/libraries/logrotate_config.rb
x logrotate/metadata.json
x logrotate/metadata.rb
x logrotate/recipes
x logrotate/recipes/default.rb
x logrotate/recipes/global.rb
x logrotate/templates
x logrotate/templates/default
x logrotate/templates/default/logrotate-global.erb
x logrotate/templates/default/logrotate.erb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Upload the logrotate cookbook

```
Uploading logrotate  
Uploaded 1 cookbook.
```

```
[ 1 . 4 . 0 ]
```

# Exercise: Upload the logrotate cookbook

```
$ knife cookbook upload logrotate
```

```
Uploading logrotate  
Uploaded 1 cookbook.
```

```
[ 1 . 4 . 0 ]
```

# Exercise: Upload the chef-client cookbook

```
Uploading chef-client [ 3.1.0 ]
Uploaded 1 cookbook.
```

# Exercise: Upload the chef-client cookbook

```
$ knife cookbook upload chef-client
```

```
Uploading chef-client [ 3.1.0 ]
Uploaded 1 cookbook.
```

# Exercise: Add chef-client recipe to base role



**OPEN IN EDITOR:**

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

**SAVE FILE!**

# Exercise: Add chef-client recipe to base role



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

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

**SAVE FILE!**

# Exercise: Upload the base role

Updated Role base!

# Exercise: Upload the base role

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

Updated Role base!

# Exercise: Re-run the Chef Client

```
...
Recipe: chef-client::delete_validation
  * file[/etc/chef/validation.pem] action delete
INFO: Processing file[/etc/chef/validation.pem] action delete (chef-client::delete_validation line 25)
INFO: file[/etc/chef/validation.pem] deleted file at /etc/chef/validation.pem

  - delete file /etc/chef/validation.pem

...
  * service[chef-client] action enable
INFO: Processing service[chef-client] action enable (chef-client::init_service line 31)
INFO: service[chef-client] enabled

  - enable service service[chef-client]

  * service[chef-client] action start
INFO: Processing service[chef-client] action start (chef-client::init_service line 31)
INFO: service[chef-client] started

  - start service service[chef-client]

...
INFO: template[/etc/init.d/chef-client] sending restart action to service[chef-client] (delayed)
Recipe: chef-client::init_service
  * service[chef-client] action restart
INFO: Processing service[chef-client] action restart (chef-client::init_service line 31)
INFO: service[chef-client] restarted

  - restart service service[chef-client]
```

Monday, 30 June 14

The validation certificate is deleted.

Aside from some directory creation and the init script itself, the chef-client service is started!

(output trimmed for brevity)

# Exercise: Verify chef-client is running

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

Monday, 30 June 14

Its using our full-stack installation path automatically, and the -d says "runs as a daemon in the background"

We log to /var/log/chef/client.log (configurable). The -i is the interval, a number of seconds (30 minutes here) (configurable) and the last bit is splay (mitigating the thundering herd problem)

Note: embedded dir path -- talk about Omnibus installer if not mentioned already

# Exercise: Verify chef-client is running

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

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

Monday, 30 June 14

Its using our full-stack installation path automatically, and the -d says "runs as a daemon in the background"

We log to /var/log/chef/client.log (configurable). The -i is the interval, a number of seconds (30 minutes here) (configurable) and the last bit is splay (mitigating the thundering herd problem)

Note: embedded dir path -- talk about Omnibus installer if not mentioned already

# Convergent infrastructure

- Our node is now running chef-client as a daemon, and it will converge itself over time on a (by default) 30 minute interval.
- The amount of resources converged may vary with longer intervals, depending on configuration drift on the system.
- Because Chef resources are idempotent, it will only configure what it needs to each run.

Monday, 30 June 14

Get nerdy about convergence!

chef-client daemon can be signaled with USR1 to wake it up

# Download and use ntp cookbook

v1.2.3

Monday, 30 June 14

# NTP Cookbook

- Network time protocol - keeps system clocks in sync
- Chef Server authentication is time sensitive!

Monday, 30 June 14

Mention the time sensitive client-request signature: tolerable API drift (5 mins by default)

Prevents replay attacks.

# Exercise: Download the ntp cookbook

Downloading ntp from the cookbooks site at version 1.5.0 to /Users/YOU/chef-repo/ntp-1.5.0.tar.gz

Cookbook saved: /Users/YOU/chef-repo/ntp-1.5.0.tar.gz

Monday, 30 June 14

If asked: some cookbooks don't use best practices. Some of the oldest cookbooks evolved along with chef. What you are seeing is evolution of best practices in older cookbooks. NTP is one of the older cookbooks.

# Exercise: Download the ntp cookbook

```
$ knife cookbook site download ntp
```

Downloading ntp from the cookbooks  
site at version 1.5.0 to /Users/YOU/  
chef-repo/ntp-1.5.0.tar.gz

Cookbook saved: /Users/YOU/chef-repo/  
ntp-1.5.0.tar.gz

Monday, 30 June 14

If asked: some cookbooks don't use best practices. Some of the oldest cookbooks evolved along with chef. What you are seeing is evolution of best practices in older cookbooks. NTP is one of the older cookbooks.

# Exercise: Extract the ntp cookbook

```
x ntp/
x ntp/CHANGELOG.md
x ntp/README.md
x ntp/attributes
x ntp/attributes/default.rb
x ntp/files
x ntp/files/default
x ntp/files/default/ntp.ini
x ntp/files/default/ntp.leapseconds
x ntp/files/default/tests
x ntp/files/default/tests/minitest
x ntp/files/default/tests/minitest/default_test.rb
x ntp/files/default/tests/minitest/support
x ntp/files/default/tests/minitest/support/helpers.rb
x ntp/files/default/tests/minitest/undo_test.rb
x ntp/metadata.json
x ntp/metadata.rb
x ntp/recipes
x ntp/recipes/default.rb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Exercise: Extract the ntp cookbook

```
$ tar -zxvf ntp*.tar.gz -C cookbooks/
```

```
x ntp/
x ntp/CHANGELOG.md
x ntp/README.md
x ntp/attributes
x ntp/attributes/default.rb
x ntp/files
x ntp/files/default
x ntp/files/default/ntp.ini
x ntp/files/default/ntp.leapseconds
x ntp/files/default/tests
x ntp/files/default/tests/minitest
x ntp/files/default/tests/minitest/default_test.rb
x ntp/files/default/tests/minitest/support
x ntp/files/default/tests/minitest/support/helpers.rb
x ntp/files/default/tests/minitest/undo_test.rb
x ntp/metadata.json
x ntp/metadata.rb
x ntp/recipes
x ntp/recipes/default.rb
```

Monday, 30 June 14

Note: Windows users cannot use \*, they must pass in the full file name

Savvy students may notice that we have a "knife cookbook site install" command as well that does the whole download-untar-get-dependencies incantation.

Only problem is that it does all kinds of stuff with vendor branches and tags in Git that may not be desireable, so here's where you can mention that.

# Examining the ntp cookbook

- The cookbook is quite flexible, but for this exercise we're just interested in the most basic use, an NTP client.
  - default recipe

Monday, 30 June 14

The least intrusive lowest common denominator is typically what ends up being default recipe. So in this case, `ntpclient` is default.

# Exercise: View the ntp::default recipe

```
node['ntp']['packages'].each do |ntppkg|
  package ntpkg
end

service node['ntp']['service'] do
  supports :status => true, :restart => true
  action [:enable, :start]
end

template node['ntp']['conffile'] do
  source  'ntp.conf.erb'
  owner   node['ntp']['conf_owner']
  group   node['ntp']['conf_group']
  mode    '0644'
  notifies :restart,
  "service[#{node['ntp']['service']}]"
end
```

- All **packages** are installed
- The **service** is enabled and started
- The **template** notifies the service

Monday, 30 June 14

Explain: we have some multi-platform handling, and it follows the package/service/template pattern

This recipe has been refactored in an outstanding pull request :)

# Exercise: Upload the ntp cookbook

```
Uploading ntp  
Uploaded 1 cookbook.
```

[ 1.5.0 ]

# Exercise: Upload the ntp cookbook

```
$ knife cookbook upload ntp
```

```
Uploading ntp [1.5.0]
Uploaded 1 cookbook.
```

# Exercise: Add the ntp recipe to the base role



**OPEN IN EDITOR:**

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

**SAVE FILE!**

# Exercise: Add the ntp recipe to the base role



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

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

**SAVE FILE!**

# Exercise: Upload the base role

Updated Role base!

# Exercise: Upload the base role

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

Updated Role base!

# Exercise: Re-run the Chef Client

```
Recipe: ntp::default
  * package[ntp] action install
INFO: Processing package[ntp] action install (ntp::default line 21)

  - install version 1:4.2.6.p3+dfsg-1ubuntu3.1 of package ntp

  * package[ntpdate] action install
INFO: Processing package[ntpdate] action install (ntp::default line 21)
  (up to date)
...
  * template[/etc/ntp.conf] action create
INFO: Processing template[/etc/ntp.conf] action create (ntp::default line 44)
INFO: template[/etc/ntp.conf] backed up to /var/chef/backup/etc/ntp.conf.chef-20130702034747
INFO: template[/etc/ntp.conf] updated content
INFO: template[/etc/ntp.conf] owner changed to 0
INFO: template[/etc/ntp.conf] group changed to 0
INFO: template[/etc/ntp.conf] mode changed to 644

  - update template[/etc/ntp.conf] from 4eb9a0 to 7805aa

...
INFO: template[/etc/ntp.conf] sending restart action to service[ntp] (delayed)
Recipe: ntp::default
  * service[ntp] action restart
INFO: Processing service[ntp] action restart (ntp::default line 33)
INFO: service[ntp] restarted

  - restart service service[ntp]
```

Monday, 30 June 14

Maybe nothing happened, why? Student output will vary based on their node's base OS configuration (usually like the above)

The OS already had the package installed, service enabled and started, and the cookbook used the config file from the package originally.

# Review Questions

- What is the Chef Community site URL?
- What are two ways to download cookbooks from the community site?
- What is the first thing you should read when downloading a cookbook?
- Who vets the cookbooks on the community site?
- Who has two thumbs and reads the recipes they download from the community site?

# Further Resources

v1.2.3



Monday, 30 June 14

# Further Resources: Cookbooks and Plugins

- Useful cookbooks
  - DNS: djbdns, pdns, dnsimple, dynect, route53
  - Monitoring: nagios, munin, zenoss, zabbix
  - Package repos: yum, apt, freebsd
  - Security: ossec, snort, cis\_benchmark
  - Logging: rsyslog, syslog-ng, logstash, logwatch
- Application cookbooks:
  - application, database
  - python, java, php, ruby
- Plugins
  - Cloud: knife-ec2, knife-rackspace, knife-openstack, knife-hp
  - Windows: knife-windows
- More listed on [docs.opscode.com](http://docs.opscode.com)

# Further Resources

- <http://opscode.com/>
- <http://community.opscode.com/>
- <http://docs.opscode.com/>
- <http://learnchef.com>
- <http://lists.opscode.com>
- <http://youtube.com/user/Opscode>
- irc.freenode.net #chef, #chef-hacking, #learnchef
- Twitter #opschef

# Food Fight Show

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



# Opscode #ChefConf 2014

- When:
  - April 15-17, 2014
- Where:
  - Hyatt Regency, San Francisco, CA
- Register:
  - <http://chefconf.opscode.com/chefconf>

# Local Meetup Groups

- Fill in local meetup groups here

# Appendix: Just Enough Ruby for Chef

A quick & dirty crash course

v1.2.3



Monday, 30 June 14

# Lesson Objectives

- After completing the lesson, you will be able to
  - Explain what irb is, and how to use it.
  - Describe the function of:
    - Variable Assignment
    - Basic Arithmetic
    - Strings
    - Truthiness
    - Operators
    - Hashes
  - Regular Expressions
  - Conditionals
  - Method Declaration
  - Classes
  - Objects

# irb - the interactive ruby shell

- irb is the interactive ruby shell
- It is a REPL for Ruby
  - Read-Eval-Print-Loop
  - LISP, Python, Erlang, Clojure, etc.
- An interactive programming environment
- Super-handy for trying things out

Monday, 30 June 14

A little more pontificating before the exercise starts.

REPL originates from LISP. REPL's for PERL, BASIC, PHP, Scala, Erlang, etc. If its an interpretive language, probably has a REPL. Statically compiled, maybe not (like C, no C REPL)

Put something into irb, see what it returns. One value at a time.

# Exercise: Start irb on your ‘target’ node

```
opscode@node1$ /opt/chef/embedded/bin/irb
```

```
irb(main):001:0>
```

Monday, 30 June 14

Since we are using omnibus installers, we need to use the same irb that we shipped, since you very likely do not have ruby installed.

# Variable assignment

```
opscode@node1$ /opt/chef/embedded/bin/irb
```

```
irb(main):001:0> x = "hello"  
=> "hello"
```

```
irb(main):002:0> puts x  
hello  
=> nil
```

Monday, 30 June 14

Variables are assigned with an "=" sign

The REPL prints the return value of the last statement, with a => in front

puts is the ruby equivalent of "print" in other languages – it automatically includes a newline

note that puts is a function with a return value – everything has one, and puts returns nil

# Arithmetic

```
irb(main):003:0> 1 + 2  
=> 3
```

Monday, 30 June 14

Plus is addition – numbers are unquoted

Note the return code!

Note if strings: "1" + "2" = "12"

# Arithmetic

```
irb(main):004:0> 18 - 5  
=> 13
```

# Arithmetic

```
irb(main):005:0> 2 * 7  
=> 14
```

# Arithmetic

```
irb(main):006:0> 5 / 2  
=> 2
```

Monday, 30 June 14

5 / 2 is... 2?

Division on whole numbers means no decimal points

# Arithmetic

```
irb(main):007:0> 5 / 2.0  
=> 2.5
```

Monday, 30 June 14

Division on floating point numbers means decimal points

# Arithmetic

```
irb(main):008:0> 5.class  
=> Fixnum
```

```
irb(main):009:0> 5.0.class  
=> Float
```

Monday, 30 June 14

Ruby has dots to call methods – and everything in ruby is an object, including a number. We can find out what **class** an object is by calling ".class". 5 is a "fixnum" – and fixed numbers in ruby are generally whole numbers.

# Arithmetic

```
irb(main):010:0> 1 + (2 * 3)  
=> 7
```

# Strings

```
irb(main):011:0> 'jungle'  
=> "jungle"
```

Monday, 30 June 14

Strings in ruby can use single quotes

When REPL prints back to you, it prints as double quoted version. Why? To explicitly state this is a string.

# Strings

```
irb(main):012:0> 'it\'s alive'  
=> "it's alive"
```

Monday, 30 June 14

You escape the quote delimiter with a forward slash

REPL gives double quoted version, so no escape character

# Strings

```
irb(main):013:0> "animal"  
=> "animal"
```

# Strings

```
irb(main):014:0> "\\"so easy\\"
=> "\\\"so easy\\\""
```

```
irb(main):015:0> puts "\\"so easy\\"
"so easy"
=> nil
```

Monday, 30 June 14

You have to escape the delimiter

note that the repl shows us the return value here is a string, and quotes it for you – if we do it again, you see the string doesn't have the escapes in it when we print the value

# Strings

```
irb(main):016:0> x = "pretty"  
=> "pretty"
```

```
irb(main):017:0> "#{x} nice"  
=> "pretty nice"
```

```
irb(main):018:0> '#{x} nice'  
=> "\#{x} nice"
```

Monday, 30 June 14

You can embed the value of an expression with #{}, as we saw earlier in the class

Note that you cant do it in single-quoted strings! – you get the literal string

# Truthiness

```
irb(main):019:0> true
=> true
```

```
irb(main):020:0> false
=> false
```

```
irb(main):021:0> nil
=> nil
```

```
irb(main):022:0> !!nil
=> false
```

```
irb(main):023:0> !!0
=> true
```

```
irb(main):024:0> !!x
=> true
```

Monday, 30 June 14

In ruby, you can force a value to be converted to true or false with "!!". This is mostly useful in the repl, but occasionally you will see it in idiomatic usage.

Remember that 0 is true (unlike in perl, and some other languages)

Remember that any other value is true when pressed

# Operators

```
irb(main):025:0> 1 == 1  
=> true
```

```
irb(main):026:0> 1 == true  
=> false
```

```
irb(main):027:0> 1 != true  
=> true
```

```
irb(main):028:0> !!1 == true  
=> true
```

Monday, 30 June 14

`==` tests for equality

1 evaluates to true \*when pressed\* – but it is not true

# Operators

```
irb(main):029:0> 2 < 1  
=> false
```

```
irb(main):030:0> 2 > 1  
=> true
```

```
irb(main):031:0> 4 >= 3  
=> true
```

```
irb(main):032:0> 4 >= 4  
=> true
```

```
irb(main):033:0> 4 <= 5  
=> true
```

```
irb(main):034:0> 4 <= 3  
=> false
```

Monday, 30 June 14

Greater than, less than, greater than or equal to, less than or equal to

# Operators

```
irb(main):035:0> 5 <=> 5  
=> 0
```

```
irb(main):036:0> 5 <=> 6  
=> -1
```

```
irb(main):037:0> 5 <=> 4  
=> 1
```

Monday, 30 June 14

The spaceship operator – returns 0 if it is equal, -1 if it is greater, and 1 if it is less than. Super, duper useful for sorting!

# Arrays

```
irb(main):038:0> x = [ "a", "b", "c"]
=> [ "a", "b", "c"]
```

```
irb(main):039:0> x[0]
=> "a"
```

```
irb(main):040:0> x.first
=> "a"
```

```
irb(main):041:0> x.last
=> "c"
```

Monday, 30 June 14

They have methods just like everything else – first and last, for instance

# Arrays

```
irb(main):042:0> x + ["d"]
=> [ "a", "b", "c", "d" ]
```

```
irb(main):043:0> x
=> [ "a", "b", "c" ]
```

```
irb(main):044:0> x = x + ["d"]
=> [ "a", "b", "c", "d" ]
```

```
irb(main):045:0> x
=> [ "a", "b", "c", "d" ]
```

Monday, 30 June 14

Adding the element didn't mutate x – still the original value  
The operation was not destructive to the original object  
We have to assign the value of the expression if we want that

# Arrays

```
irb(main):046:0> x << "e"
=> [ "a", "b", "c", "d", "e" ]
```

```
irb(main):047:0> x
=> [ "a", "b", "c", "d", "e" ]
```

Monday, 30 June 14

The "<<" append operator will add an item to an array!

It also mutates it! – this is more idiomatic than using the + operator

Note: there is no "pre-pend operator", but there are functions that allow you to prepend

# Arrays

```
irb(main):048:0> x.map { |i| "the letter #{i}" }
=> ["the letter a", "the letter b", "the letter
c", "the letter d", "the letter e"]
```

```
irb(main):049:0> x
=> [ "a", "b", "c", "d", "e" ]
```

Monday, 30 June 14

Ruby is full of iterators – we showed you "each" earlier. Map iterates the array and takes a block, whose return value becomes the value of an item in a new array.

Note we did not change x!

We talked earlier about block syntax, and that {}'s are the same as do..end. Here we are using {} on one line, which is idiomatic.

# Arrays

```
irb(main):050:0> x.map! { |i| "the letter #{i}" }
=> ["the letter a", "the letter b", "the letter
c", "the letter d", "the letter e"]
```

```
irb(main):051:0> x
=> ["the letter a", "the letter b", "the letter
c", "the letter d", "the letter e"]
```

Monday, 30 June 14

The same thing, only with a ! on the end of map means "be destructive to the object I am operating on".. the results are a mutated x

Think of this as being "with feeling"... Once more, with feeling!

# Hashes

```
irb(main):052:0> h = {  
irb(main):053:1* "first_name" => "Gary",  
irb(main):054:1* "last_name" => "Gygax"  
irb(main):055:1> }  
=> {"first_name"=>"Gary",  
"last_name"=>"Gygax"}
```

# Hashes

```
irb(main):056:0> h.keys  
=> ["first_name", "last_name"]
```

```
irb(main):057:0> h["first_name"]  
=> "Gary"
```

```
irb(main):058:0> h["age"] = 33  
=> 33
```

```
irb(main):059:0> h.keys  
=> ["first_name", "last_name", "age"]
```

Monday, 30 June 14

We can add items to the hash by putting the key in [], and using assignment on that key name

# Hashes

```
irb(main):060:0> h.values  
=> [ "Gary", "Gygax", 33]
```

Monday, 30 June 14

We can get all the items in the hash with .values

# Hashes

```
irb(main):061:0> h.each { |k, v| puts "#{k}: #{v}" }
first_name: Gary
last_name: Gygax
age: 33
=> {"first_name"=>"Gary", "last_name"=>"Gygax",
"age"=>33}
```

Monday, 30 June 14

We can iterate over the hash with .each – here we print out the keys and values

# Regular Expressions

```
irb(main):062:0> x = "I want to believe"  
=> "I want to believe"
```

```
irb(main):063:0> x =~ /I/  
=> 0
```

```
irb(main):064:0> x =~ /lie/  
=> 12
```

```
irb(main):065:0> x =~ /smile/  
=> nil
```

```
irb(main):066:0> x !~ /smile/  
=> true
```

Monday, 30 June 14

The first character matches, so we return 0

The 12th character matches, so we return 12

No smiles, so nil

!~ is an invalidated regular expression – not equal to the regex, right?

**what happens to those numbers when pressed?** – hence, you can use these as values in conditionals later on!

# Regular Expressions

```
irb(main):067:0> x.sub(/t/, "T")
=> "I wanT to believe"
```

```
irb(main):068:0> puts x
I want to believe
=> nil
```

```
irb(main):069:0> x.gsub!(/t/, "T")
=> "I wanT To believe"
```

```
irb(main):070:0> puts x
I wanT To believe
=> nil
```

# Conditionals

```
irb(main):071:0> x = "happy"
=> "happy"

irb(main):072:0> if x == "happy"
irb(main):073:1>   puts "Sure am!"
irb(main):074:1> elsif x == "sad"
irb(main):075:1>   puts "Boo!"
irb(main):076:1> else
irb(main):077:1*>   puts "Therapy?"
irb(main):078:1> end
Sure am!
=> nil
```

Monday, 30 June 14

If/elsif/else is pretty common in the world

What is uncommon – this construct has a return value of the return value of the last expression in the leg that matches.

You might see assignment based on this in ruby

# Conditionals

```
irb(main):079:0> case x
irb(main):080:1> when "happy"
irb(main):081:1>   puts "Sure Am!"
irb(main):082:1>   1
irb(main):083:1> when "sad"
irb(main):085:1>   puts "Boo!"
irb(main):086:1>   2
irb(main):087:1> else
irb(main):088:1>   puts "Therapy?"
irb(main):089:1>   3
irb(main):090:1> end
Sure Am!
=> 1
```

Monday, 30 June 14

Case is a shorter way to evaluate more than two conditionals on the same variable. By default, you get equality – but you can do regular expressions too, class names, all sorts of stuff.

We added some return values here to illustrate the point

Why did we use case here? To illustrate a point. Rule of thumb: If possibilities  $\geq 3$ , use if and elsif. If more than 3, use case. Style choice.

# Method Definition

```
irb(main):091:0> def metal(str)
irb(main):092:1>   puts "!!#{str} is metal!!"
irb(main):093:1> end
=> nil
```

```
irb(main):094:0> metal("ozzy")
!!ozzy is metal!!
=> nil
```

Monday, 30 June 14

Methods – "def" defines a new one with a string input. The return code of the last expression is the return code of the method – in this case, the last expression is puts, and as we have seen a million times: puts returns nil

Check it – even defining a new method has a return code!!! "def" is itself a function! The rabbit hole is so **dddddddeeeeeeeeeeeepppppppp**

Note: Ruby is not statically typed. Bad for manipulexity, great for whipuptitude. If it were statically typed, any non-programmers would have to learn type theory before picking up Chef.

# Classes

```
irb(main):095:0> class Person
irb(main):096:1>   attr_accessor :name, :is_metal
irb(main):097:1>
irb(main):098:1>   def metal
irb(main):099:2>     if @is_metal
irb(main):100:3>       puts "!!#{@name} is metal!!"
irb(main):101:3>     end
irb(main):102:2>   end
irb(main):103:1> end
=> nil
```

Monday, 30 June 14

The hard way strikes again!!!! Typing!

Ruby is object oriented – everything has a class.  
A class is just a collection of properties with methods attached

attr\_accessor creates two "getter/setter" (aka read/write) methods for a person – their name, and if they are metal. also exists attr\_reader & attr\_writer

We change our metal method to use the @ variable – remember from templates? these are instance variables, they are different based on the object we create

# Classes

```
irb(main):104:0> p = Person.new  
=> #<Person:0x891ab4c>  
  
irb(main):105:0> p.name = "Adam Jacob"  
=> "Adam Jacob"  
  
irb(main):106:0> p.is_metal = true  
=> true  
  
irb(main):107:0> p.metal  
!!Adam Jacob is metal!!  
=> nil  
  
irb(main):108:0> p.is_metal = false  
=> false  
  
irb(main):109:0> p.metal  
=> nil
```

Monday, 30 June 14

Create a new instance of your class with ".new"  
Use the accessors with .name = ""

# Review Questions

- What is irb?
- What is true in ruby? What is false?
- How do you press for the truth?
- What does `>=` do?
- How do you define a method?
- What is a class?
- What is an object?
- The book you want: "Programming Ruby 1.9" <http://pragprog.com/book/ruby3/programming-ruby-1-9>

v1.2.3



Monday, 30 June 14