

Chef Fundamentals - Windows

training@chef.io
Copyright (C) 2014 Chef Software, Inc.

v2.1.1_WIN



Introductions

v2.1.1_WIN



Instructor introduction

- * Name, email
- * Previous job roles/background
- * Current job role
- * Experience with Chef

Hand out your business card

Instructor Introduction

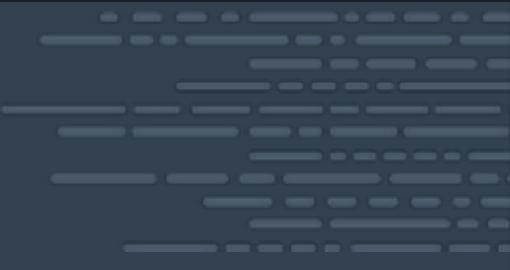
- **Name:** Larry Eichenbaum
- **Current job role:** Automation Engineer
- **Previous job roles/background:** Automation Engineer for ‘other’ automation tools, Change/Release Mgr., QA Mgr., Technical Project Mgr., SysAdmin, farmer...
- **Experience with Chef/Config Management:** Well versed with variety of CM tools, process definition, etc.
- **Favorite Text Editor:** Sublime Text 2 and vi

Introduce Yourselves

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

Student introduction

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



Course Objectives and Style

v2.1.1_WIN



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

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

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

We are going to do things the **hard way** – which gets you to fluency fast. It's only the "hard" way because it doesn't cut corners.

1. You will go through each exercise. You can't be absent, or late, or left behind.
2. You will type in each line of code by hand
3. We will make it run, and fix any bugs when they happen

We will introduce concepts when you type them, and we will repeat them as we go.

Copy-pasting or transferring files from another student will undermine your success – **ban it :)**

The result is you will know how to actually use Chef.

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

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.

Fundamentals

- This course covers the fundamentals of Chef
- We likely will not have time to discuss the latest trends in the Chef ecosystem
- Any discussion items that are too far off topic will be captured
 - We'll return to these items as time permits

Stages of Learning - Shuhari

- Shuhari - first learn, then detach, and finally transcend
 - shu - "obey" - traditional wisdom
 - ha - "detach" - break with tradition
 - ri - "separate" - transcend

守破離

<http://en.wikipedia.org/wiki/Shuhari>

"It is known that, when we learn or train in something, we pass through the stages of *shu*, *ha*, and *ri*. These stages are explained as follows. In *shu*, we repeat the forms and discipline ourselves so that our bodies absorb the forms that our forbearers created. We remain faithful to the forms with no deviation. Next, in the stage of *ha*, once we have disciplined ourselves to acquire the forms and movements, we make innovations. In this process the forms may be broken and discarded. Finally, in *ri*, we completely depart from the forms, open the door to creative technique, and arrive in a place where we act in accordance with what our heart/mind desires, unhindered while not overstepping laws."

Agenda

v2.1.1_WIN



Topics

- Overview of Chef
- Workstation Setup
- Organization Setup
- Test Node Setup
- Writing your first cookbook
- Dissecting your first Chef run
- Introducing the Node object
- Attributes, Templates, and Cookbook Dependencies

We're going to go through an overview of all the concepts you are going to learn.
Make a note of how much of the focus is on writing cookbooks.

Topics

- Template Variables, Notifications, and Controlling Idempotency
- Data bags, search
- Roles
- Environments
- Using community cookbooks
- Further Resources

Breaks!

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

Take 5 minute breaks frequently

We have a lot of content to go through.

Take a 15 minute break after every few sections or 2 hours (whichever comes first)

Take a working (~30 minute) lunch break after the first round of writing Cookbooks

Lunch time lecture: Using Community Resources (talk through community cookbooks and the wiki). Break for 15 mins after lunch session (cleanup etc)

Take the last break right after Refactoring IIS (deserves its own break before and after)

Overview of Chef

v2.1.1_WIN



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

If the class has folks who already know some Chef, this will be a review. The course will go into detail on each topic – so if you have specific questions about a part of Chef, or a best practice, best to hold it until later. Talk relationships and high level.

Takes ~45 minutes, and prepare them for this being the last lecture for a while.

Complexity



As sysadmins, developers, service providers, we have a lot of complexity to manage. This could be a small 4 node application, right up to an application with 1000's nodes

Items of Manipulation (Resources)

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

A tale of growth...



Application

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

Add a database



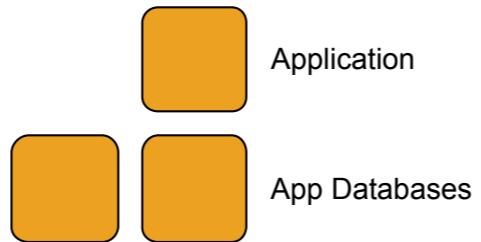
Application



Application Database

Apps are cool, but storing data is cooler

Make database redundant



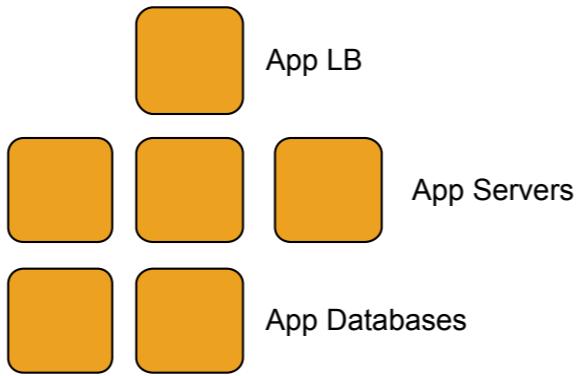
Of course, we want to have redundant HA database servers

Application server redundancy



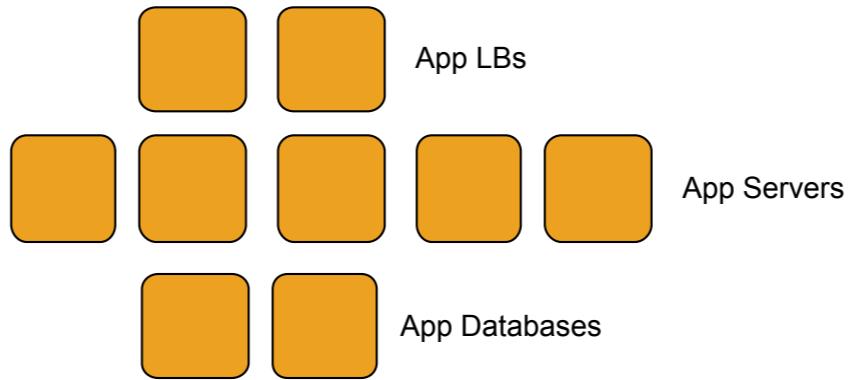
We need more app servers, redundancy right?

Add a load balancer



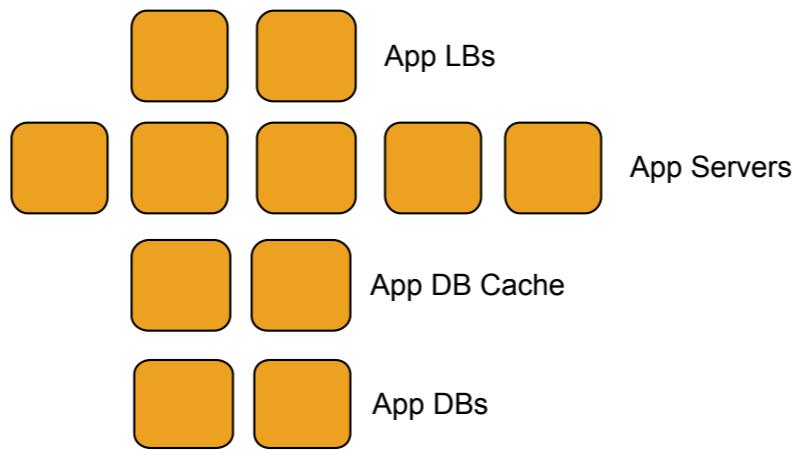
We need a load balancer in front of all these app servers

Webscale!



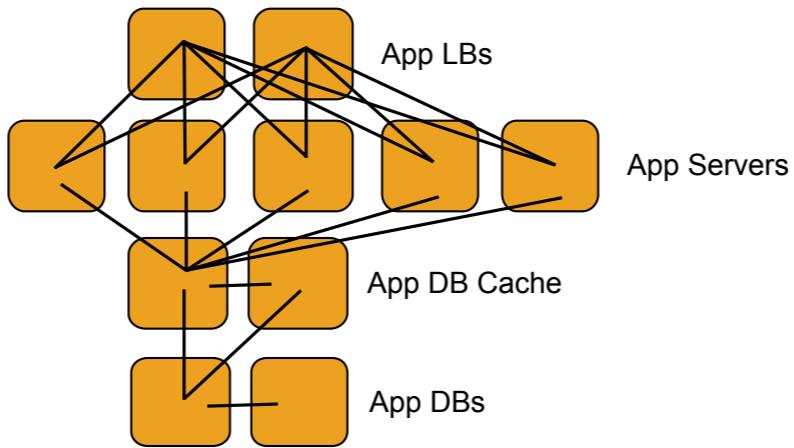
Scaling out the app layer, redundant LB

Now we need a caching layer



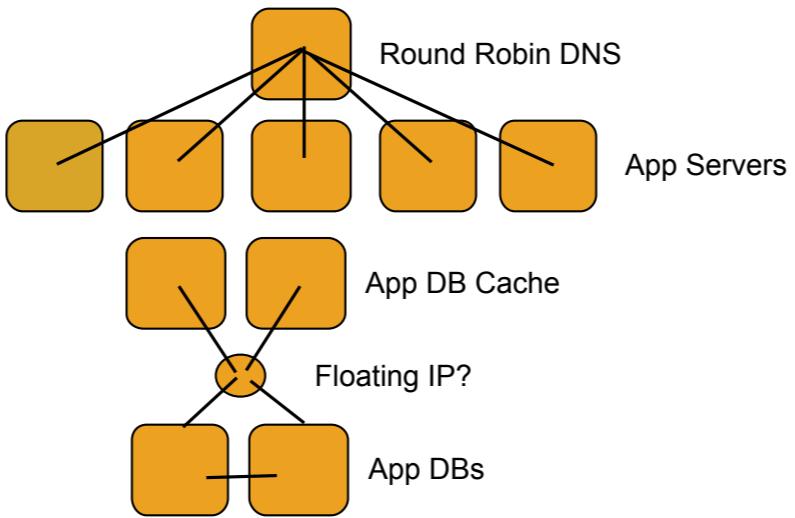
Posted on slashdot, hacker news or techcrunch! We must construct additional pylons

Infrastructure has a Topology



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

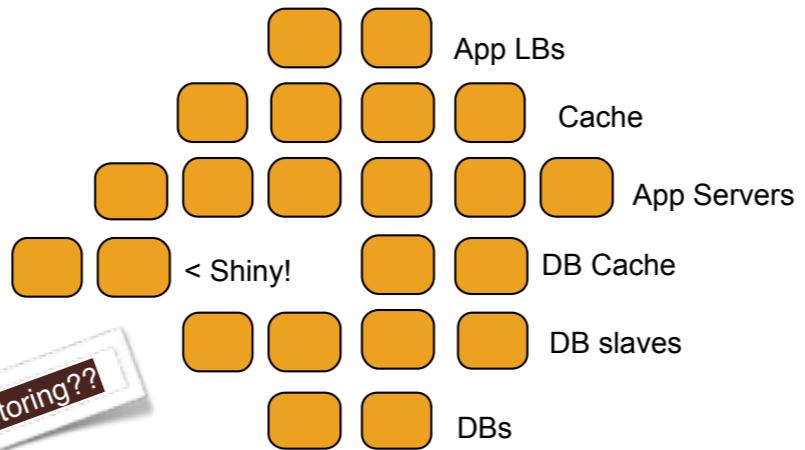
Your Infrastructure is a Snowflake



How should I know. It's your application.

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

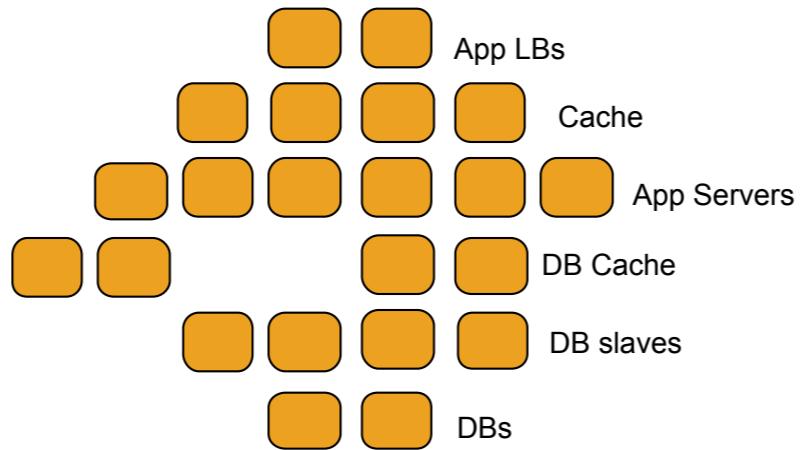
Complexity Increases Quickly



This complexity comes in the form of:

- Resolving technical debt
- Business requirements
- Shiny object syndrome

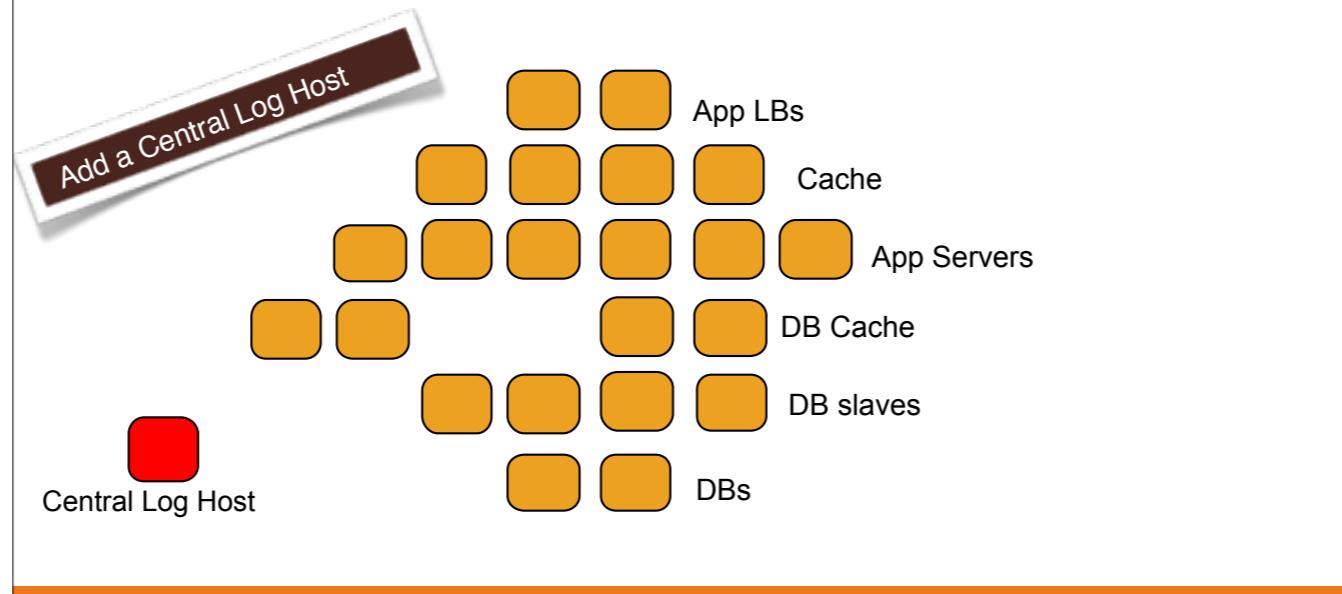
...and change happens!



It's not just about growing your infrastructure.

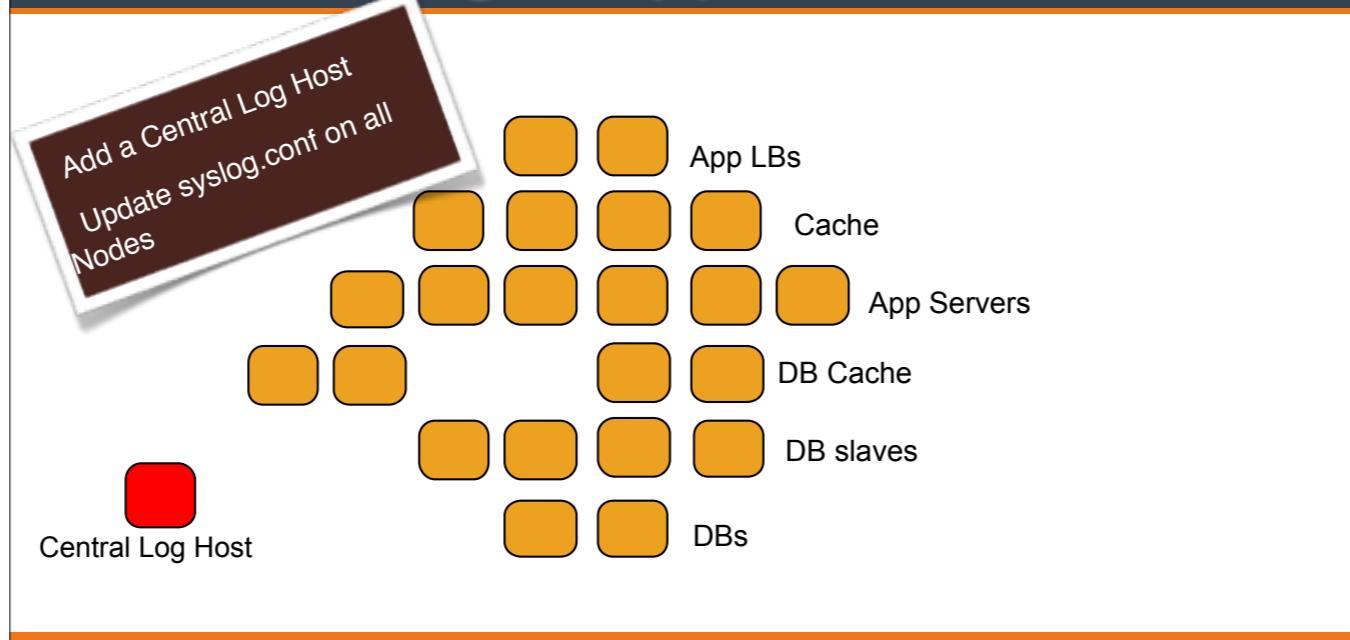
Your infrastructure will also need to change over time.

...and change happens!



Add a central log host

...and change happens!



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

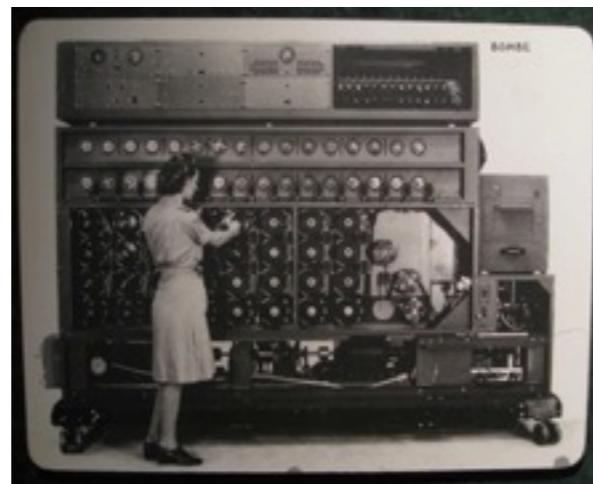
Chef Solves This Problem



CHEF™
GETCHEF.COM

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

Chef is Infrastructure as Code

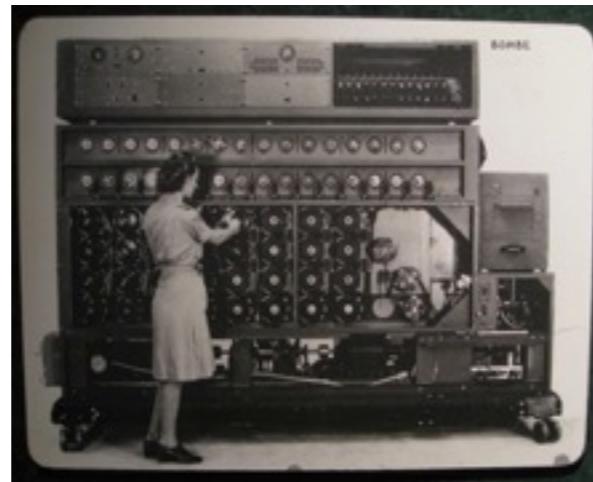


<http://www.flickr.com/photos/louisb/4555295187/>

- Programmatically provision and configure components

-- Repeatability
-- Documentation

Chef is Infrastructure as Code

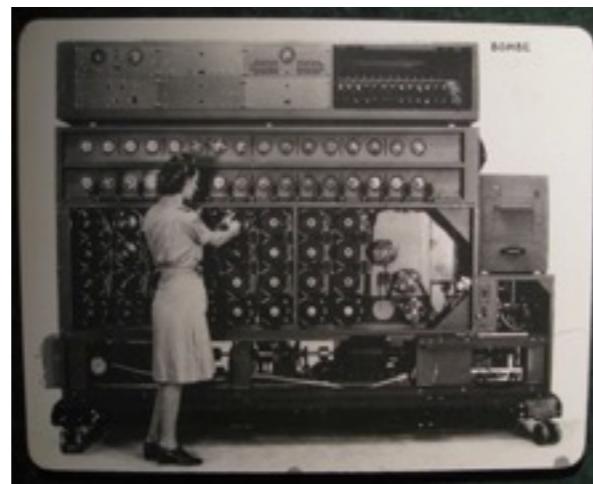


<http://www.flickr.com/photos/louisb/4555295187/>

- Treat like any other code base

-- Repeatability
-- Documentation

Chef is Infrastructure as Code

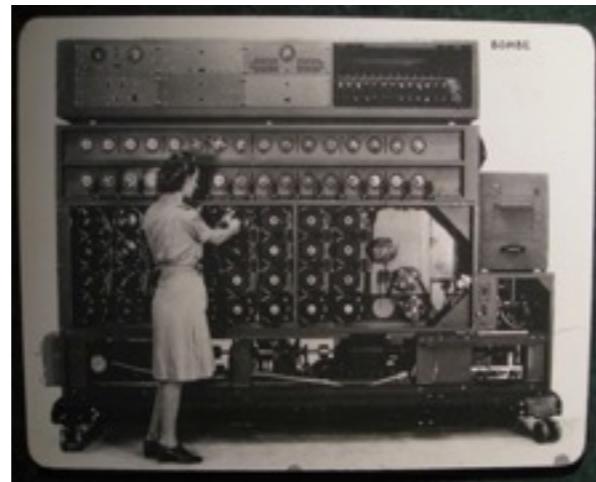


<http://www.flickr.com/photos/louisb/4555295187/>

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

-- Repeatability
-- Documentation

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 compute resources**

-- Repeatability
-- Documentation

Configuration Code

- Chef ensures each Node complies with the policy
- Policy is determined by the configurations 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

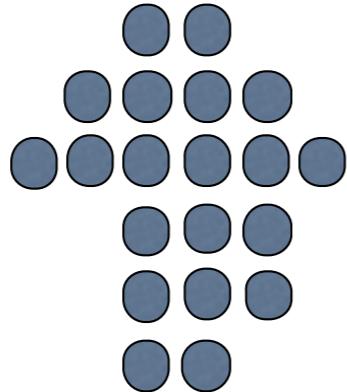
Managing Complexity

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

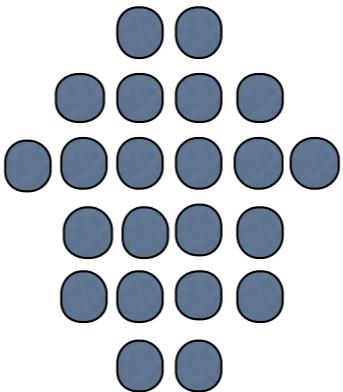
Chef gives us number of tools to help us manage this complexity

Organizations

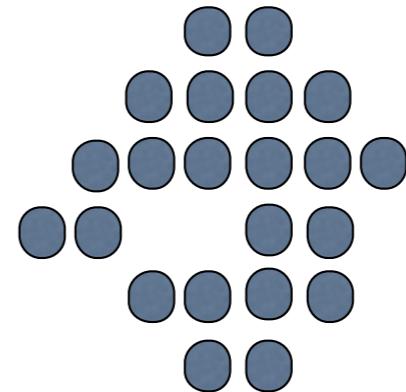
My Infrastructure



Your Infrastructure



Their Infrastructure



Organizations

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

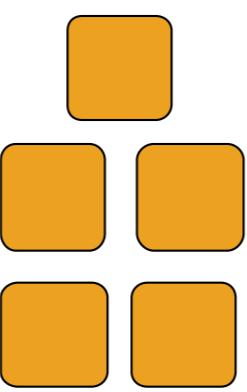
Organizations are independent tenants in Enterprise Chef

Environments

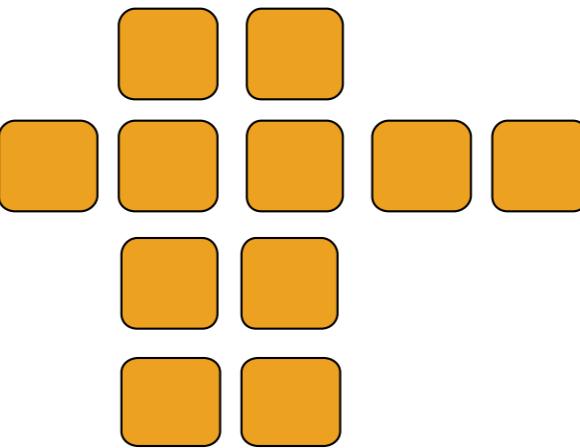
Development



Staging



Production



Environments are used to manage stages in your application

Environments

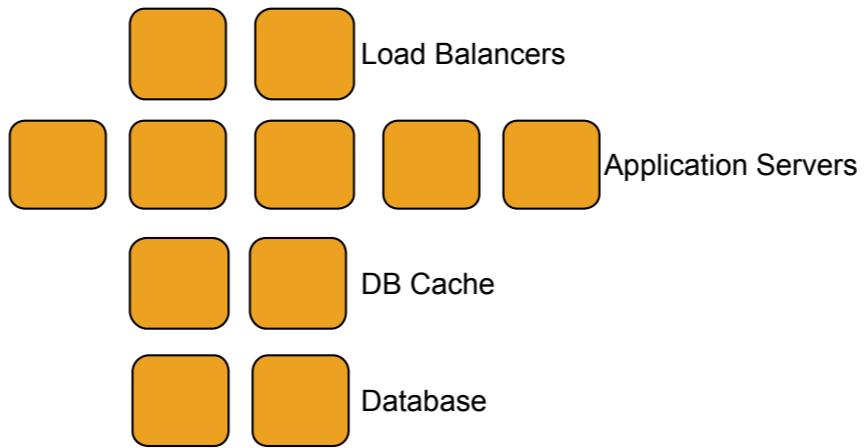
- Environments reflect your patterns and workflow, and can be used to model the life-stages of your applications
 - Development
 - Test
 - Staging
 - Production
 - etc.
- Every Organization starts with a single environment!

An environment is a container within an organization that can mimic structural hierarchy via custom attributes and cookbook restraints

Environments Define Policy

- Environments may include data attributes necessary for configuring your infrastructure, e.g.
 - 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 are a way of classifying the different types of server in your infrastructure

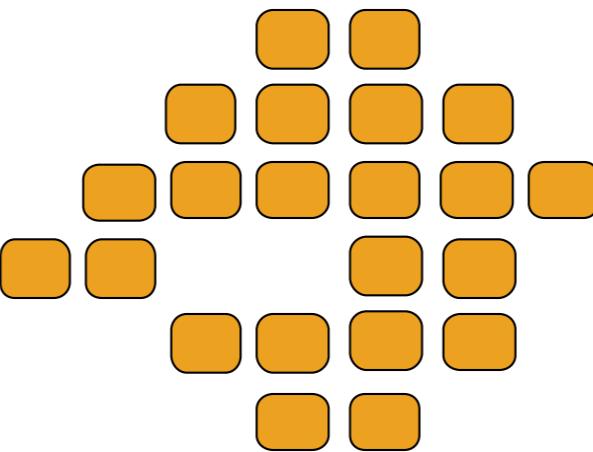
Roles

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

Roles Define Policy

- Roles may include an ordered list of Chef configuration files that should be applied
 - This list is called a Run List
 - Order is always important in the Run List
- Roles may include data attributes necessary for configuring your infrastructure, for example:
 - The port that the application server listens on
 - A list of applications that should be deployed

Nodes



You could have many nodes in your infrastructure, in any configuration.

Nodes

- Nodes represent anything that needs configuration
- Typically the servers in your infrastructure
 - Could be physical servers or virtual servers
 - May represent hardware that you own or compute instances in a public or private cloud
- Could also be network hardware - switches, routers, etc...

network device support for some vendors Arista, Juniper, Cisco

Node

- Each Node will
 - Belong to one Organization
 - Belong to one Environment
 - Have zero or more Roles

Nodes Adhere to Policy

- The chef-client application runs on each node, which
 - Gathers the current system configuration of the node
 - Downloads the desired system configuration policies from the Chef server for that node
 - Configures the node such that it adheres to those policies

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 scheduled 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

"Recipes are segments of code that do task X."

Example Recipe

```
windows_package 'Microsoft .NET Framework 4.0' do
  action :install
  source 'http://download.microsoft.com/download/dotNetFx40_Full_x86_x64.exe'
  options '/quiet /norestart'
end
```

```
windows_registry 'HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server' do
  values 'FdenyTSConnections' => 0
end
```

```
service 'W3SVC' do
  action [:disable,:stop]
end
```

As opposed to Linux (packages, files, services), the holy trinity in the Windows world is more “packages, registry keys, services).

If you feel the class can grok it now - Take some time here to walk through the code.

The windows_package resource has a mock URL (MS URLs are dreadfully long). It takes options “/quiet and /restart”. The action is to install.

The windows_registry resource enables Remote Desktop and pokes a firewall hole.

The W3SVC is IIS.

Or this...

```
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
```

58

Or Linux! Chef works on multiple platforms.
We will look at these resources more carefully

If you feel the class can grok it now – Take some time here to walk through the code. Describe what's happening within each resource declaration. Show how notifications work. Cover sufficient detail that it is clear that each resource in a recipe will (or may):

- * Describe the desired state of that resource (via the “action” keyword)
- * Include a number of resource-specific attributes (owner, mode, supports, etc.)
- * Interact with other resources through the notification mechanism.
- * Will only take action if required. If a resource is out of policy, it will be repaired.

Go into as much detail as you like and as the students will consume.

“What questions do you have?”

Cookbooks

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

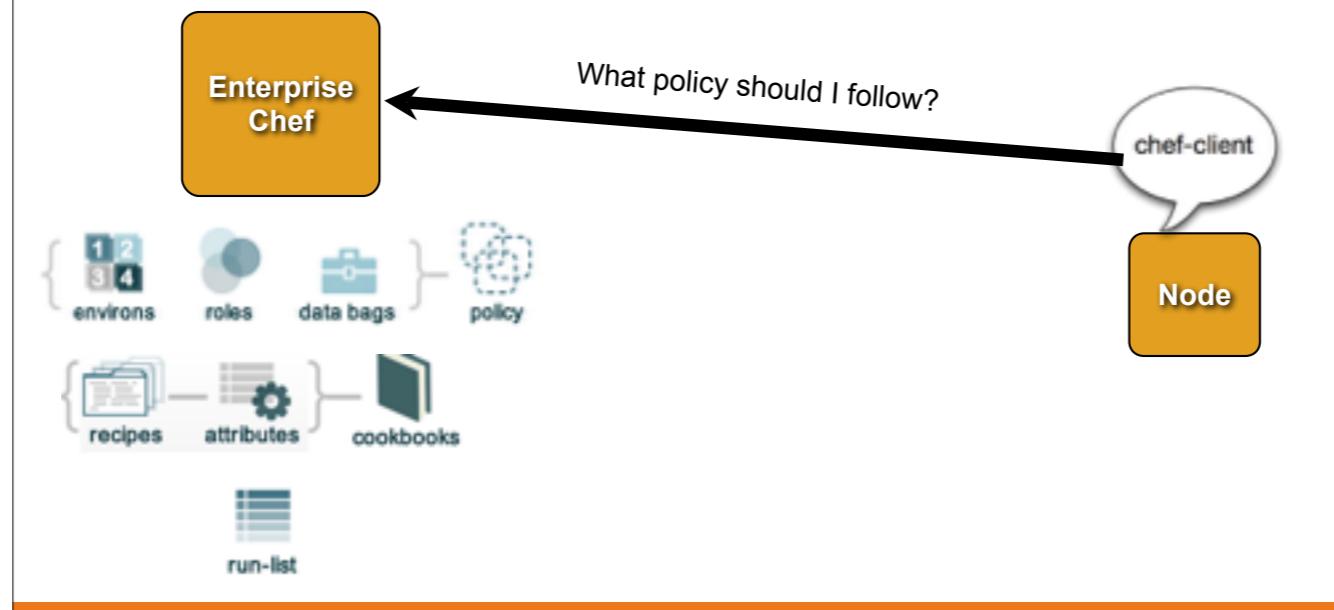


<http://www.flickr.com/photos/shutterhacks/4474421855/>

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

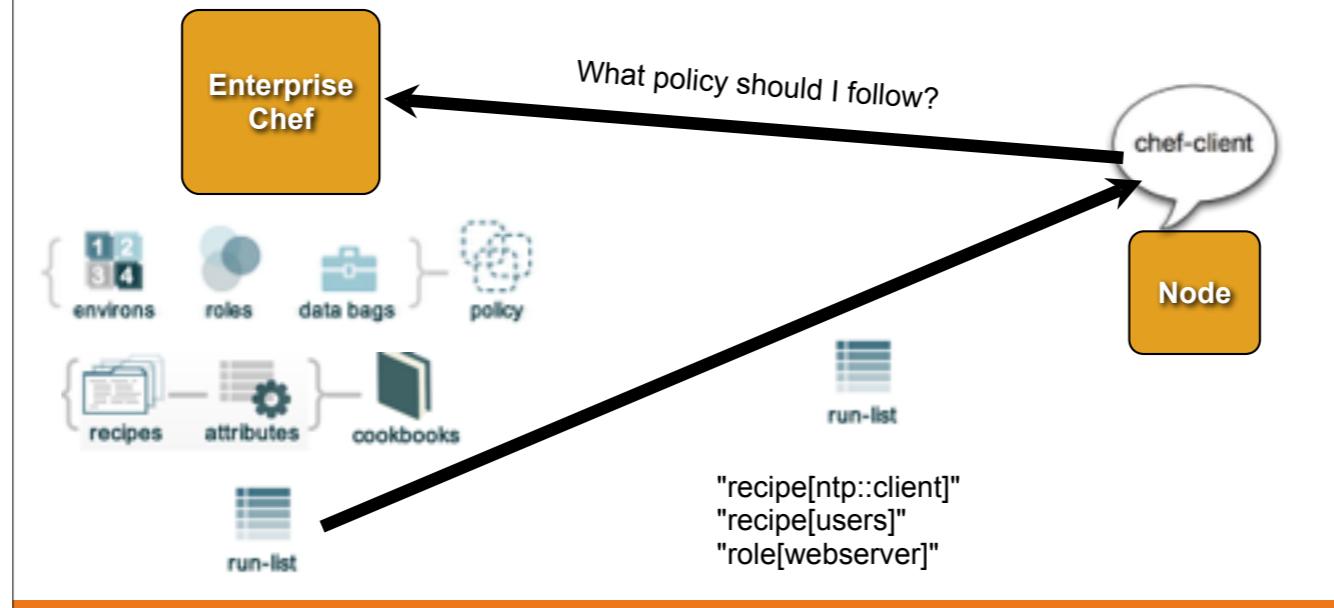
"Cookbooks are a collection of recipes, templates, attributes, providers resources that can be executed against a node to achieve convergence and idempotency"

Run List



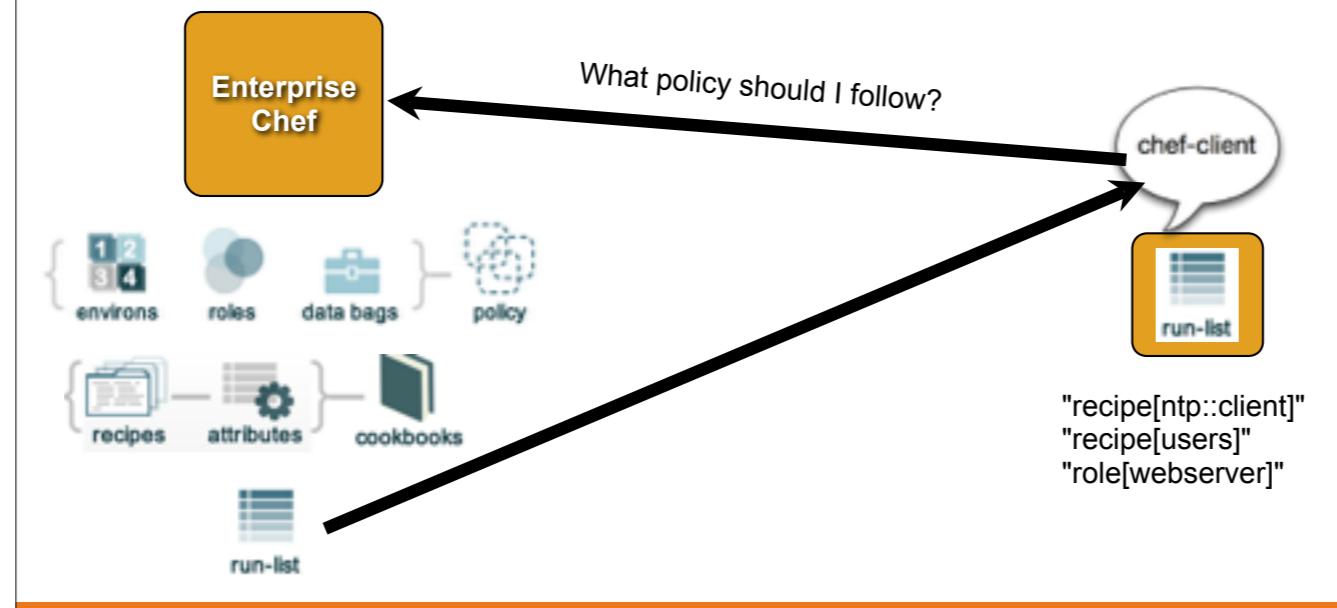
When the chef-server run on the node it asks the chef server what policy should I follow, or what is my run list

Run List



Chef server looks at the run_list and sends the appropriate cookbooks to the node

Run List



Chef-client then executes the resources on the run list

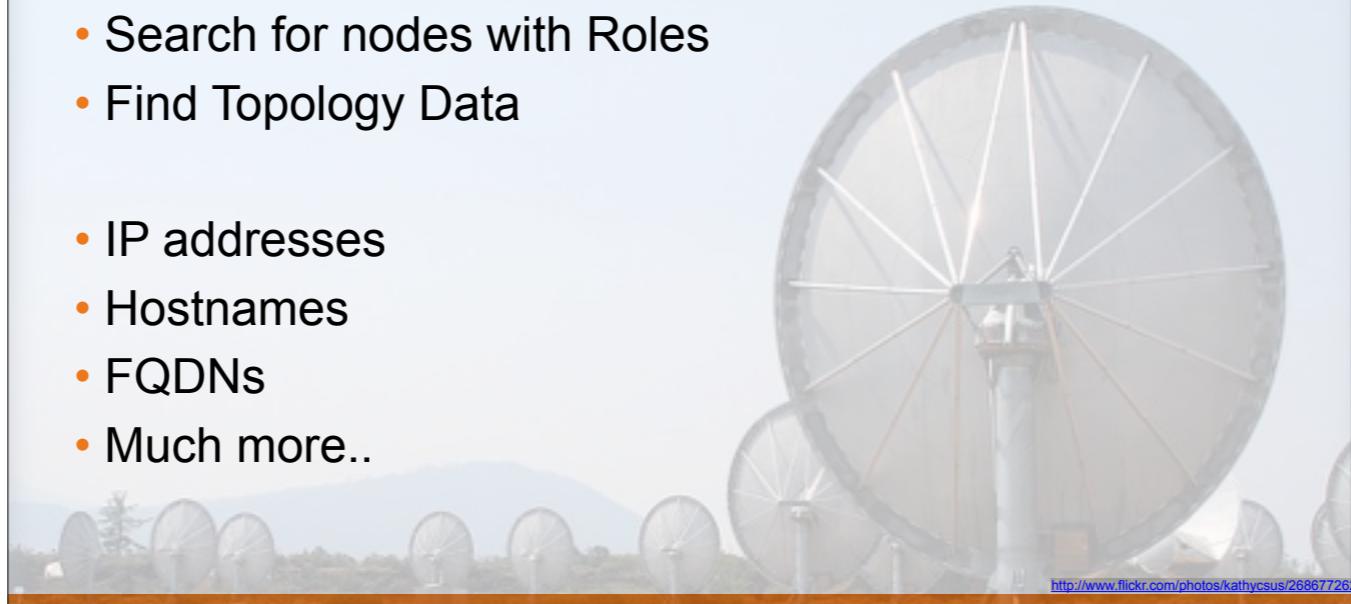
All the heavy lifting is done on the node!! the server is quite light weight!

Run List Specifies Policy

- The Run List is an ordered 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
- Much more..



<http://www.flickr.com/photos/kathycsus/26867726>

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 - Recipe

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

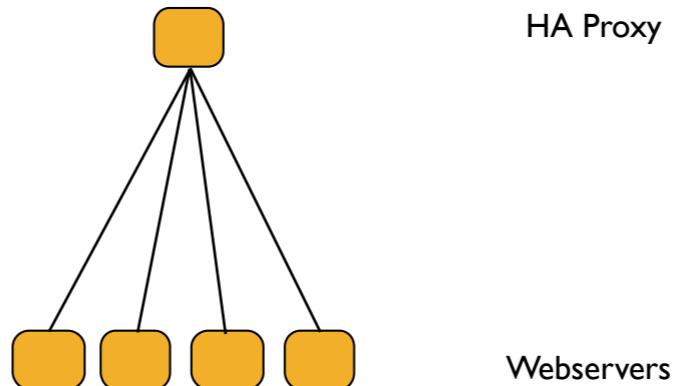
template "/etc/haproxy/haproxy.cfg" do
  source "haproxy-app_lb.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  variables :pool_members => pool_members.uniq
  notifies :restart, "service[haproxy]"
end
```

65

This is an example of Search and template.

We are using HAProxy but you can pretend it's a Netscaler or an F5

HAProxy Configuration



66

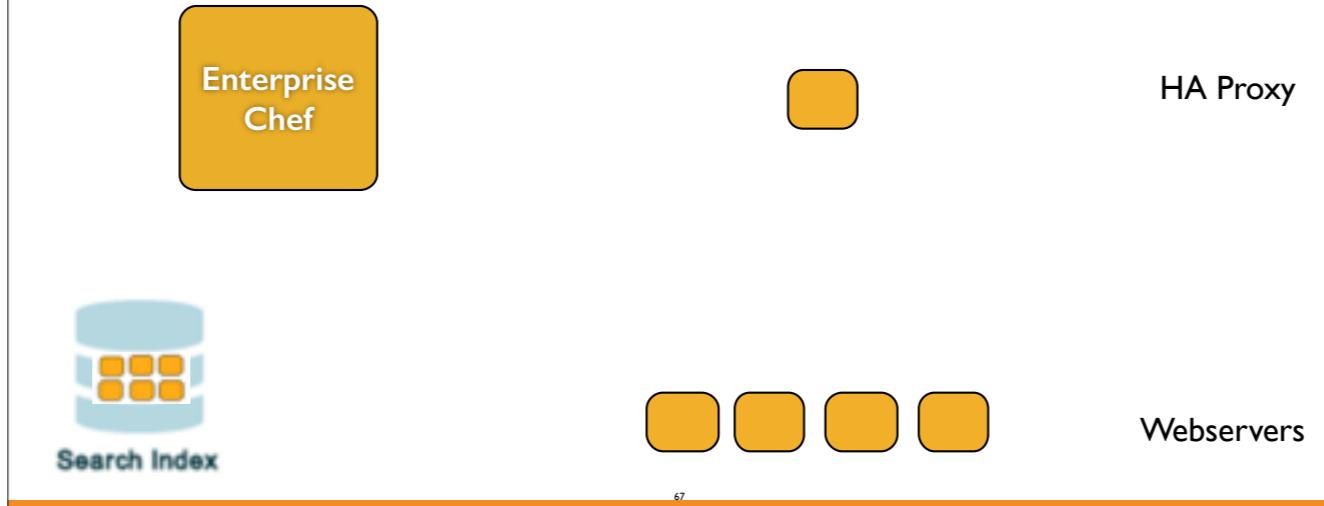
Typical load balancer setup - you can pretend it's a Netscaler or an F5

HA Proxy = High Availability proxy

Some info in case you never seen it: http://support.righscale.com/06-FAQs/FAQ_0101_-_What_is_HAProxy_and_how_does_it_work%3F

HAProxy Load Balancer

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



story:
Chef runs on the proxy server

HAProxy Load Balancer

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



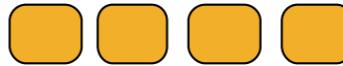
Webservers?



HA Proxy



Search Index

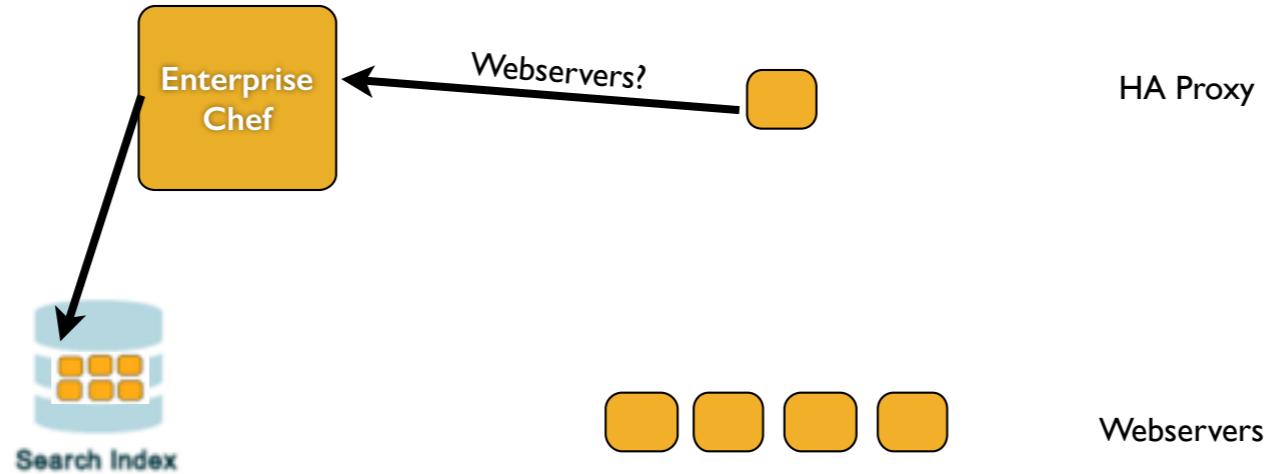


Webservers

68

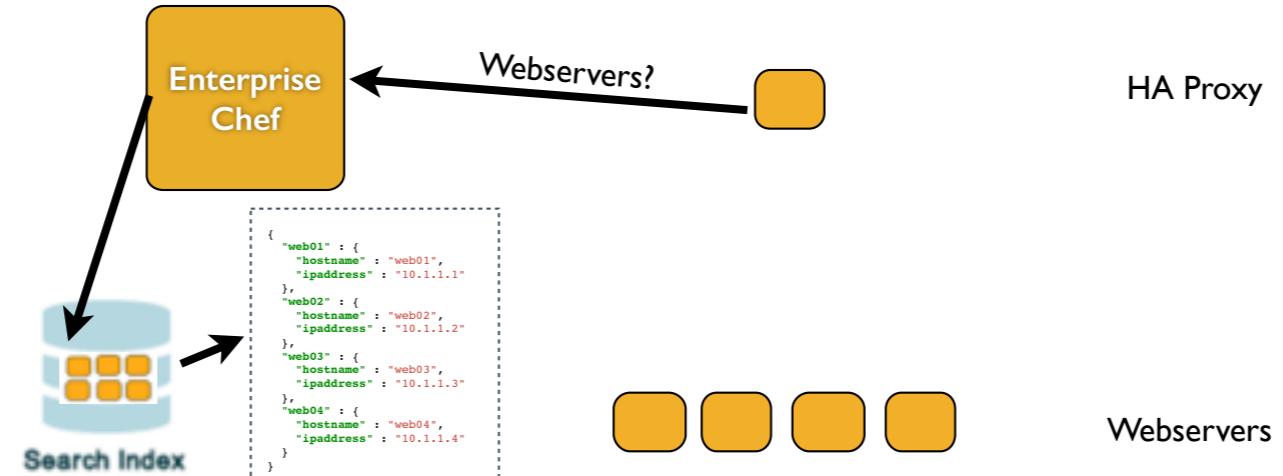
HAProxy Load Balancer

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



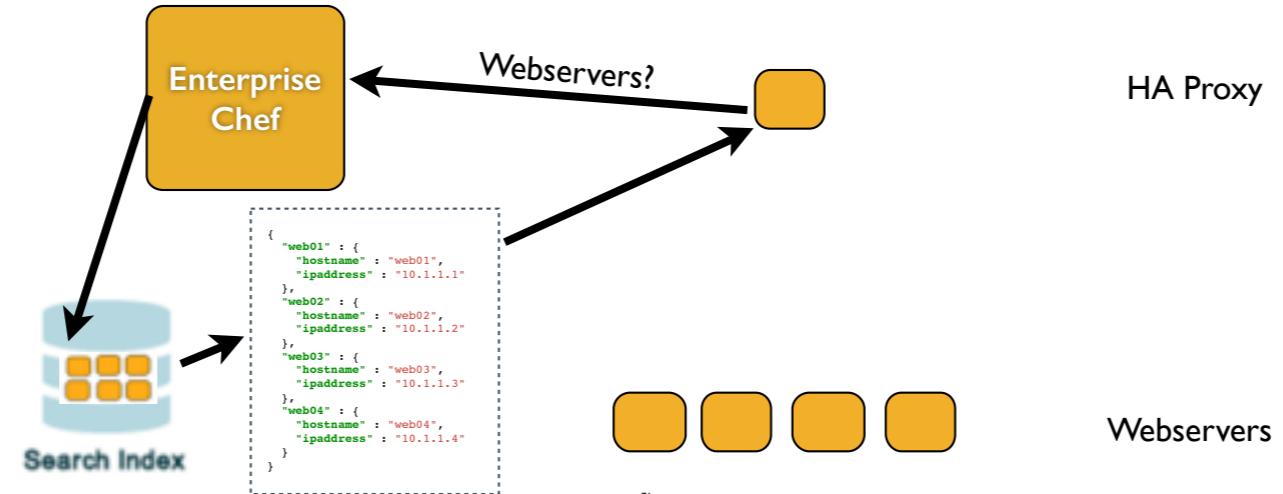
HAProxy Load Balancer

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



HAProxy Load Balancer

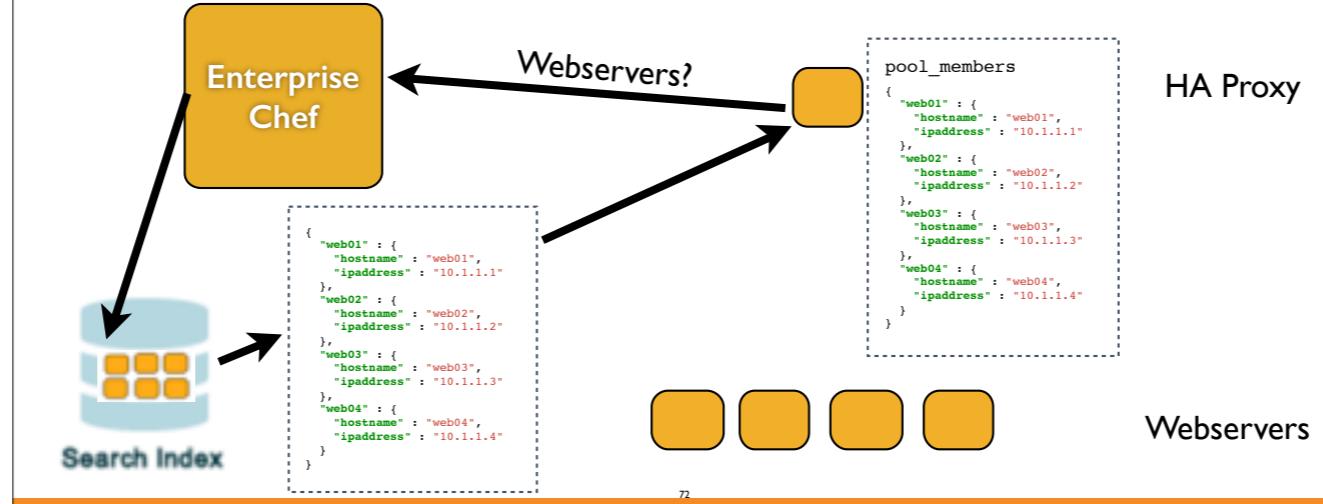
```
pool_members = search("node", "role:webserver")
```



71

HAProxy Load Balancer

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



72

Once the search comes back, we populate the **pool_members** variable

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
```

73

Execute a search against the Chef Server's search index and store the results of that search in the "pool_member" variable.

Pass the unique pool_members to the template as a template variable named pool_members.

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 -%>
```

74

Take a minute to walk through the example:

Templates in Chef are .ERB – Embedded Ruby.

Embedded ruby templates allow combination of code and plain text.

HAProxy Configuration

```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>; weight 1 maxconn 1 check
<% end -%>
```

```
pool_members
```

```
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  }
}
```

HA Proxy



Webservers

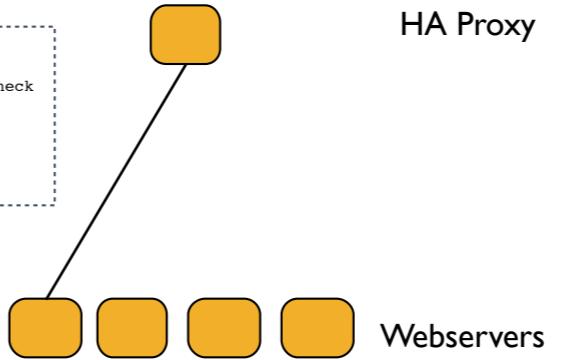
HAProxy Configuration

```
<% @pool_members.each do |member| -%>
  server <%= member[:hostname] %> <%= member[:ipaddress] %>: weight 1 maxconn 1 check
<% end -%>
```

```
pool_members
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  }
}
```

haproxy.cfg

```
server web01 10.1.1.1 weight 1 maxconn 1 check
```



HA Proxy

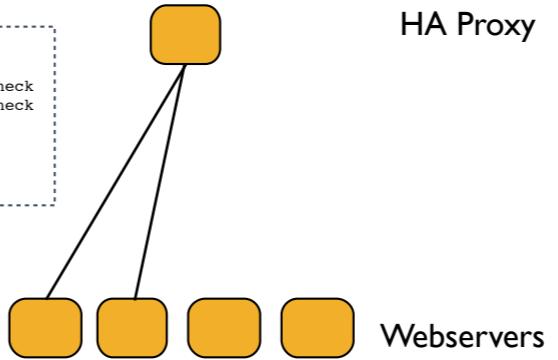
Webservers

HAProxy Configuration

```
<% @pool_members.each do |member| -%>
  server <%= member[:hostname] %> <%= member[:ipaddress] %>: weight 1 maxconn 1 check
<% end -%>
```

```
pool_members
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  }
}
```

```
haproxy.cfg
server web01 10.1.1.1 weight 1 maxconn 1 check
server web02 10.1.1.2 weight 1 maxconn 1 check
```

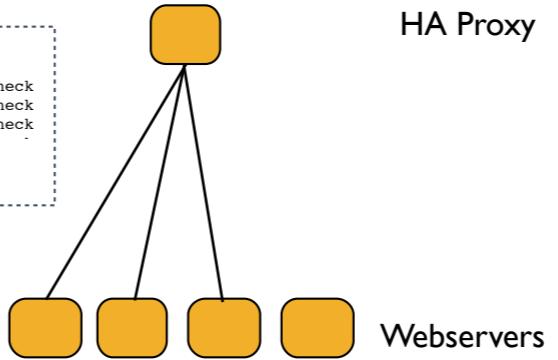


HAProxy Configuration

```
<% @pool_members.each do |member| -%>
  server <%= member[:hostname] %> <%= member[:ipaddress] %>: weight 1 maxconn 1 check
<% end -%>
```

```
pool_members
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  }
}
```

```
haproxy.cfg
server web01 10.1.1.1 weight 1 maxconn 1 check
server web02 10.1.1.2 weight 1 maxconn 1 check
server web03 10.1.1.3 weight 1 maxconn 1 check
```

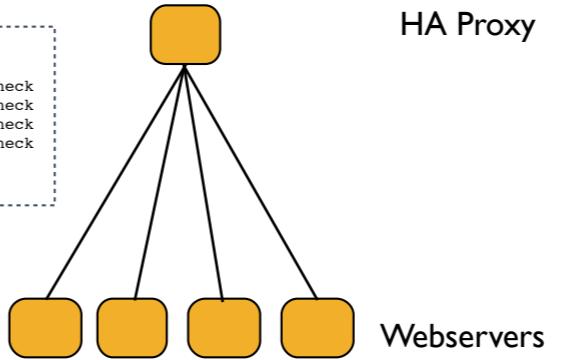


HAProxy Configuration

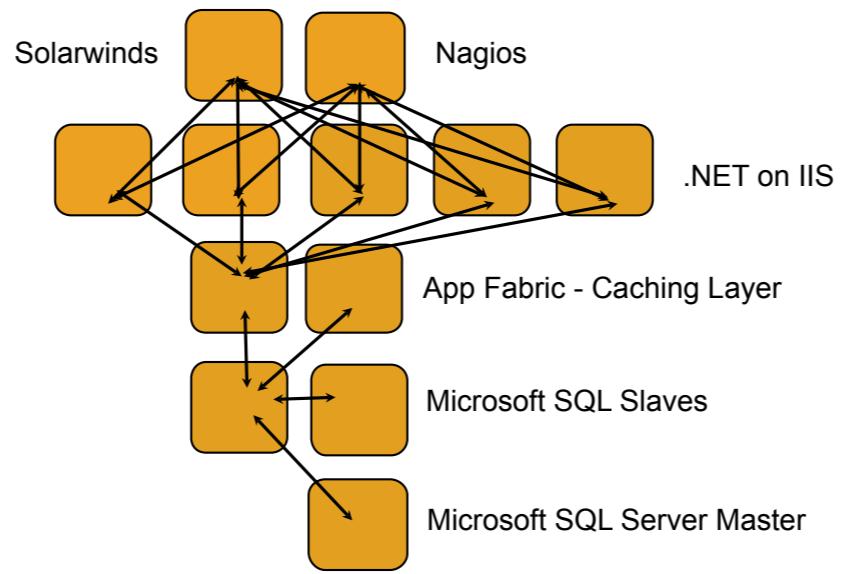
```
<% @pool_members.each do |member| -%>
  server <%= member[:hostname] %> <%= member[:ipaddress] %>: weight 1 maxconn 1 check
<% end -%>
```

```
pool_members
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  }
}
```

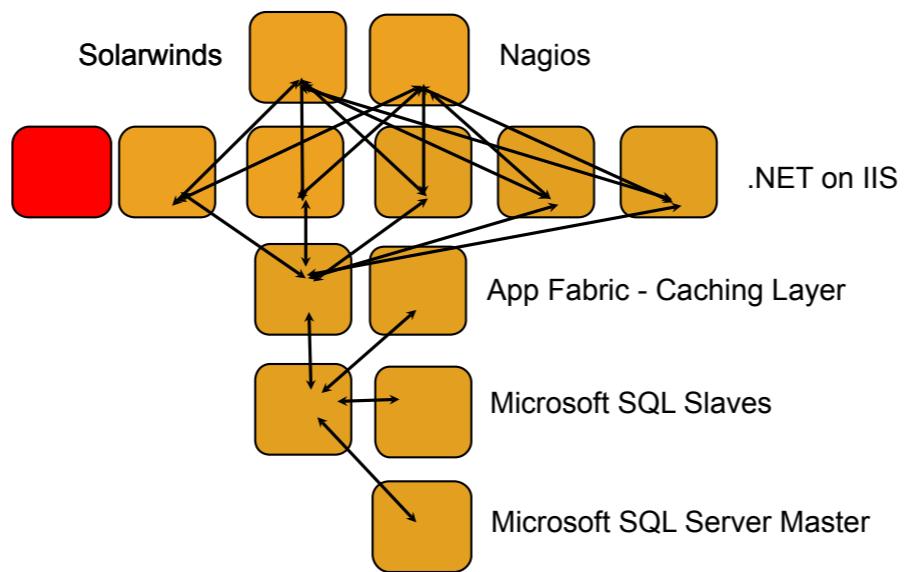
```
haproxy.cfg
server web01 10.1.1.1 weight 1 maxconn 1 check
server web02 10.1.1.2 weight 1 maxconn 1 check
server web03 10.1.1.3 weight 1 maxconn 1 check
server web04 10.1.1.4 weight 1 maxconn 1 check
```



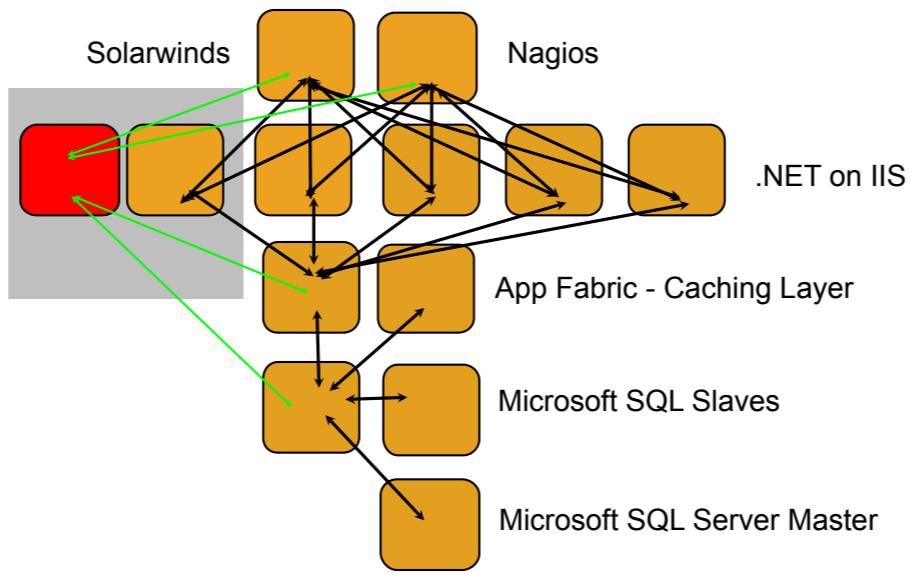
So when this...



...becomes this



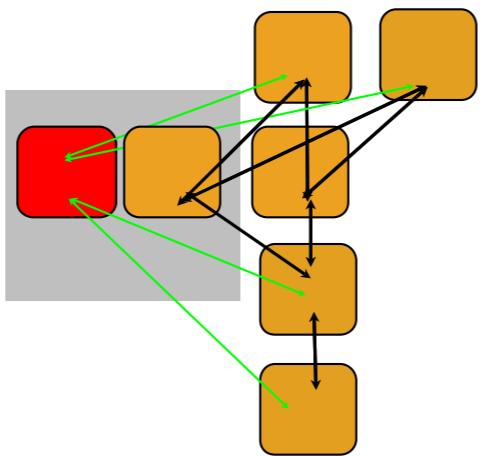
...this can happen automatically



you may get some questions here (about instant notifications or frequency of chef runs, etc..)

The answer is probably push jobs.

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
- Solarwinds CPU
- Solarwinds Memory
- Solarwinds Disk
- Solarwinds SNMP
- DB Cache firewall
- SQL Server firewall
- SQL Server ACL

... 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

I.e. Configuration Drift is when your code is out of date to your deployment needs.

Design Tenets of Chef

- Whippitude - whipping things up quickly
- Reasonability
- Sane defaults
- Flexibility
- Manipulexity - ability to manipulate complex things

What's reasonable for your environment may not be for mine.
On "sane defaults": sanity is in the eye of the beholder

Might add or mention
Principle of least surprise - TIMTOWTDI - There Is More Than One Way To Do It

Design Tenets of Chef

#ChefConf 2012 Blame Larry Wall - I do :)

- Manipulexity: the manipulation of complex things
- Whipuptitude: the aptitude for whipping things up

<http://www.youtube.com/watch?v=bAWjqE5FCxI> (8:40)

From Larry Wall, creator of Perl

manipulexity: You can create anything in C - but not in 20 minutes you cant.

whipuptitude: Using shell script you're done in 5 minutes - until you get to 1000 lines shell script

note: If you've never seen this video, you should watch it. It'll give you a good idea of the company you're working for, and for the students, it'll give them an idea of the reason the tool exists, and how it came about.

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?

A node is a server you want to configure

A resource is an item of manipulation, ie. a primitive that's contained in a recipe.

A recipe is collection of resources

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

v2.1.1_WIN



It's all Windows from here...

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

v2.1.1_WIN



Legend: Do I run that command on my workstation?

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

```
PS \> whoami  
i-am-a-workstation
```

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

```
Remote@PS\> whoami  
i-am-a-chef-node
```

This is a very frequent question in trainings.

Legend: Example Terminal Command and Output

```
PS \> ipconfig
```

```
Windows IP Configuration

Ethernet adapter Local Area Connection:

  Connection-specific DNS Suffix . :
  IPv4 Address . . . . . : 10.38.161.230
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 10.38.161.1
```

Legend: Example of editing a file on your workstation

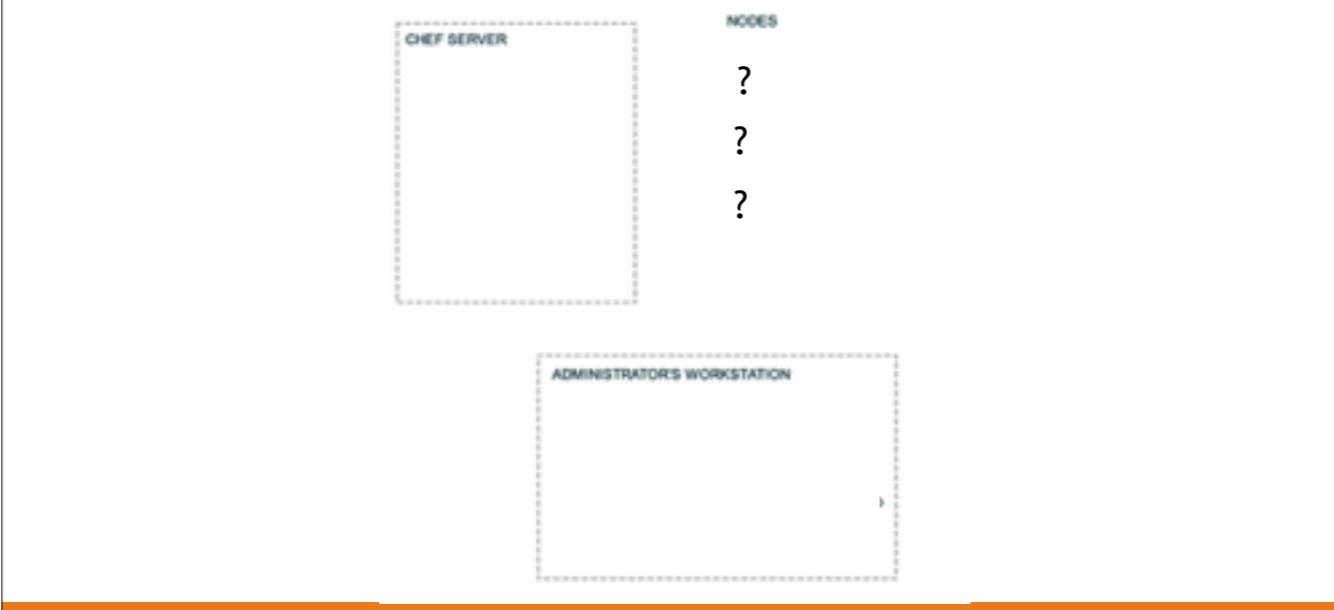
OPEN IN EDITOR: hello_world.txt

Hi!

I am a friendly file.

SAVE FILE!

Landscape of a Chef-managed Infrastructure



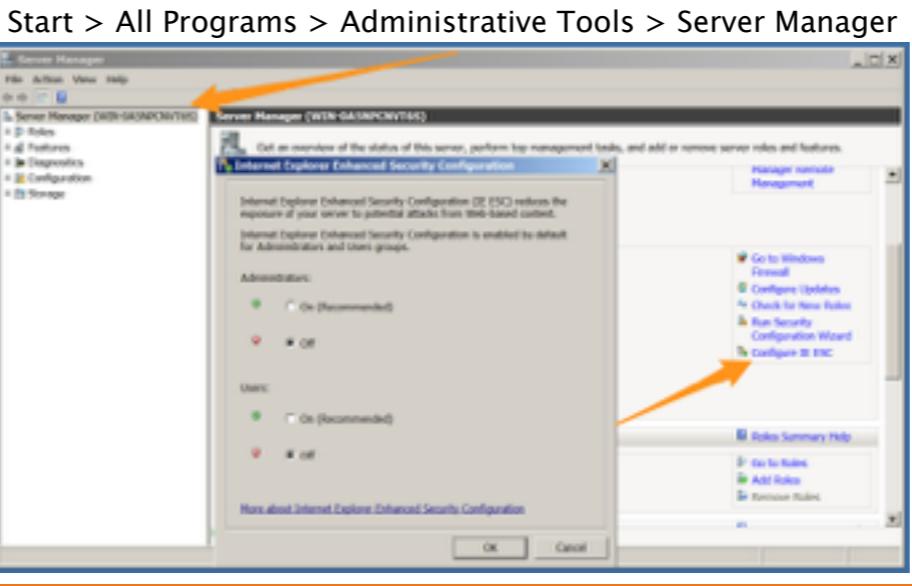
This diagram will be used throughout to help students remember which of the three components we're talking about / working on.

Three main areas

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

Exercise: Configure Management Station

- We're going to need to download a few items, so let's turn off IE Enhanced Security Configuration in Server Manager to make our downloads easier:



Note - this change is only needed if students are using freshly built 2008 or 2012 server as a host machine.

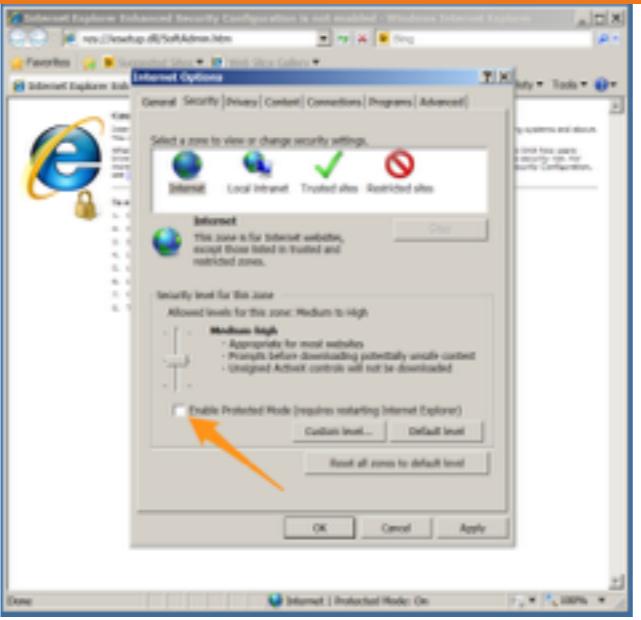
Otherwise, students will see multiple messages prompting them to Add a site to a list of safe sites. A student can be prompted multiple times for just one download, and after the sites are added, the student has to go back and download again. We're turning this off to avoid the hassles.

The Server Manager icon can be found immediately to the right of the Start button. From the resulting screen, click "Configure IE ESC" (bottom right). Set both parameters to Off.

Also available via - Start > All Programs > Administrative Tools > Server Manager

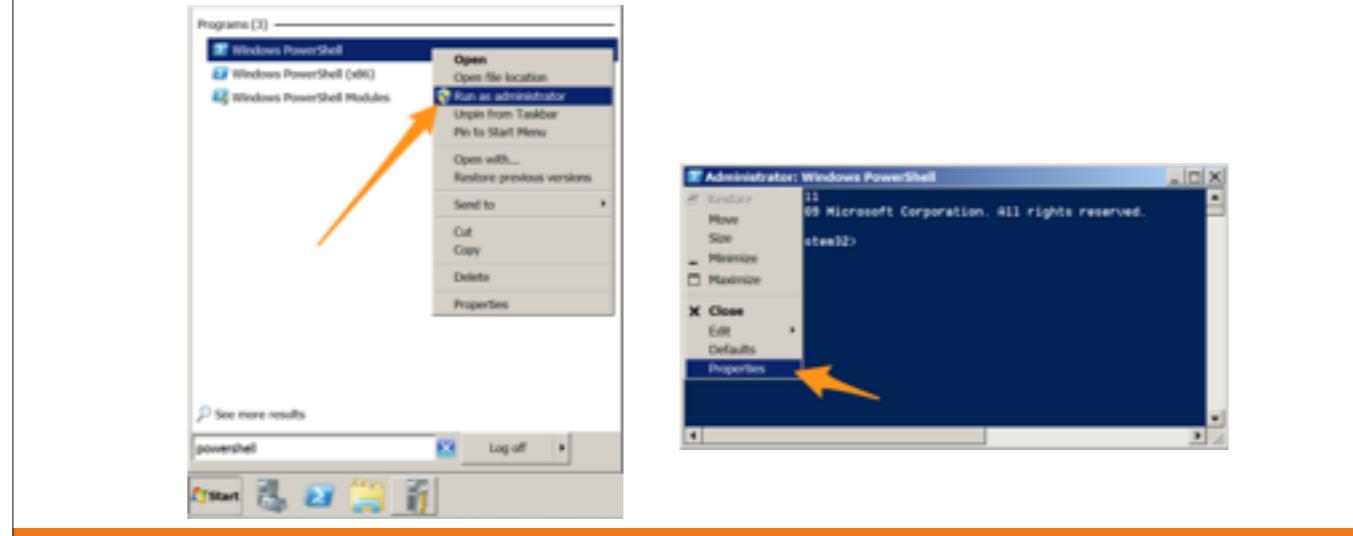
Exercise: Configure Management Station

- Turn off Enabled Protected mode in IE (IE Options --> Tools --> Uncheck “Enable Protected mode”)



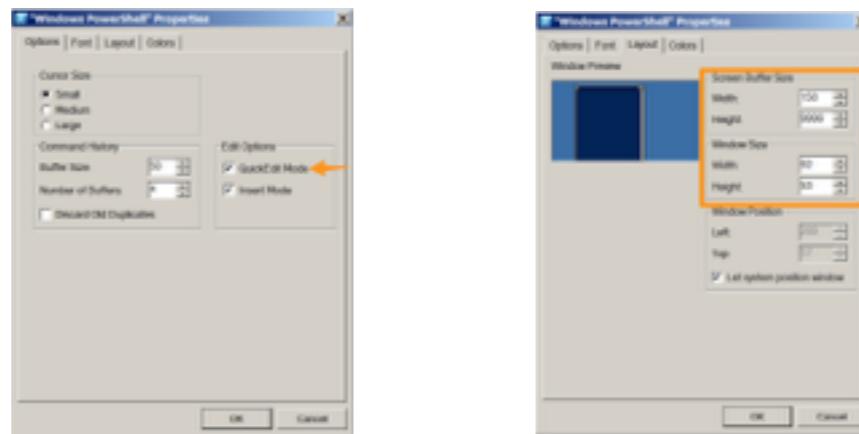
Configure PowerShell

- Run PowerShell as Administrator and open Properties



Configure PowerShell

- Enable QuickEdit Mode
- Set Height buffer to 9999



Workstation Setup

- <http://downloads.chef.io/>
- Select Chef-Client
- 2008 (Windows 7) or 2012 (Windows 8)
- i686 (32-bit) or x86_64 (64-bit)

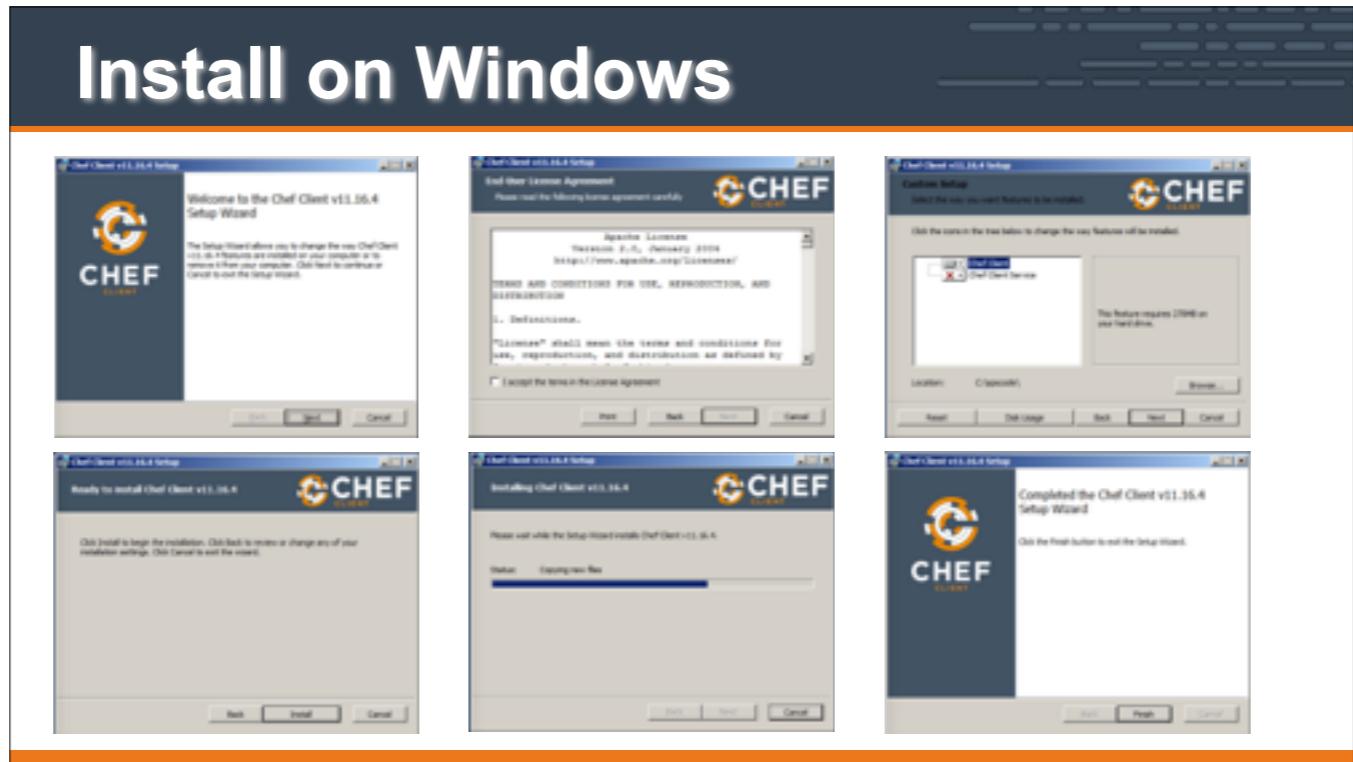
- Or browse to

- <https://www.chef.io/chef/install.msi>



Note: If IE encounters issues, use Chrome or Safari or download from CLI via next slide which is hidden

Install on Windows



Screen shot of Windows installer

Workstation Setup - Mac OS X / Linux

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

```
% Total    % Received % Xferd  Average Speed   Time     Time      Current  
Dload  Upload Total Spent   Left Speed100  6515  100  6515    0     0  
20600      0 ---:--- ---:--- ---:--- 31172Downloading Chef for  
ubuntu...Installing ChefSelecting previously unselected package chef.(Reading  
database ... 47446 files and directories currently installed.)Unpacking chef  
(from .../tmp.MqRJP6lz/chef_amd64.deb) ...Setting up chef (11.4.4-2.ubuntu.  
11.04) ...Thank you for installing Chef!Processing triggers for initramfs-tools  
...update-initramfs: Generating /boot/initrd.img-3.2.0-48-virtual
```

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

What just happened?

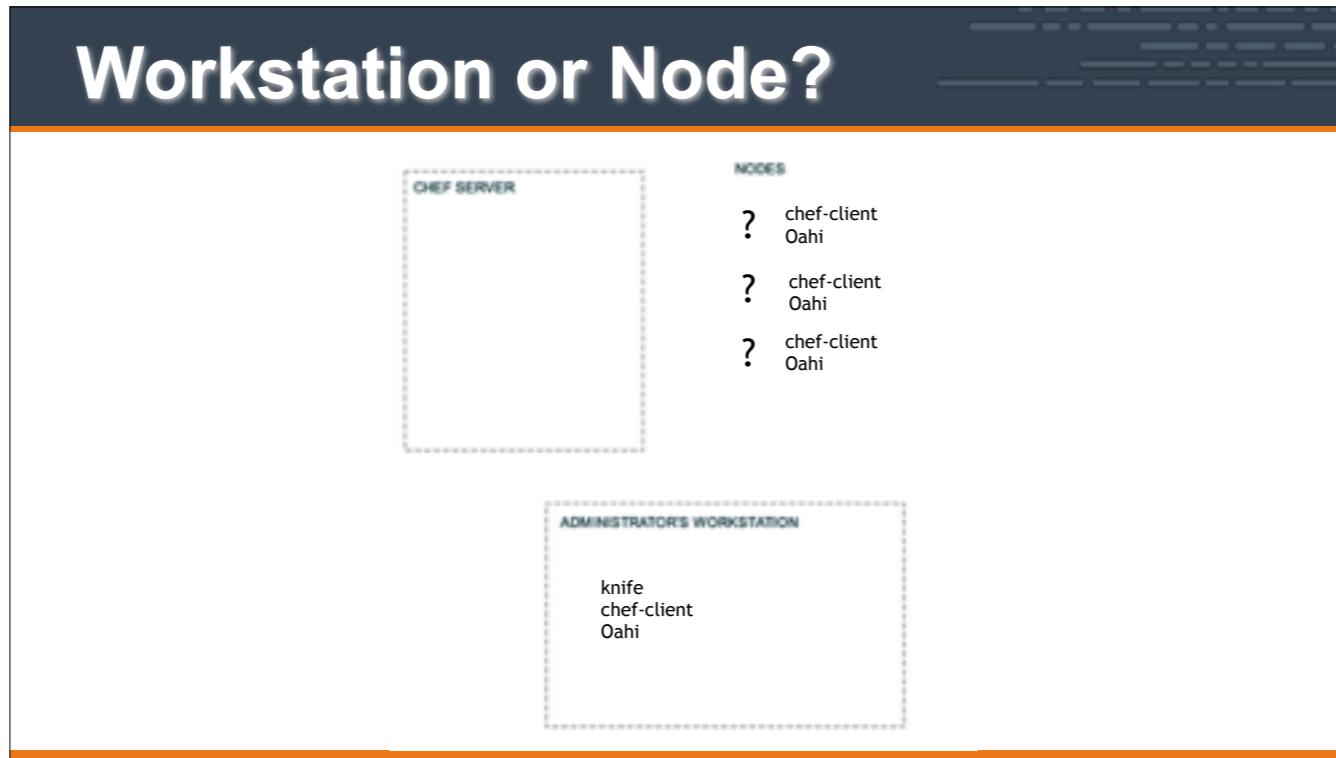
- Chef and all of its dependencies installed via an operating system-specific package ("omnibus installer")
- Installation includes
 - The Ruby language - used by Chef
 - knife - Command line tool for administrators
 - chef-client - Client application
 - ohai - System profiler
 - ...and more

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:\opscode' (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.

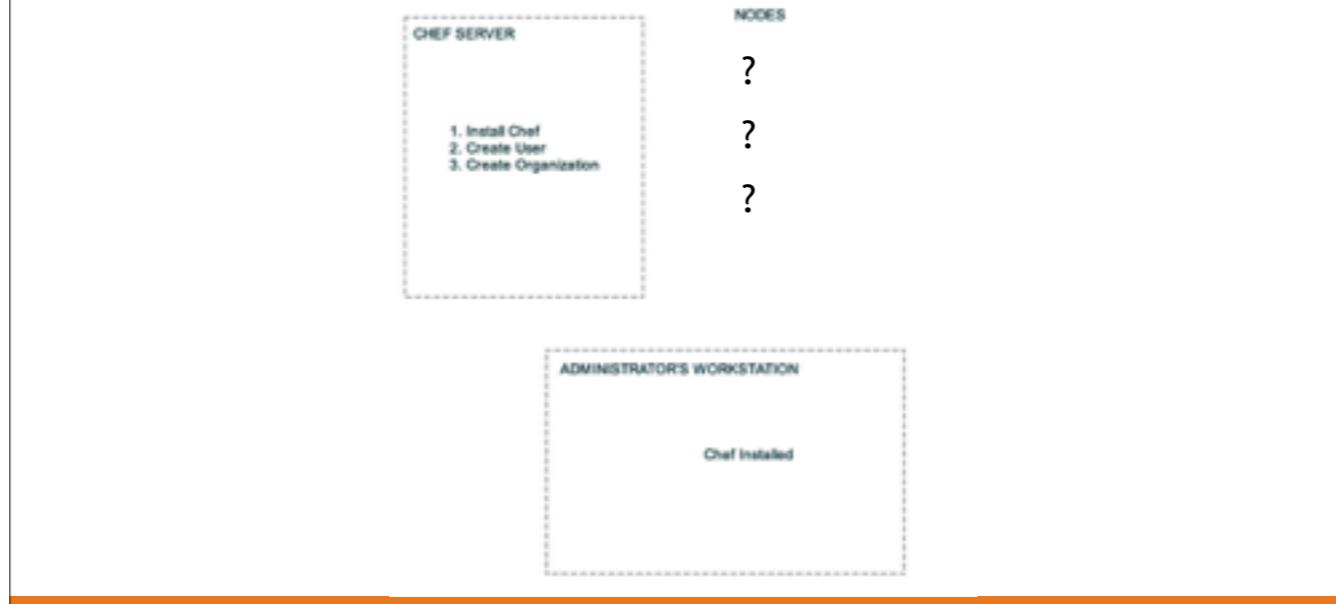
Workstation or Node?



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! Ability to use knife is controlled by ‘master’ certificate – covered a few slides later.

Landscape of a Chef-managed Infrastructure

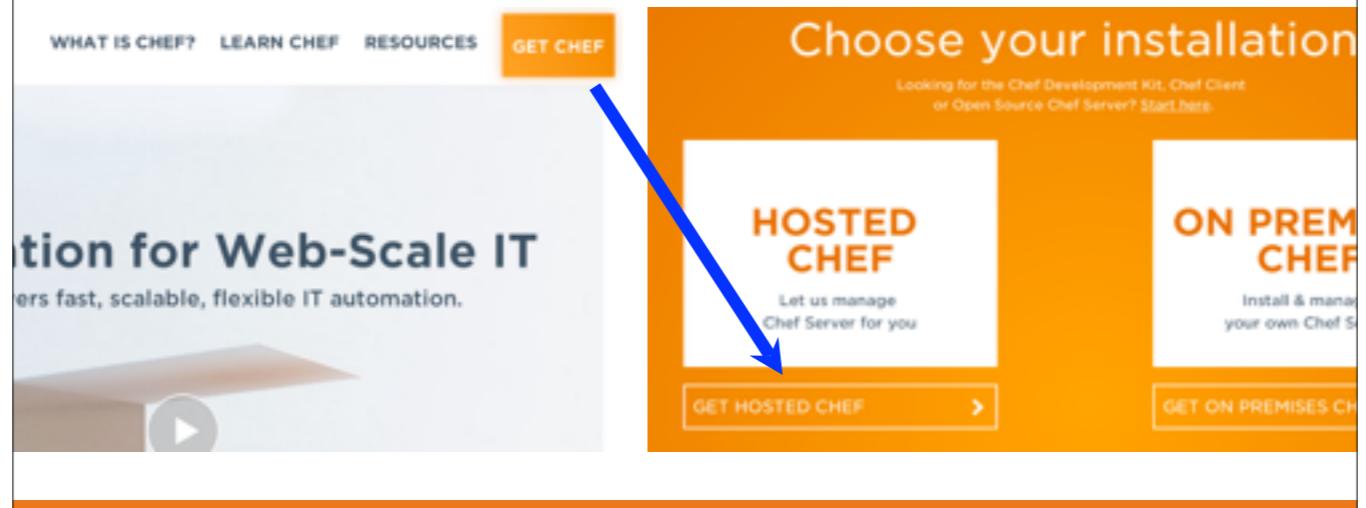


With Chef installed on the workstation, it's now time to get the Chef Server setup.

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

Your Chef Server for this class...

- Hosted Enterprise Chef <http://www.chef.io>



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](#)

NOTE: Chef Organization must have a globally unique value.

NOTE: CREATE A NEW USER, EVEN IF YOU HAVE ONE ALREADY. The Starter Kit will re-create your user auth keypair.

Create new Organization

The screenshot shows the Chef Manage interface. On the left, there is a modal window titled "Create Organization". It has two input fields: "Full Name (example: Chef, Inc.)" with the placeholder "Any name. Doesn't have to be related to where you work." and "Short Name (example: chef)" with the value "i_watch_dora_the_explorer". Below these fields are "Cancel" and "Create Organization" buttons. A blue arrow points from the "Create Organization" button on the right towards the "Create New Organization" button on the main page. On the main page, there is a "Download Starter Kit" button with a blue arrow pointing to it from the bottom right.

Welcome to
CHEF
MANAGE

Thank you for using Chef!

You are not a member of any organizations, so please either create a new organization or accept a pending invitation.

If you are trying to join a specific organization and don't have any invitations, get someone in the organization to send you an invitation, then hit the refresh button.

Sign Out

Create New Organization

Accept Invite (0 pending)

Full Name (example: Chef, Inc.)

Any name. Doesn't have to be related to where you work.

Short Name (example: chef)

i_watch_dora_the_explorer

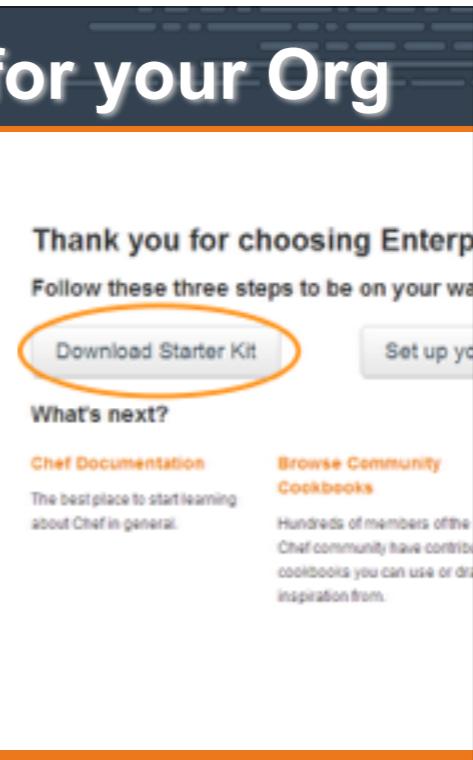
Cancel

Create Organization

Download Starter Kit

Download "Starter Kit" for your Org

- 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:\chef-repo (Win)
 - /Users/you/chef-repo (Mac)
 - /home/you/chef-repo (Linux)
- We'll look at it in detail shortly...



NOTE: CREATE A NEW USER, EVEN IF YOU HAVE ONE ALREADY. The Starter Kit will re-create your user auth keypair.

Download the starter kit from the Chef Server

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

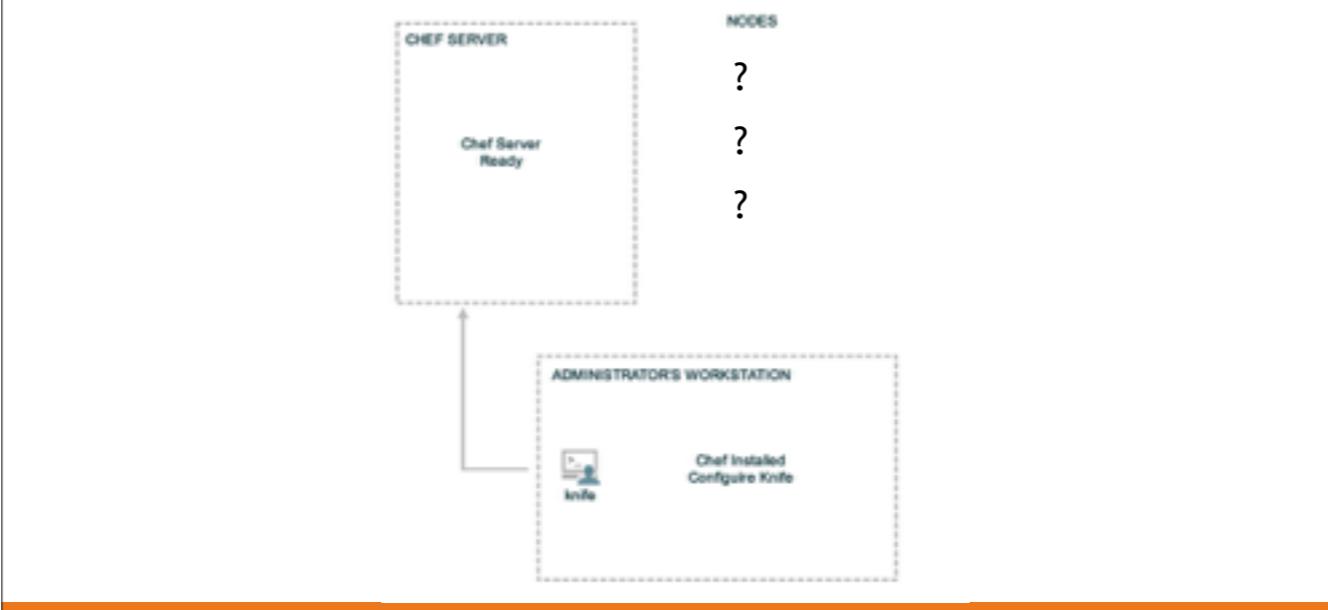
NOTE: Windows XP users should NOT have spaces in their directory- if they put it in a path with spaces (like C:\Documents and Settings\chef-repo), they will get errors when they try to upload cookbooks.

Unzip the zip file in that directory.

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

The 'Starter Kit' is a archive file (e.g. chef-starter.zip) that contains-)
- A sample chef repository with a sample 'starter' cookbook-)
- Configuration files allowing the workstation to talk to the chef server
We'll look at this in detail later...

Landscape of a Chef-managed Infrastructure



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

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

```
PS\> cd SOMEDIR\chef-repo
```

```
PS \chef-repo>
```

A quick tour of the chef-repo

```
PS\> dir
```

Mode	LastWriteTime	Length	Name
----	-----		
d---	9/24/2013 11:09 PM		.berkshelf
d---	9/24/2013 11:09 PM		.chef
d---	9/24/2013 11:09 PM		cookbooks
d---	9/24/2013 11:09 PM		roles
-a---	9/24/2013 11:09 PM	495	.gitignore
-a---	9/24/2013 11:09 PM	1433	Berksfile
-a---	9/24/2013 11:09 PM	588	chefignore
-a---	9/24/2013 11:09 PM	2292	README.md
-a---	9/24/2013 11:09 PM	3572	Vagrantfile

http://docs.opscode.com/essentials_repository.html

For reference

What's inside the .chef directory?

```
PS\> dir .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 is the command-line tool for Chef

- Knife provides an API interface between a local Chef repository and the Chef Server, and 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

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.

knife.rb

- Default location
 - `~/ .chef/knife.rb` (OSX)
 - `c:\Users\You\.chef\knife.rb` (Windows)
- Use a project specific configuration
 - `<project>/ .chef/knife.rb` of the current directory
 - `chef-repo/.chef/knife.rb`
- http://docs.chef.io/config_rb_knife.html

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 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"]
```

<<Review Comment - Don't think this slide was in original Windows version - was it removed for a reason?

Verify Knife

```
PS \> knife --version  
Chef: 11.16.4
```

```
PS \> knife client list  
ORGNAME-validator
```

- Reopen PS for new path to take effect
- Your version may be different, that's ok!



May need to close out of PowerShell then reopen it for chef install changes to take effect & can use knife

knife client list

1. Reads the chef_server_url from knife.rb
2. Invokes HTTP GET to `#{{chef_server_url}}/client`
3. Displays the result



What happens exactly when you execute it?

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

Exercise: Run ‘knife help list’

```
PS \> knife help list
```

```
Available help topics are:  
  bootstrap  
  chef-shell  
  client  
  configure  
  cookbook  
  cookbook-site  
  data-bag  
  environment  
  exec  
  index  
  Knife  
  ...
```

Careful – this doesn’t work on Windows XP

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

Exercise: Run ‘knife client list’

```
PS \> knife client list
```

ORGNAME-validator

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

Dont go forward until everyone is successful.

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

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 – although you could argue that the noun in question is the search index!)

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.

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.

Benefits of a good text editor

- A good editor will
 - Show line numbers
 - Highlight syntax
 - Autocomplete commands
 - Allow you to open multiple files

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



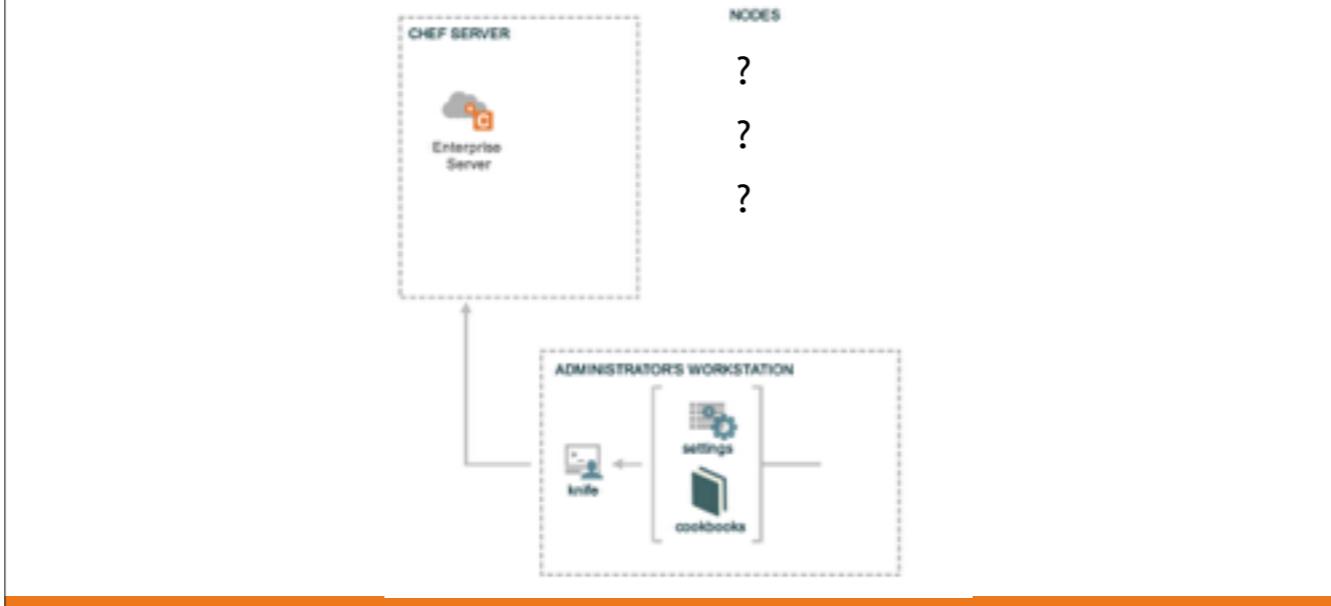
Note: We have them open the 'starter kit' repo in Sublime when writing the Apache cookbook in upcoming section
Point out that Sublime Text is an unrelated tool although it has a chef plugin to help write chef code (autocomplete etc)

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"

Checkpoint



Chef Server is up and running
Knife is installed and configured on the workstation

What's Next?



This is an optional slide! Let students know that in 'real life' they would do it, but for the purpose of the class, we're skipping it.

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

Source Code Repository



This is an optional slide! Let students know that in 'real life' they would do it, but for the purpose of the class, we're skipping it.

We should now create a source code repository for our code.
No more My_script_01, My_script_02, My_script_03 – this isn't 1993

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? :)

-) Directory that contains .chef files, cookbooks, etc.
-) Administrator tool to interact with chef server
-) knife.rb
-) <pwd>/./chef/knife.rb or ~./chef/knife.rb or /etc/chef/knife.rb
-) any answer

Organization Setup

Setup an Organization

v2.1.1_WIN

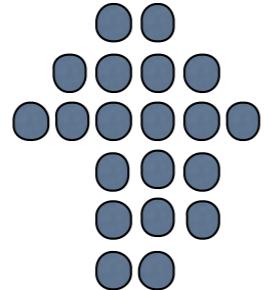


Lesson Objectives

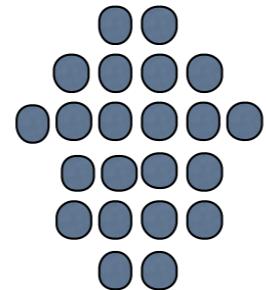
- After completing the lesson, you will be able to
 - Explain the purpose of Organizations
 - Manage your Chef Organization
 - Invite users into your organization
 - Accept invitations into other organizations

Organizations

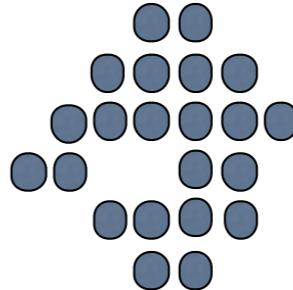
My Infrastructure



Your Infrastructure



Their Infrastructure



Organizations

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

multi-tenancy is a differentiator between paid chef server 11 and free one.

With paid version you have ability to host multiple orgs on one server.

With free you have one org per server

Not a real thing with chef server 12

Manage Organizations

- Login to your Hosted Enterprise Chef (chef.io)
- Select Management Console



Login to Enterprise Chef to view your nodes

Organizations

[Nodes](#)[Reports](#)[Policy](#)[Administration](#)

> Organizations

- [Create](#)
- [Reset Validation Key](#)
- [Generate Knife Config](#)
- [Invite User](#)
- [Leave Organization](#)
- [Starter Kit](#)

Users

Groups

Global Permissions

Showing All Organizations

Organization

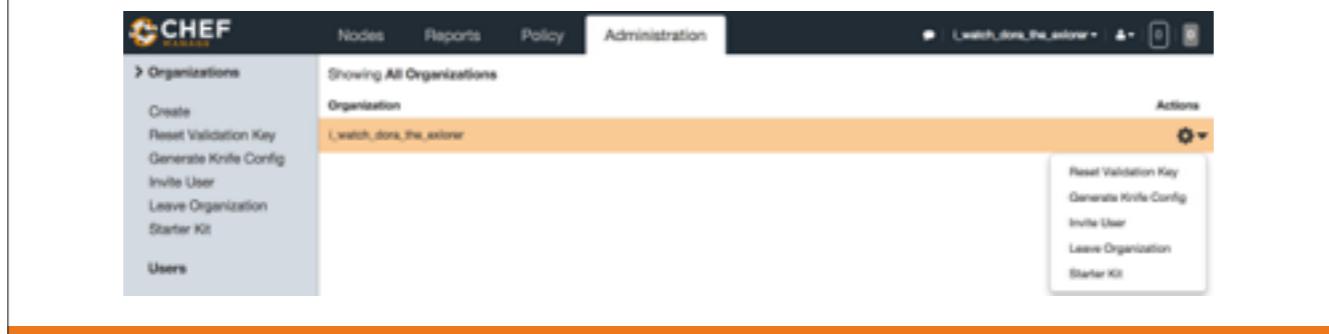
i_watch_dora_the_exlorer

Click the Administrative tab to see the Organizations

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

Manage Organizations

- Reset Validation Key
- Generate Knife Config
- Leave Organization
- Download the 'Starter Kit'



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 Organizations

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

Exercise: Invite users into your org

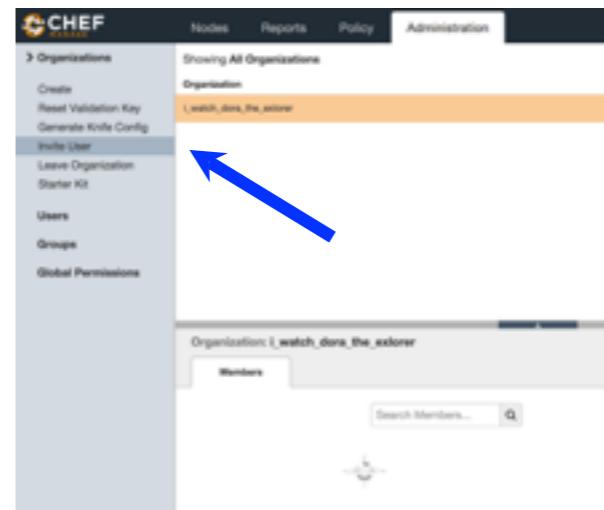
- **The Problem:** You need assistance from a colleague on one of your chef projects
- **Proposed Solution:** Invite the colleague into your organization so they can log into the chef server with **their own** credentials, but can see **your** organization

Exercise: Invite Users into your Org

- For this Lab Exercise you can team up with one other person in the class, then you will
 1. Invite your classmate into your organization
 2. Accept your classmate's invite into their org
 3. Look around your classmate's organization while they look around yours
 4. Finally remove your classmate's access from your account

Exercise: Invite a classmate into your org

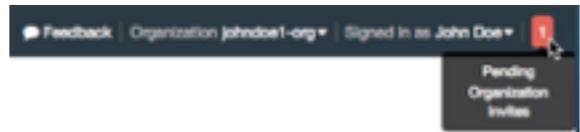
- Navigate to <http://manage.chef.io>
- Click the "Administration" tab,
- Select the appropriate Organization
- Click "Invite User" from the left menu
- Enter your classmate's 'Chef Username' and click Invite



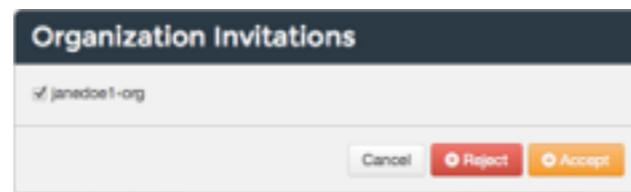
Note: they can get the username from the list of current members of the org.

Exercise: Accept an invite

- You will get a notification once your classmate has invited you into their account



- Click the notification, select the Organization and click 'Accept'



Exercise: View classmate's org

- Select your classmate's organization from the drop down list and peruse their org

The screenshot shows the OpenSUSE Manager interface. On the left, there's a sidebar with links for 'Nodes', 'Reports', 'Policy', 'Administration', 'Feedback', 'OpenSUSE.org', and 'Search in John Doe'. The 'Administration' tab is active. In the center, under 'Organizations', it says 'Showing All Organizations' and lists 'phoebet-org' and 'johndoe1-org'. The 'johndoe1-org' row is highlighted with a yellow background. Below, a modal window titled 'Organization: johndoe1-org' shows a 'Members' section with a table containing one row: 'Name: phoebet' with a 'Remove' button. At the bottom of the modal, there are 'Feedback', 'What's New?', and 'About OpenSUSE Manager' links.

Exercise: Revoke access

- Now either 'Leave Organization' you've been invited into, or remove your classmate from your organization

The screenshot shows the Chef web interface. On the left, there's a sidebar with options like 'Create', 'Reset Validation Key', 'Generate Knife Config', 'Invite User', and 'Leave Organization'. The 'Leave Organization' link is circled in orange. The main area shows a list of organizations: 'johndoe1-org' and 'johndoe2-org'. The 'johndoe2-org' row is also circled in orange. Below this, a modal window is open for 'Organization: johndoe1-org'. It shows a 'Members' section with a table containing one row: 'johndoe1'. To the right of this table is a 'Remove' button, which is also circled in orange.

Review Questions

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

Test Node Setup

Setup a node to manage

v2.1.1_WIN



Lesson Objectives

- After completing the lesson, you will be able to
 - Install chef-client on a node 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



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

Install Windows plugin for Knife

```
PS \> gem install knife-windows
```

```
Successfully installed knife-windows 0.8.2  
19 gems installed
```

Lets Knife know about WinRM protocol

note: if download is SUPER slow, use next (hidden slide to add --no-rdoc --no-ri flags)

ON WINDOWS, TO CHECK WHICH GEM COMMAND IS BEING RESOLVED (if other ruby environs are present)

- In CMD.exe - where gem
- In PowerShell - get-command gem | fl *

If the wrong path is being discovered, either update the PATH environmental variable OR use the fully qualified path.

Nodes

- Nodes represent the servers in your infrastructure
 - Could be physical servers or virtual servers
 - Hardware that you own
 - Compute instances in a public or private cloud
- Could also be network hardware - switches, routers, printers, etc...

network device support for some vendors - Arista, Juniper, Cisco

We Have No Nodes Yet

The screenshot shows the Chef Manage web application. At the top, there's a dark header bar with the title "We Have No Nodes Yet". Below this is a navigation bar with tabs: "Nodes" (which is active), "Reports", "Policy", and "Administration". On the left side, there's a sidebar with a "CHEF MANAGE" logo and 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." The entire interface has a clean, modern design with a white background and a light gray sidebar.

Just to show there aren't any nodes yet

Training Node

- The labs require a node to be managed
- We use the CloudShare training environment
 - Windows Server 2012 x64
 - 2GB RAM
 - 40GB HDD

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

Note: if there are terrible CloudShare issues, you can spin up AWS boxes using Generic 2008 or 2012 server with user_data.ps1. Generic Gist located here - you'll have to modify username/password: <https://gist.github.com/vinyar/d85c145e9dc471358aa6>

CloudShare Node

- Class URL: <Class URL goes here>

CloudShare Node

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



CloudShare Node

1 Environment is ready

Chef Fundamentals 2.x (Windows)

Windows Server 2012

Windows Server 2012 x64 Standard
Description: OS: Windows Server 2012 x64
Spec: 40 GB HD/2 GB RAM
OS: Windows
State: Running

[More details >](#)

View VM

RDP file Reboot VM Revert



155

CloudShare Node

Chef Fundamentals 2.x (Windows)

Windows Server 2012

Windows Server 2012 x64 Standard
Description: OS: Windows Server 2012 x64
Spec: 40 GB HD/2 GB RAM
OS: Windows
State: Running
[Hide details](#)

VM Details

External Address:	uvoloi149k6qBwf7z27.vm.cloudapp.net
Internal IP:	10.160.33.230
Total Memory:	2048 MB
Disk Size:	40 GB
CPU:	1

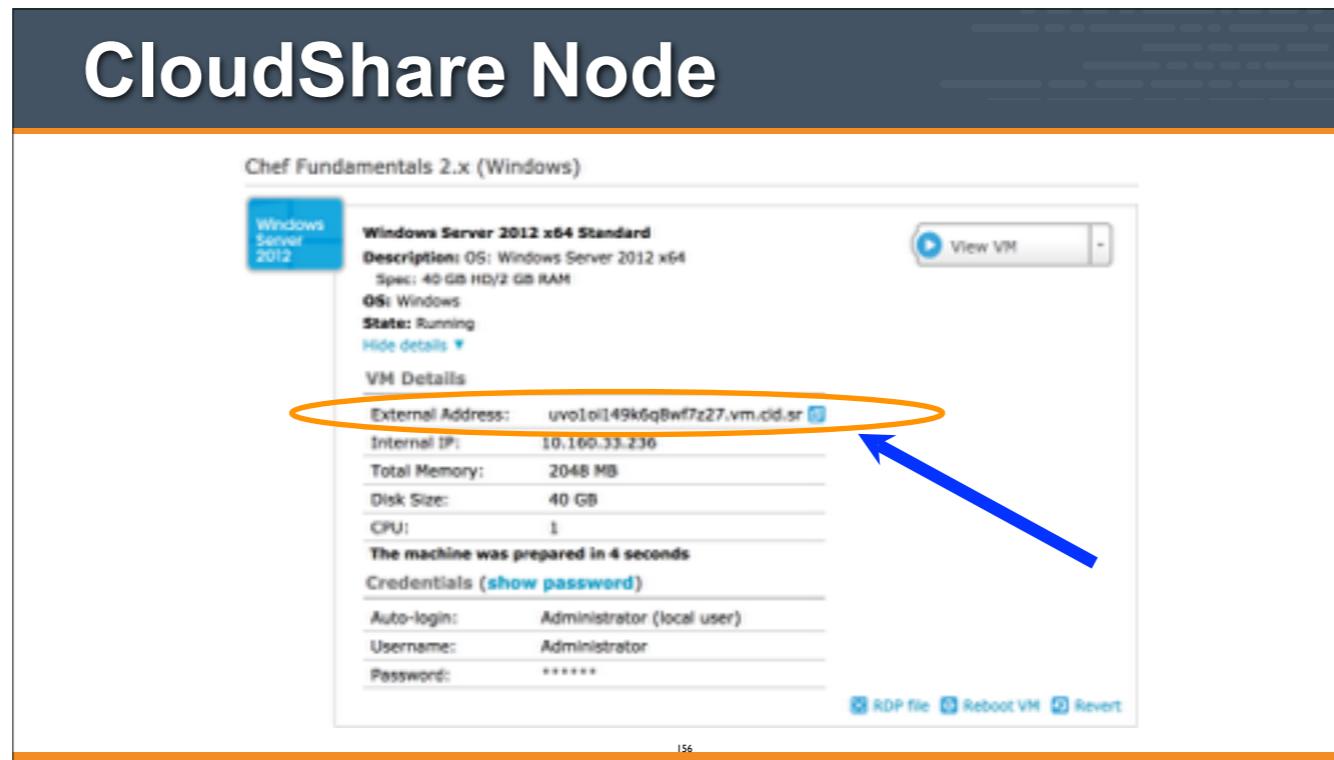
The machine was prepared in 4 seconds

Credentials ([show password](#))

Auto-login:	Administrator (local user)
Username:	Administrator
Password:	*****

RDP file Reboot VM Revert

156



The screenshot shows a VM details page for a Windows Server 2012 instance. The external address is highlighted with an orange oval and a blue arrow points to it from the left.

External Address:	uvoloi149k6qBwf7z27.vm.cloudapp.net
Internal IP:	10.160.33.230
Total Memory:	2048 MB
Disk Size:	40 GB
CPU:	1

The machine was prepared in 4 seconds

Credentials ([show password](#))

Auto-login:	Administrator (local user)
Username:	Administrator
Password:	*****

RDP file Reboot VM Revert

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

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 RDP connection
 - 'RDP' permissions on the VM

Checkpoint



The bootstrap process installs chef-client on a target node so it can communicate with a chef server.

Chef Server is up and running

Knife is installed and configured

Source Code repository has been created

"Bootstrap" the Target Instance

```
PS \> knife bootstrap windows winrm <ETERNAL_ADDRESS> -x chef -P chef -N "node1"
```

```
Bootstrapping Chef on uvo1bv4gjw3ktiwifb3.vm.cld.sr
Enter your password:
uvo1bv4gjw3ktiwifb3.vm.cld.sr "Rendering "C:\Users\chef\AppData\Local\Temp\bootstrap-2000-139299860
uvo1bv4gjw3ktiwifb3.vm.cld.sr Checking for existing directory "C:\chef"...
uvo1bv4gjw3ktiwifb3.vm.cld.sr Existing directory not found, creating.
uvo1bv4gjw3ktiwifb3.vm.cld.sr C:\Users\chef>(
...

```

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.

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

knife bootstrap

```
knife bootstrap windows winrm HOSTNAME -x ...
```



Chef Server



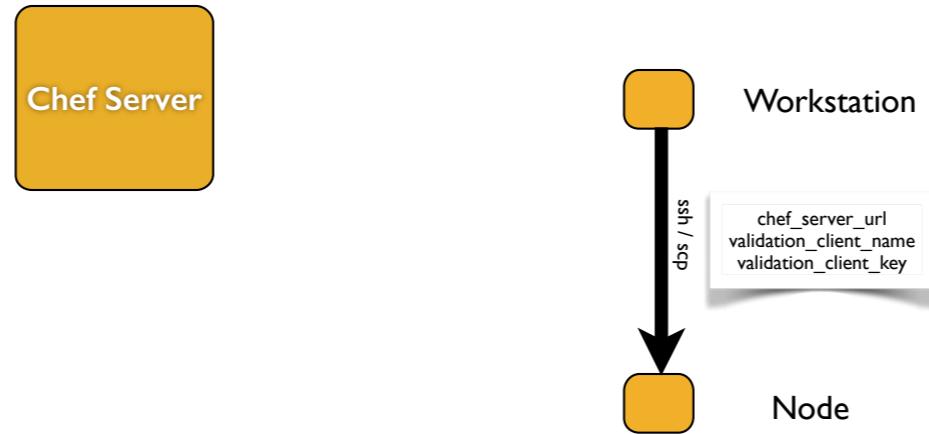
Workstation



Node

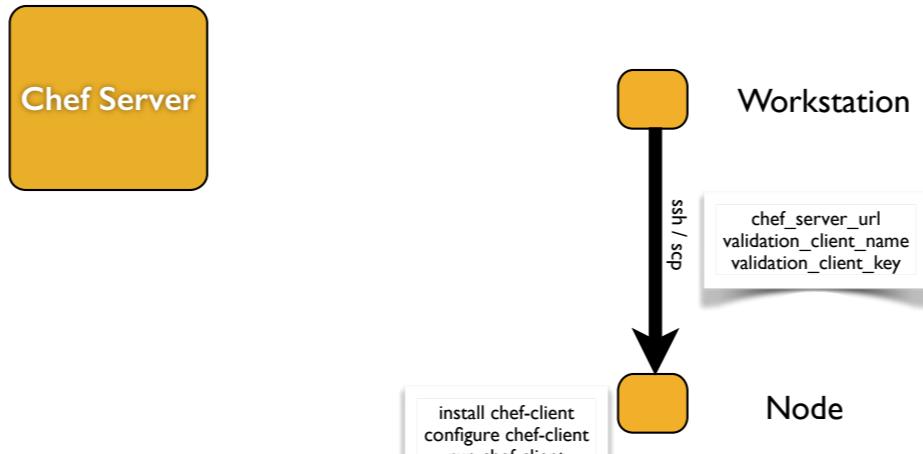
knife bootstrap

```
knife bootstrap windows winrm HOSTNAME -x ...
```



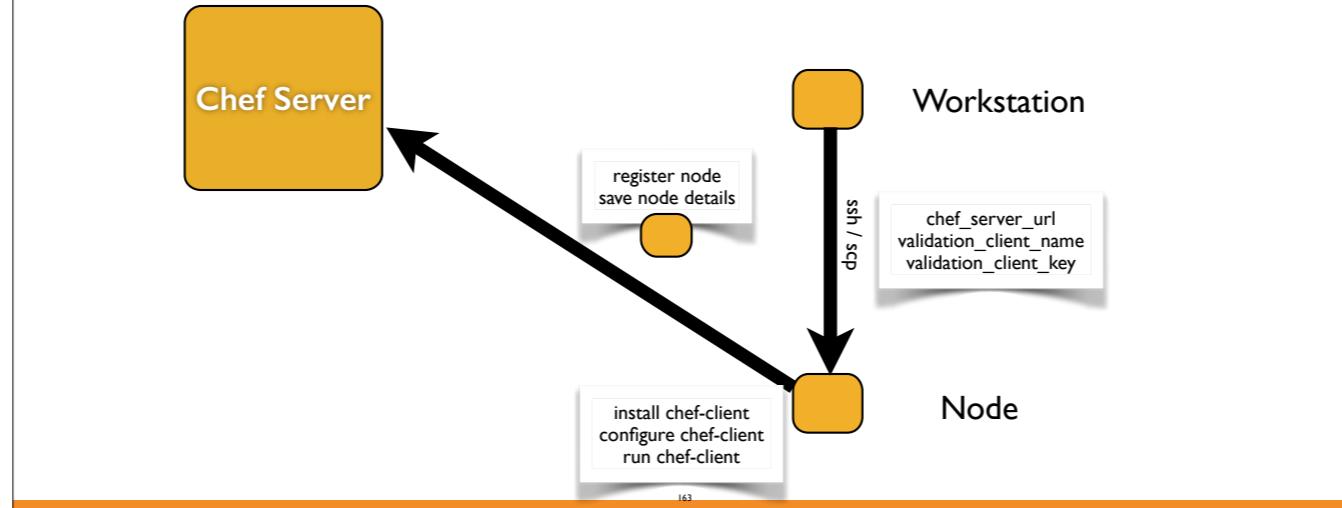
knife bootstrap

```
knife bootstrap windows winrm HOSTNAME -x ...
```



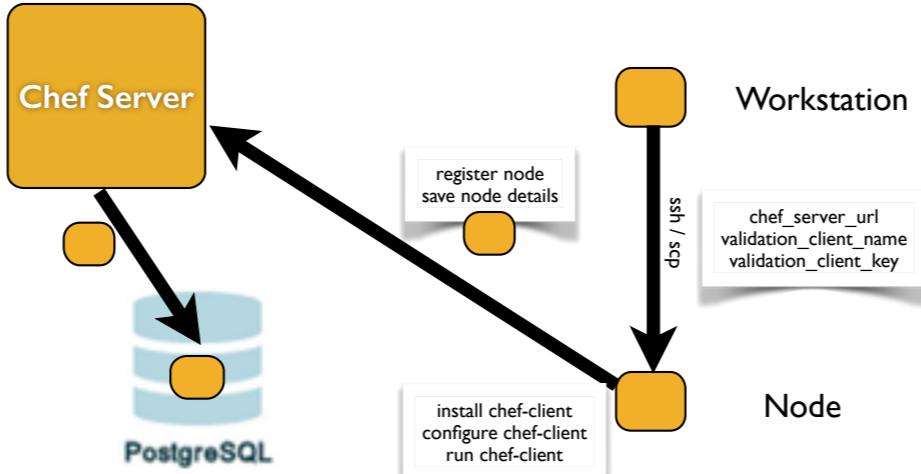
knife bootstrap

```
knife bootstrap windows winrm HOSTNAME -x ...
```



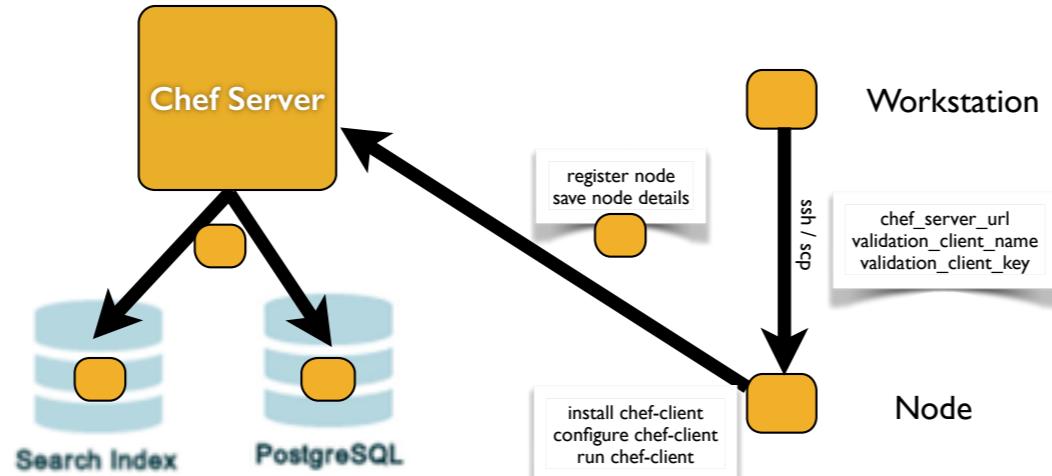
knife bootstrap

```
knife bootstrap windows winrm HOSTNAME -x ...
```



knife bootstrap

```
knife bootstrap windows winrm HOSTNAME -x ...
```



What just happened?

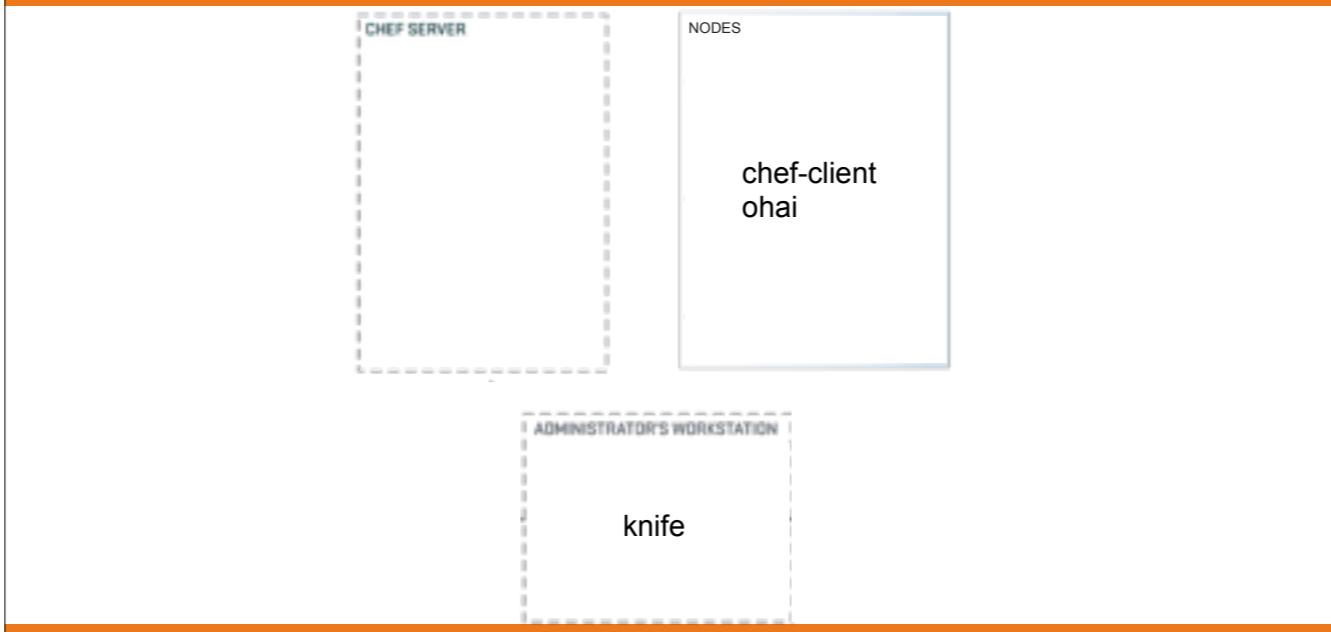
- Chef and all of its dependencies installed via an operating system-specific package ("omnibus installer")
- Installation includes
 - The Ruby language - used by Chef
 - knife - Command line tool for administrators
 - chef-client - Client application
 - ohai - System profiler
 - ...and more

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:\opscode (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.

Review - Workstation or Node?

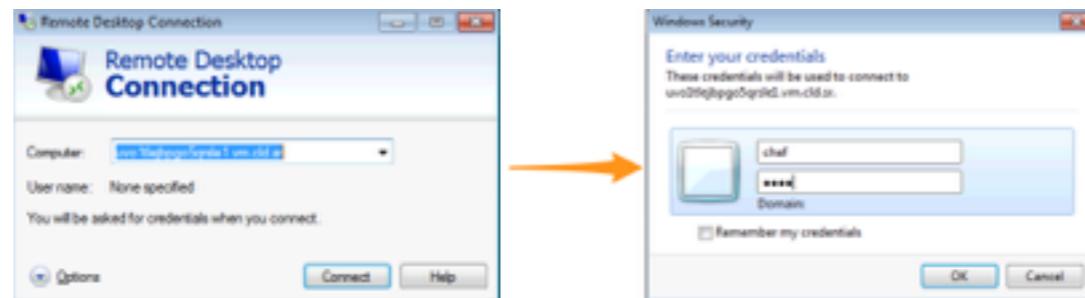


We've just installed chef-client, knife, ohai and ruby along with any supporting ruby gems. This is exactly the same as gets installed on a node during bootstrap. Difference is the node does not use knife, while the workstation client does not use chef-client – but each could be configured to perform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

Also, knife workstation has your 'god' .pem key, and node only has validator.pem and client.pem

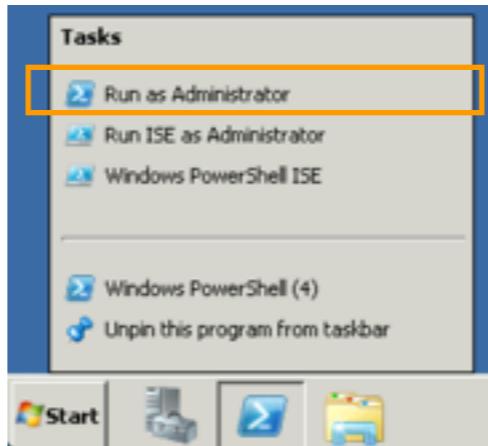
Verify Your Target Instance's Chef-Client is Configured Properly

- Use Remote Desktop to connect to the host.
- User: **chef** Password: **chef**



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

- Open an **Administrator** Powershell on the VM



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

```
remote@PS > dir c:\chef
```

```
Directory: C:\chef

Mode                LastWriteTime      Length Name
----                -----          ----  --
d----
```

Mode	LastWriteTime	Length	Name
d----	10/7/2013 3:41 PM		cache
-a---	10/4/2013 7:18 PM	1702	client.pem
-a---	10/4/2013 7:17 PM	434	client.rb
-a---	10/4/2013 7:17 PM	17	first-boot.json
-a---	10/4/2013 7:17 PM	1706	validation.pem
-a---	10/4/2013 7:16 PM	161	wget.ps1
-a---	10/4/2013 7:16 PM	1273	wget.vbs

Examine C:\chef\client.rb

```
remote@PS > gc c:\chef\client.rb
```

```
log_level      :info
log_location   STDOUT

chef_server_url "https://api.opscode.com/organizations/ORGNAME"
validation_client_name ORGNAME-validator"
client_key      "c:/chef/client.pem"
validation_key   "c:/chef/validation.pem"

file_cache_path "c:/chef/cache"
file_backup_path "c:/chef/backup"
cache_options    ({:path => "c:/chef/cache/checksums", :skip_expires =>
true})

node_name "node1"
```

Verify the log level on your test node

```
remote@PS > gc c:\chef\client.rb
```

```
log_level      :info
log_location   STDOUT

chef_server_url "https://api.opscode.com/organizations/ORGNAME"
validation_client_name ORGNAME-validator"
client_key      "c:/chef/client.pem"
validation_key   "c:/chef/validation.pem"

file_cache_path "c:/chef/cache"
file_backup_path "c:/chef/backup"
cache_options    ({:path => "c:/chef/cache/checksums", :skip_expires =>
true})

node_name "node1"
```

as of chef 11.16.4 this was set automatically. If change is needed use wordpad (NOT notepad). From command line: **start wordpad c:\chef\client.rb**

Examine C:\chef\first-boot.json

```
remote@PS > gc c:\chef\first-boot.json
```

```
{"run_list":[]}
```

- You can bootstrap nodes with recipes/run lists right away
- Disaster recovery!

View Node on Chef Server

- Login to your Hosted Enterprise Chef
 - via Management Council
 - or <https://manage.chef.io>



Sign In

Username

Password

Sign In

Login to Enterprise Chef to view your nodes

View Node on Chef Server

- Click the 'Details' tab

The screenshot shows the Chef Server interface for viewing a node named 'node1'. The top navigation bar includes 'Nodes', 'Reports', 'Policy', and 'Administration'. The left sidebar has links for 'Nodes', 'Delete', 'Manage Tags', 'Reset Key', 'Edit Run List', and 'Edit Attributes'. The main content area displays the node details for 'node1', which is a 'windows' platform with FQDN 'C1263834251' and IP address '10.160.33.236'. The node was last checked in 24 minutes ago and has been up for 44 minutes since 2014-02-21 14:51. The 'Details' tab is highlighted with an orange circle. Other tabs include 'Attributes' and 'Permissions'. Below the tabs, there are sections for 'Tags' (with an 'Add' button) and 'Run List' (with 'Expand All' and 'Collapse' buttons). A message at the bottom states 'There are no items to display.'

View Node on Chef Server

- Click the 'Attributes' tab

The screenshot shows the Chef Server web interface. At the top, there's a navigation bar with tabs for Nodes, Reports, Policy, and Administration. Below this is a table titled "Showing All Nodes" with columns for Node Name, Platform, FQDN, IP Address, Uptime, and Last Checkin. A single node named "node1" is listed, with its row highlighted in orange. Below the table, a specific node is selected: "Node: node1". Under this node, there are three tabs: Details, Attributes (which is highlighted with an orange oval), and Permissions. The main content area is titled "Attributes" and contains a collapsible tree view. The expanded tree shows the following structure:

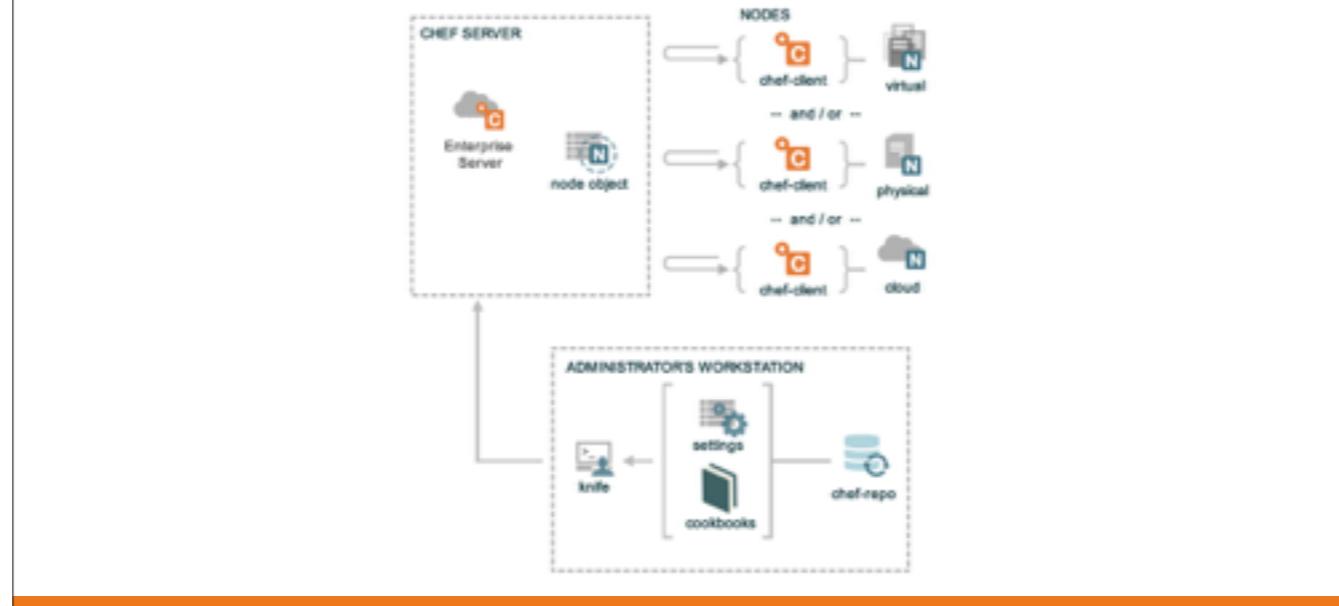
- tags:
 - language
 - kernel
 - os: windows
 - os_version: 6.2.6000
- chef_packages
 - hostname: C1263834251
 - ipaddr: C1263834251
 - domain:
- network
 - interfaces
- cpu

Node

- The node is registered with Chef Server
- The Chef Server displays information about the node
- This information comes from Ohai - we'll see Ohai later.....

We'll be looking at ohai in a later section...

Checkpoint



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?

-) /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 Resources and Recipes

Writing an IIS demo cookbook

v2.1.1_WIN



Lesson Objectives

- After completing the lesson, you will be able to
 - Describe in detail what a **cookbook** is
 - Create a new cookbook
 - Explain what a **recipe** is
 - Describe how to use the `powershell_script`, `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
 - Explain the output of a chef-client run

I always mention – before we start – to **not actually write your own Apache cookbook** and to use the community one. We get to that section later, but explain that we're just doing this as an exercise.

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.

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 IIS
 - Start the service, and make sure it will start when the machine boots
 - Write out the home page
-
- Please note in this course we're teaching Chef primitives, not web server management
 - This is probably not the IIS configuration you would use in production

Exercise: Create a new Cookbook

```
PS \> knife cookbook create iis_demo
```

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

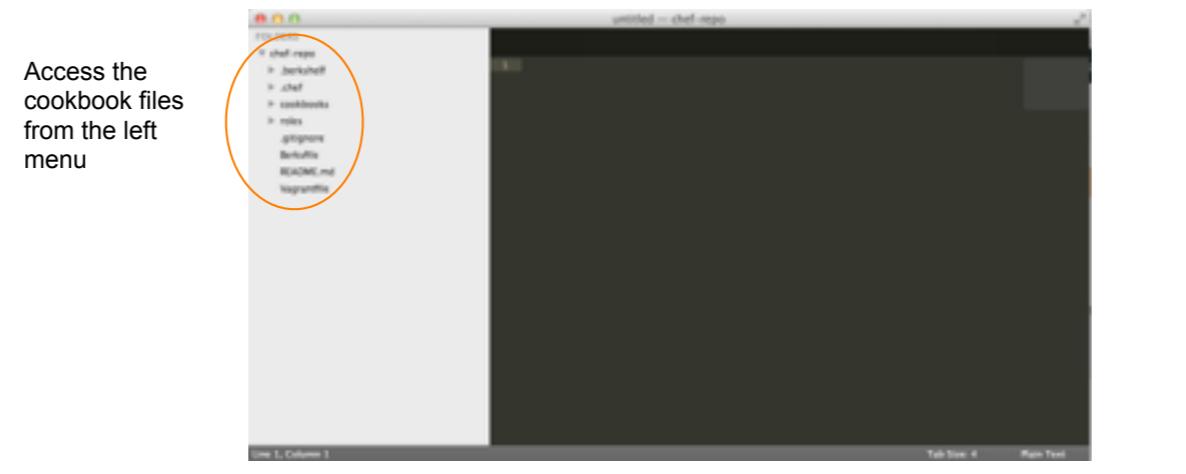
Exercise: Explore the cookbook

```
PS \> dir cookbooks\iis_demo
```

Mode	LastWriteTime	Length	Name
d----	9/5/2013 1:52 PM		attributes
d----	9/5/2013 1:52 PM		definitions
d----	9/5/2013 1:52 PM		files
d----	9/5/2013 1:52 PM		libraries
d----	9/5/2013 1:52 PM		providers
d----	9/5/2013 1:52 PM		recipes
d----	9/5/2013 1:52 PM		resources
d----	9/5/2013 1:52 PM		templates
-a---	9/5/2013 1:52 PM	472	CHANGELOG.md
-a---	9/5/2013 1:52 PM	287	metadata.rb
-a---	9/5/2013 1:52 PM	1531	README.md

Exercise: Open a project drawer if you're using Sublime Text

- If you're using Sublime, then File>Open the chef-repo directory you created earlier



Access the cookbook files from the left menu

If they have their environment set up correctly, they should be able to do "subl ." to open sublime at their current location

Say something like "If you're using Sublime, then....", If they're not using it then gloss over it. In

You can access all files in the cookbook from the left menu

i.e.:C:\Users\you\chef-repo (Win)/Users/you/chef-repo (Mac)/home/you/chef-repo (Linux)

Edit the default recipe

OPEN IN EDITOR: cookbooks\iis_demo\recipes\default.rb

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

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
```

Lets take a moment and talk about resources...

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
```

Exercise: Add a powershell resource to install IIS

OPEN IN EDITOR: cookbooks\iis_demo\recipes\default.rb

```
#  
# Cookbook Name:: iis_demo  
# Recipe:: default  
#  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
  
powershell_script "Install IIS" do  
  code "add-windowsfeature Web-Server"  
  action :run  
end
```

SAVE FILE!

Four concepts while they are typing:

- 1: Strings in ruby are enclosed in single or double quotes. Most of the time in Chef, you will use Double quotes.
- 2: :stop and :delete are Symbols – 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: Arrays are noted by square brackets. This one has two items -- “:stop” and “:delete”. The first item has a trailing comma, the last item does not.
- 4: Idiomatic ruby uses two SPACES for indentation. Sublime defaults to four. At the end of the day, whitespace is whitespace.

So the resource we just wrote...

```
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end
```

So the resource we just wrote...

- Is a **powershell_script** resource

```
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end
```

So the resource we just wrote...

- Is a **powershell_script** resource
- Whose **name** is **Install IIS**

```
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end
```

So the resource we just wrote...

- Is a **powershell_script** resource
- Whose **name** is **Install IIS**
- With code **parameter**

```
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end
```

So the resource we just wrote...

- Is a **powershell_script** resource
- Whose **name** is **Install IIS**
- With code **parameter**
- With a run **action**

```
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end
```

Things with : in front of them (:run) are symbols in ruby - if anyone asks.

Exercise: Add a service resource to ensure the service is started and enabled

OPEN IN EDITOR: cookbooks\iis_demo\recipes\default.rb

```
...
# All rights reserved - Do Not Redistribute
#
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end

service "w3svc" do
  action [ :enable, :start ]
end
```

SAVE FILE!

So the resource we just wrote...

- Is a **service** resource

```
service "w3svc" do
  action [ :enable, :start ]
end
```

So the resource we just wrote...

- Is a **service** resource
- Whose **name** is *w3svc*

```
service "w3svc" do
  action [ :enable, :start ]
end
```

So the resource we just wrote...

- Is a **service** resource
- Whose **name** is **w3svc**
- With two **actions**: **start** and **enable**

```
service "w3svc" do
  action [ :enable, :start ]
end
```

Notice we didn't say how to restart the service

- Resources are **declarative** - that means we say **what** we want to have happen, rather than **how**
- Resources take action through **Providers** - providers perform the how
- Chef determines the **platform** the node is running to determine the correct **provider** for a resource

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.

Service Resource

service "w3svc" ...



starts - **aix**
update-rc.d - **debian**
rc-update - **gentoo**
launchctl - **osx**
svcadm - **solaris**

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

<<Review Comment: Can we have a Windows example for here?

We write "package git", Chef knows which one of these commands to run.

'pacman' is a facet of Arch Linux, & 'pkg_add' is from openBSD & freeBSD

Package Resource

```
package "git"
```



```
yum install git  
apt-get install git  
pacman sync git  
pkg_add -r git
```

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

<<Review Comment: Can we have a Windows example for here?

took commands out of service resource provider

But How does it *really* work?

- Chef is Open Source - All of the source code is on GitHub
- Don't be afraid to Explore
- It's just msieexec under the hood

```
def install_package(name, version)
  ...
  shell_out!("msieexec /qn /i \"#{@new_resource.source}\" ...")
  ...
end
```

Source: <https://github.com/opscode/chef/blob/master/lib/chef/provider/package/windows/msi.rb>

This is an optional slide!

<<WIP: Should this
package_name = "#{name}="#{version}" be this
package_name = "#{name}-#{version}"

<<WIP: Should we introduce a concept of "behind the scenes", where we cover this kind of really techy stuff, nil:NilClass etc

We don't actually write this code in our recipes.

Exercise:**Add a cookbook_file resource to copy the home page in place****OPEN IN EDITOR:** cookbooks\iis_demo\recipes\default.rb

```
...
service "w3svc" do
  action [ :enable, :start ]
end

cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

SAVE FILE!

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

So the resource we just wrote...

```
cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

- Is a **cookbook_file** resource

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...

```
cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

- Is a **cookbook_file** resource
- Whose **name** is `c:\inetpub\wwwroot\Default.htm`

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...

```
cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

- Is a **cookbook_file** resource
- Whose **name** is `c:\inetpub\wwwroot\Default.htm`
- With two **parameters**:
 - **source** of `Default.htm`
 - **rights** of `:read, "Everyone"`

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.

Order Matters

- Resources are executed in order

1st

2nd

3rd

```
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end

service "w3svc" do
  action [ :enable, :start ]
end

cookbook_file 'c:\inetpub\wwwroot
\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

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.

Full contents of the IIS_demo recipe

```
# Cookbook Name:: iis_demo
# Recipe:: default
#
# Copyright 2013, YOUR_COMPANY_NAME
#
# All rights reserved - Do Not Redistribute
#
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end

service "w3svc" do
  action [ :enable, :start ]
end

cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

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

<https://gist.github.com/vinyar/6705367>

for entire correct version

NOTE: Ensure its this

cookbook_file 'c:\inetpub\wwwroot\Default.htm' do

And NOT this

cookbook_file "c:\inetpub\wwwroot\Default.htm" do

Exercise:

Add Default.htm to your cookbook's files/default directory

OPEN IN EDITOR: cookbooks\iis_demo\files\default\Default.htm

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

SAVE FILE!

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 depending on the node's platform
 - Host node name (foo.bar.com)
 - platform-version (windows-6.3)
 - platform-major (windows-6)
 - platform
 - default
- 99% of the time, you will just use **default**

<< TODO: Find a recipe that actually uses this.

Exercise: Upload the cookbook

```
PS \> knife cookbook upload iis_demo
```

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

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.)

Recipe Naming

- Recipes are referenced using the notation '*cookbook::recipe*', where 'recipe' is the name of the '.rb' file in the "cookbook/recipe" directory
- The "default.rb" recipe for a given cookbook may be referred to just by the name of the cookbook (e.g. '*iis_demo*')
- However, if we added another recipe to this cookbook named "createsite.rb", we would refer to it as '*iis_demo::createsite*'

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]`”

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

Exercise: Add iis_demo recipe to node's run list

```
$ knife node run_list add node1 'recipe[iis_demo]'
```

```
node1:
```

```
  run_list: recipe[iis_demo]
```

This is a Microsoft bug - we are working with them, but the process is VERY slow
bug here: <https://tickets.opscode.com/browse/CHEF-2486>

The workaround is to either

a: escape ` with ` (tac in upper left on keyboard) -> ``recipe[iis_demo]``

b: use cmd and single quotes -> 'recipe[iis_demo]'

c: 'role[base],' - add a comma inside single quotes.

Exercise: Run the chef-client on your test node

```
remote@PS\> chef-client
```

```
Starting Chef Client, version 11.10.4
[2014-02-24T05:37:54-08:00] INFO: *** Chef 11.10.4 ***
[2014-02-24T05:37:54-08:00] INFO: Chef-client pid: 260
[2014-02-24T05:38:09-08:00] INFO: Run List is [recipe[iis_demo]]
[2014-02-24T05:38:09-08:00] INFO: Run List expands to [iis_demo]
[2014-02-24T05:38:09-08:00] INFO: Starting Chef Run for node1
[2014-02-24T05:38:09-08:00] INFO: Running start handlers
[2014-02-24T05:38:09-08:00] INFO: Start handlers complete.
resolving cookbooks for run list: ["iis_demo"]
[2014-02-24T05:38:10-08:00] INFO: Loading cookbooks [iis_demo]
Synchronizing Cookbooks:
[2014-02-24T05:38:10-08:00] INFO: Storing updated cookbooks/iis_demo/recipes/
default.rb in the cache.
...
```

Typical errors:

- * The file is not in the 'default' subdirectory
- * The name of the package or service is wrong
- * The site was created during a previous Chef run that eventually aborted due to an error. To resolve the error, remove c:\inetpub\wwwroot\test (and its contents), remove the test web site using the IIS manager, and clear the Chef cache

CLI IIS manager

```
c:\Windows\System32\inetsrv\appcmd.exe stop site site.name:"Test Web Site"
c:\Windows\System32\inetsrv\appcmd.exe start site site.name:"Test Web Site"
```

Exercise: Verify that the home page works

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



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!)

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

Take 5



<http://bestpaperz.com/cl/8810952-funny-coffee-owl-wallpaper.html>

Reading the output of a chef-client run

```
Starting Chef Client, version 11.16.4
[2014-11-23T08:09:34-08:00] INFO: *** Chef 11.16.4 ***
[2014-11-23T08:09:34-08:00] INFO: Chef-client pid: 4996
[2014-11-23T08:09:44-08:00] INFO: Run List is [recipe[iis_demo]]
[2014-11-23T08:09:44-08:00] INFO: Run List expands to [iis_demo]
[2014-11-23T08:09:44-08:00] INFO: Starting Chef Run for node1
[2014-11-23T08:09:44-08:00] INFO: Running start handlers
[2014-11-23T08:09:44-08:00] INFO: Start handlers complete.
```

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

224

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

```
Converging 3 resources
Recipe: iis_demo::default
  * powershell_script[Install IIS] action run[2014-02-24T06:08:29-08:00]
INFO: Processing powershell_script[Install IIS]
  action run (iis_demo::default line 9)

Success Restart Needed Exit Code      Feature Result
-----
True    No          Success      {Common HTTP Features, Default Document, D...
WARNING: Windows automatic updating is not enabled. To ensure that your newly-installed role
or feature is automatically updated, turn on Windows Update.
[2014-02-24T06:09:16-08:00] INFO: powershell_script[Install IIS] ran successfully
```

- `powershell_script` is not idempotent
- Add-WindowsFeature is!

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[w3svc] action enable  
[2014-02-24T06:25:33-08:00]  
INFO: Processing service[w3svc] action enable (iis_demo::default line 14)  
(up to date)  
* service[w3svc] action start  
[2014-02-24T06:25:33-08:00]  
INFO: Processing service[w3svc] action start (iis_demo::default line 14)  
(up to date)
```

- Service is idempotent.
- No changes have been made

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

```
* cookbook_file[c:\inetpub\wwwroot\Default.htm] action create
[2014-02-24T06:25:33-08:00] INFO: Processing cookbook_file[c:\inetpub\wwwroot
\Default.htm] action create (iis_demo::default line 18)
[2014-02-24T06:25:33-08:00] INFO: cookbook_file[c:\inetpub\wwwroot\Default.htm]
created file c:\inetpub\wwwroot\Default.htm

- create new file c:\inetpub\wwwroot\Default.htm[2014-02-24T06:25:33-08:00]
INFO: cookbook_file[c:\inetpub\wwwroot\Default.htm] updated file contents c:
\inetpub\wwwroot\Default.htm

- update content in file c:\inetpub\wwwroot\Default.htm from none to 231d53
  --- c:\inetpub\wwwroot\Default.htm      2014-02-24 06:25:33.000000000 -0800
  +++ C:/Users/chef/AppData/Local/Temp/4/Default.htm20140224-1896-13q72ev
2014-02-24 06:25:33.000000000 -0800
@@ -1 +1,7 @@
+<html>
+  <body>
+    <h1>Hello, world!</h1>
+  </body>
+</html>
+[2014-02-24T06:25:33-08:00] INFO: cookbook_file[c:\inetpub\wwwroot
\Default.htm] permissions changed to [Everyone	flags:0/mask:80000000]

- change dacl
```

- Check for an Default.htm file
- If there is already one in place, backup the file
- A diff of the written file is shown with the modified lines called out
- Set permissions on the file

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

```
[2014-02-24T06:25:34-08:00] INFO: Chef Run complete in 17.432572377 seconds
[2014-02-24T06:25:34-08:00] INFO: Running report handlers
[2014-02-24T06:25:34-08:00] 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

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.

Idempotence

- Actions on resources in Chef are designed to be **idempotent**
- In practical terms, this means they only change the state of the system if they have to
- If a resource in Chef is properly configured, we move on to the next resource

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.

Good example is:

```
echo 'bla' > file1.txt - is idempotent
```

convergent: if context of file1.txt is 'bla' do nothing,
else echo 'bla' > file1.txt

Now imagine if you have to install a package on 50'000 servers. Forcing package to always install will do the job. However –“is package already installed” is way cheaper at scale.

Idempotence

- Actions on resources in Chef are designed to be ***idempotent***
 - I.e. they can be applied multiple times but the end result is still the same - like multiplying by 1 in math!
 - Or in mathematical terms, $F(F(x))=F(x) \forall x$
- Chef is a "desired state configuration" system - if a resource is already configured, no action is taken. This is called ***convergence***

(slide for 'math nerds') - notes

<<WIP/Review Comment: On page 5 of the book "Test-Driven Infrastructure with Chef", Convergence is explained as "Services should take responsibility for their own state being in line with policy; over time the overall system will tend to correctness." – does this mean the same thing as the last bullet here??

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.

$F(F(x))=F(x) \forall x$ is read as "the F of the F of x equals the F of x for all x" – Mathematicians will understand, and they'll 'get' chef!

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

Relevant bits from the Hipchat discussion on this slide -

Adam Jacobs - Function idempotenc says that if the function gets the same inputs, it provides the same outputs. That works

Sean OMeara: ^ the word idempotent means one thing and one thing exactly.

Sean OMeara: convergent is the test-and-repair bit... which is (usually) idempotent

"Convergence goes beyond idempotence. Doing something once only is not enough. It has to end in the correct state." - Mark Burgess

Exercise: Re-run the Chef Client

```
remote@PS\> chef-client
```

```
Converging 3 resources
Recipe: iis_demo::default
  * powershell_script[Install IIS] action run[2014-02-24T06:42:15-08:00] INFO: Processing powershell_script[Install IIS]
    action run (iis_demo::default line 9)

  Success Restart Needed Exit Code      Feature Result
  -----  -----  -----  -----
  True     No        NoChangeNeeded {}

[2014-02-24T06:42:16-08:00] INFO: powershell_script[install IIS] ran successfully

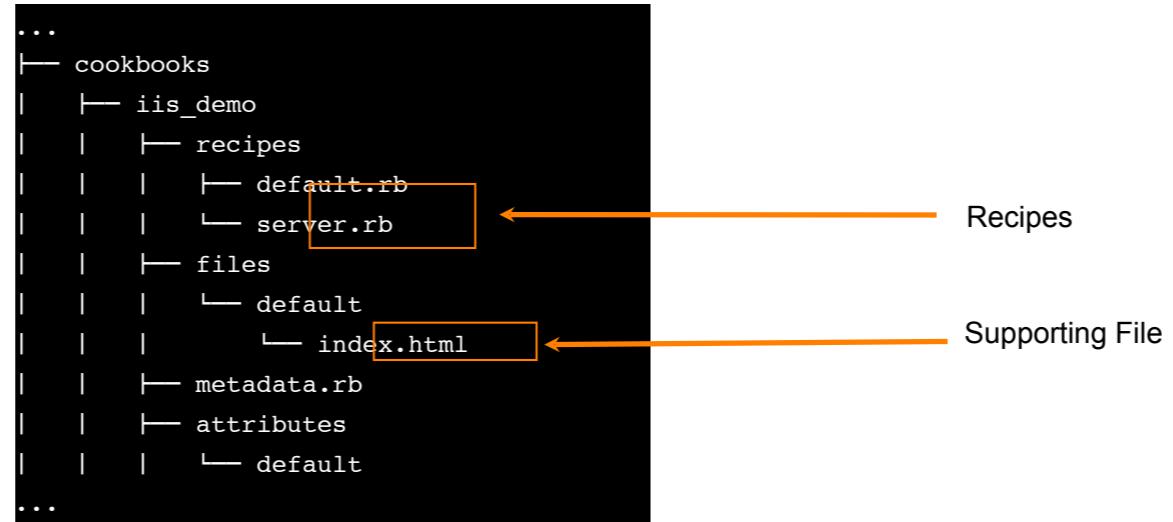
  - execute "powershell.exe" -NoLogo -NonInteractive -NoProfile -ExecutionPolicy RemoteSigned -InputFormat None -File
  "C:/Users/chef/AppData/Local/Temp/4/chef-script20140224-4608-1fwjxq.ps1"
    * service[w3svc] action enable[2014-02-24T06:42:16-08:00] INFO: Processing service[w3svc] action enable (iis_demo::default
    line 14)
    (up to date)
    * service[w3svc] action start[2014-02-24T06:42:16-08:00] INFO: Processing service[w3svc] action start (iis_demo::default
    line 14)
    (up to date)
    * cookbook_file[c:\inetpub\wwwroot\Default.htm] action create[2014-02-24T06:42:16-08:00] INFO: Processing cookbook_file[c:
    \inetpub\wwwroot\Default.htm] action create (iis_demo::default line 18)
    (up to date)
[2014-02-24T06:42:17-08:00] INFO: Chef Run complete in 3.416521 seconds
...
```

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

Recap - So resources are grouped into recipes

```
recipe[iis_demo::default] {  
    powershell_script "Install IIS" do  
        action :run  
        code "add-windowsfeature Web-Server"  
    end  
  
    service "w3svc" do  
        action [ :enable, :start ]  
    end  
  
    cookbook_file 'c:\inetpub\wwwroot\Default.htm' do  
        source "Default.htm"  
        rights :read, "Everyone"  
    end  
}
```

Cookbooks contain recipes and supporting files

```
...  
├ cookbooks  
│ └ iis_demo  
│   ├ recipes  
│   |   ├ default.rb  
|   |   └ server.rb  
|   └ files  
|     └ default  
|       └ index.html  
|   └ metadata.rb  
|   └ attributes  
|     └ default  
...  

```

Recipes

Supporting File

Cookbooks are installed as artifacts on Chef Server

Enterprise
Chef
(org)

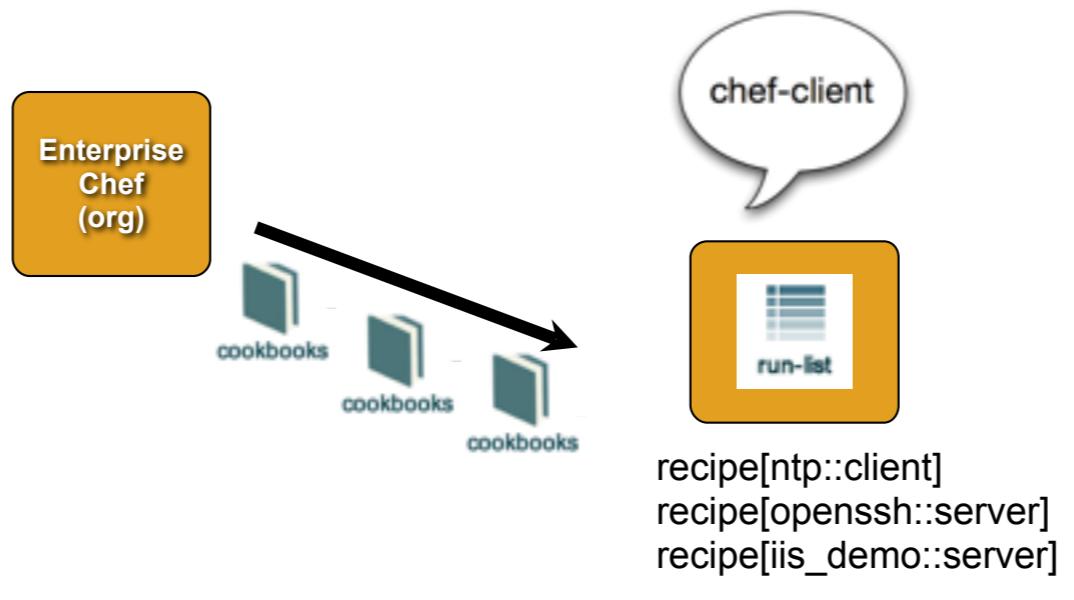


iis_demo v0.1.0

ntp v0.2.0

openssh v2.4.7

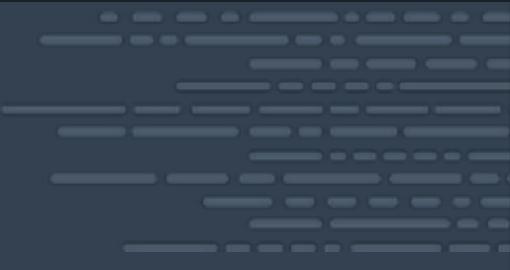
Nodes have run_lists made of recipes



Questions

- What makes up 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?

-) Like a package that contains 'all' that is needed (recipes, files, templates, etc.)
-) knife cookbook create <name>
-) Ordered list of tasks to execute upon resources.
-) A chef element (in a recipe) that aligns to an item of manipulation on the node
-) knife cookbook upload <name>
-) Ordered set of recipes to execute on node



Dissecting your first chef-client run

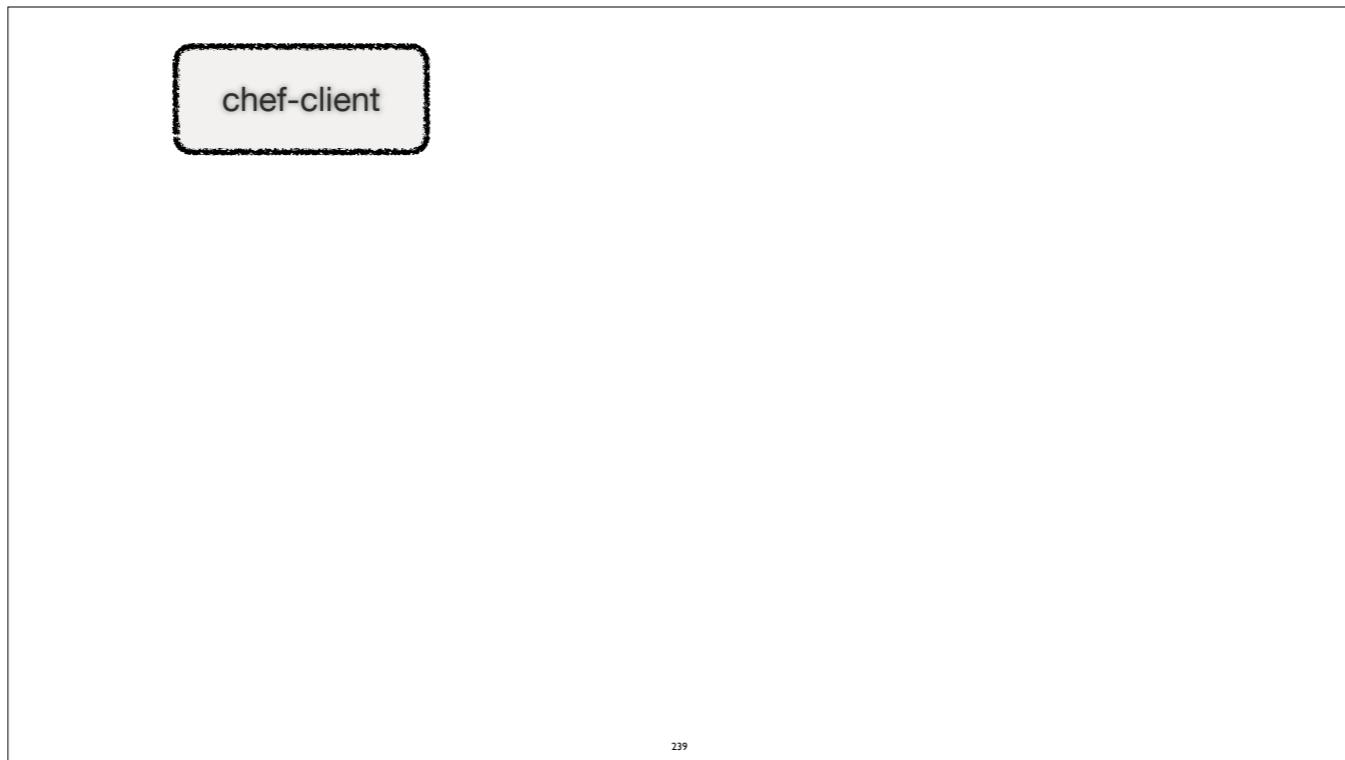
The Anatomy of a Chef run

v2.1.1_WIN



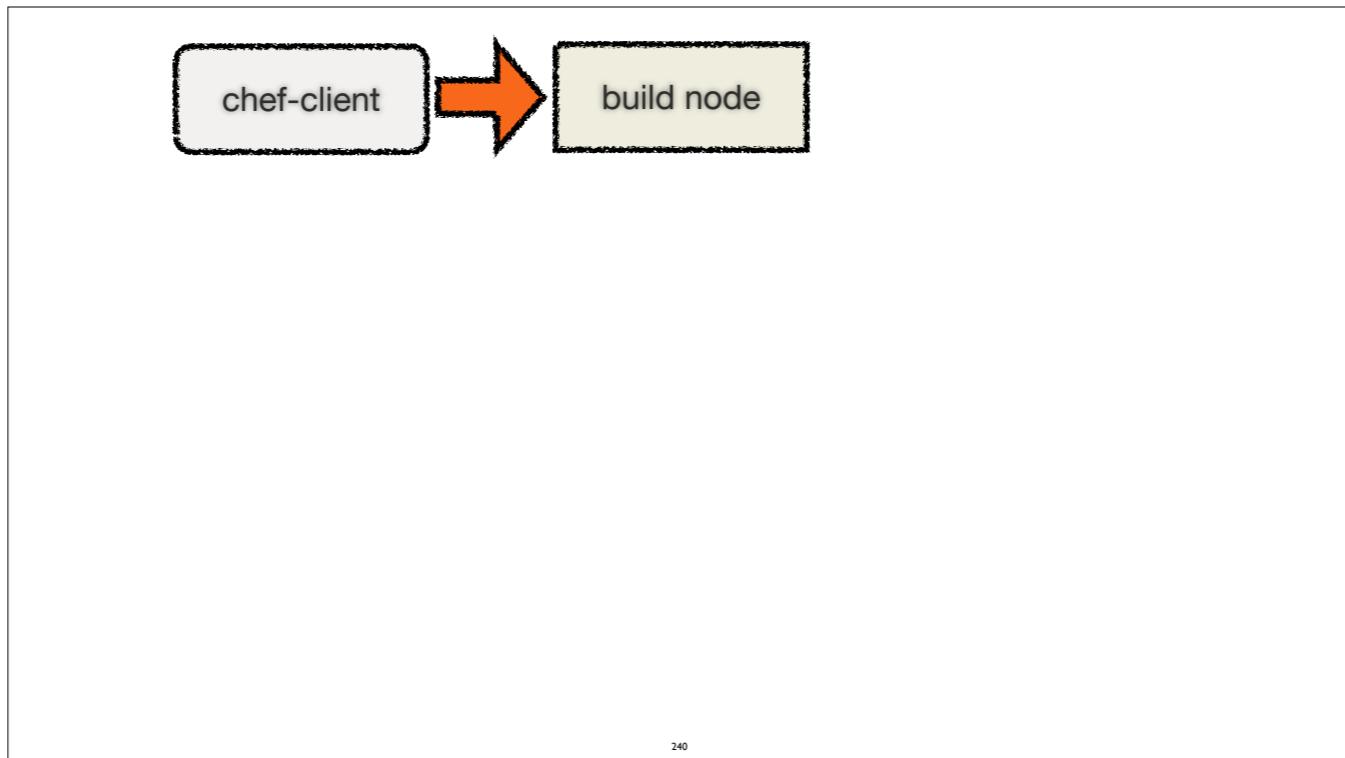
Lesson Objectives

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



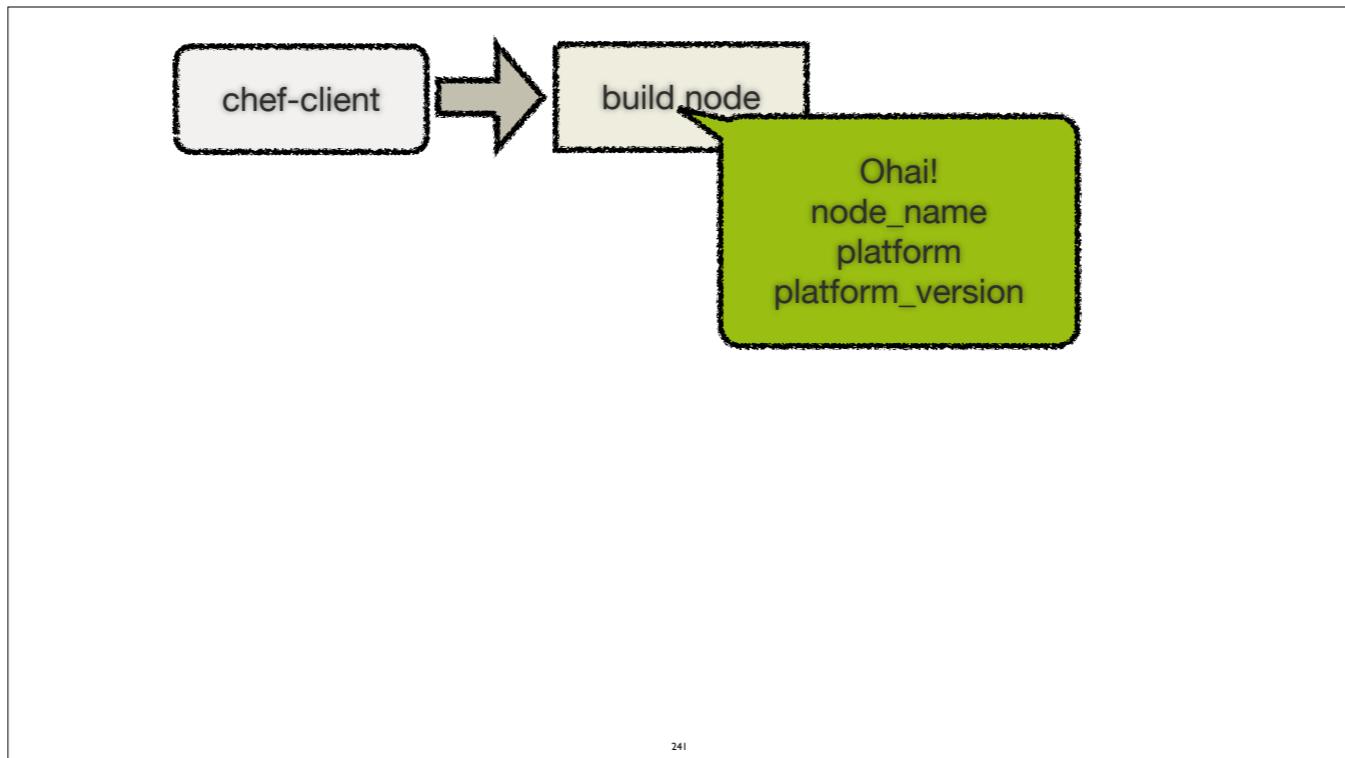
239

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.



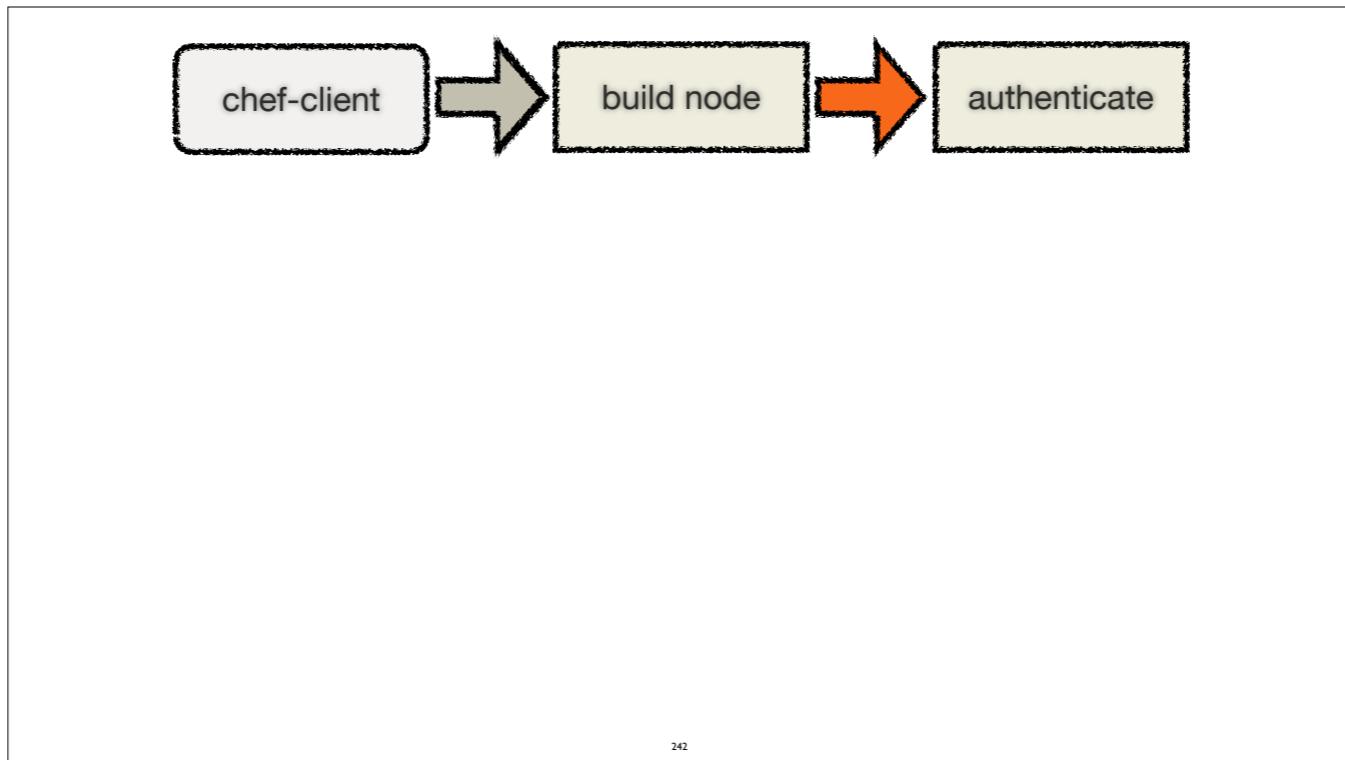
240

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.



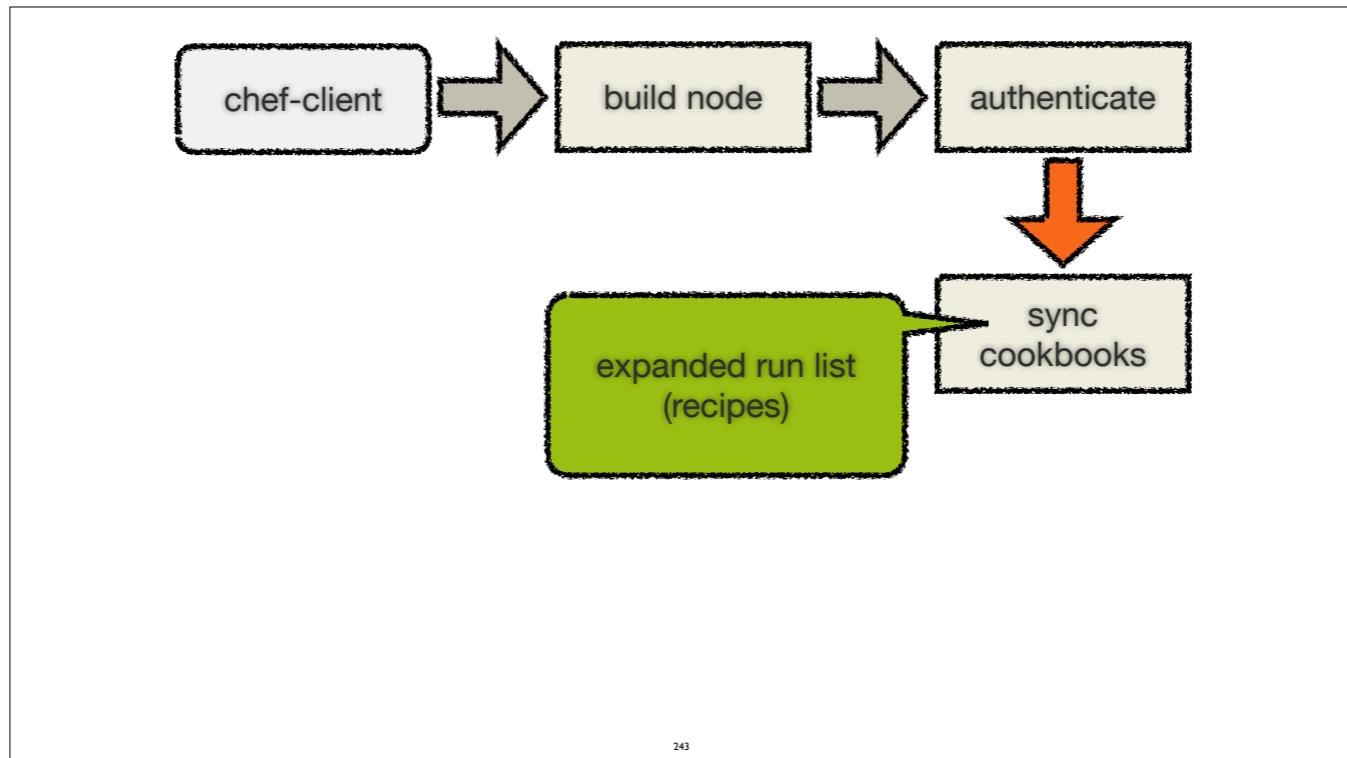
241

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.



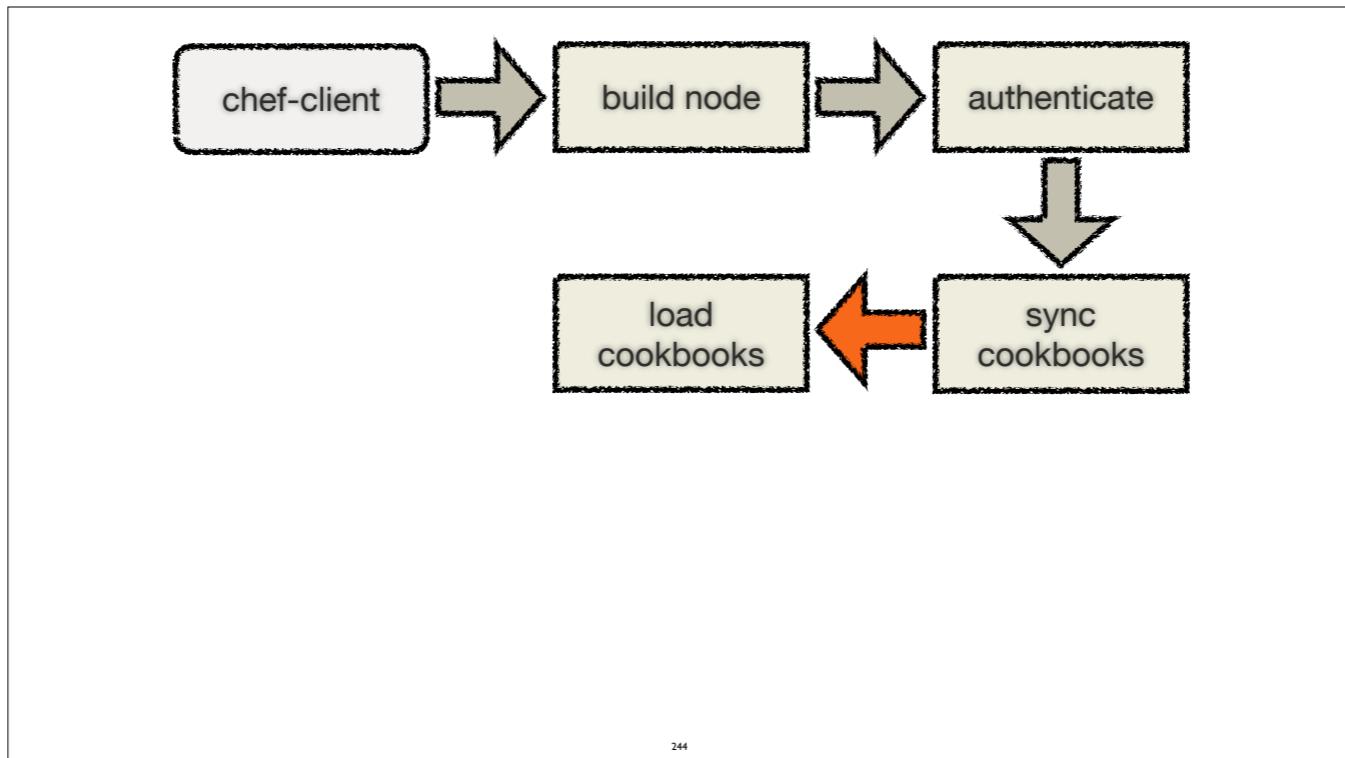
242

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.



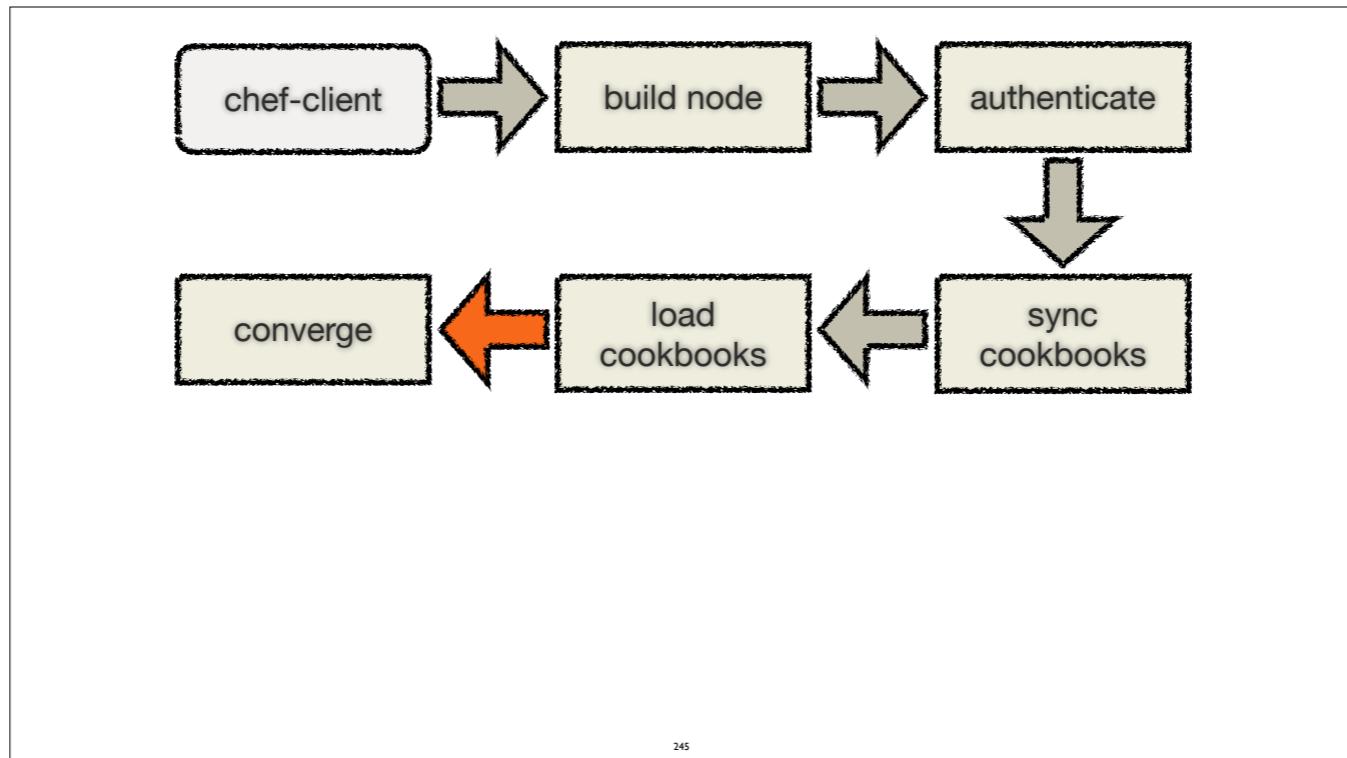
243

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.

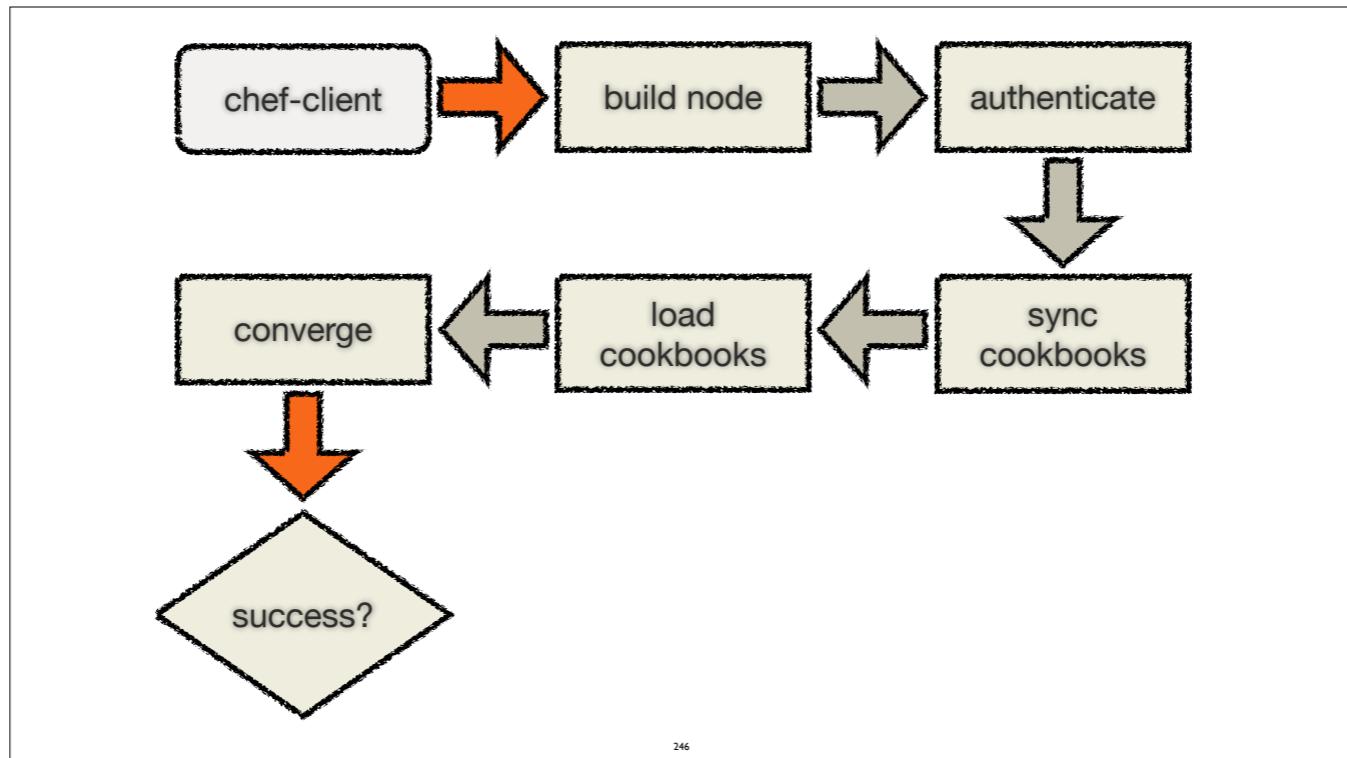


244

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.

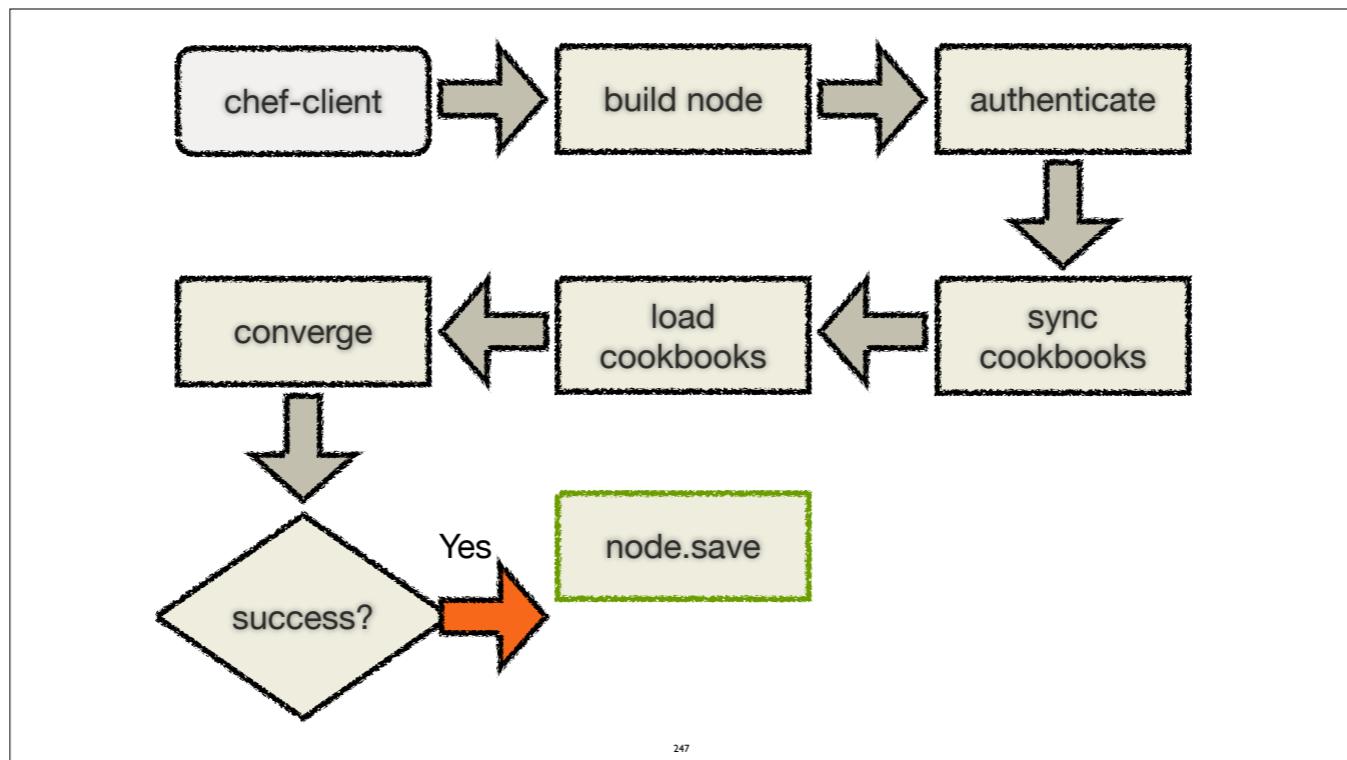


1. Chef client starts up.
2. Chef builds the node object, where it runs ohai to find out the node_name (fqdn) and the node's platform/platform_version
3. Chef authenticates with the server as the node_name, using the API client key. We'll cover the authentication cycle in detail shortly.
4. All the cookbooks 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.



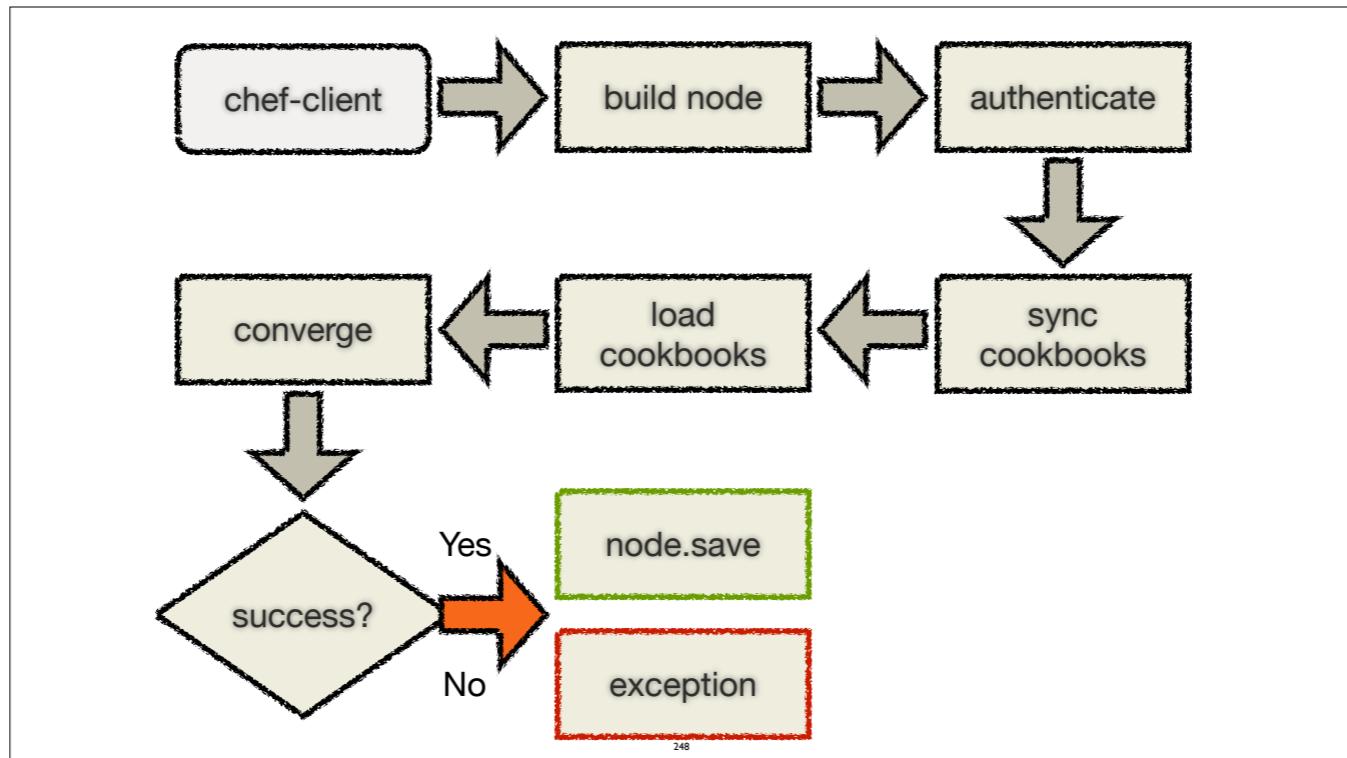
246

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.

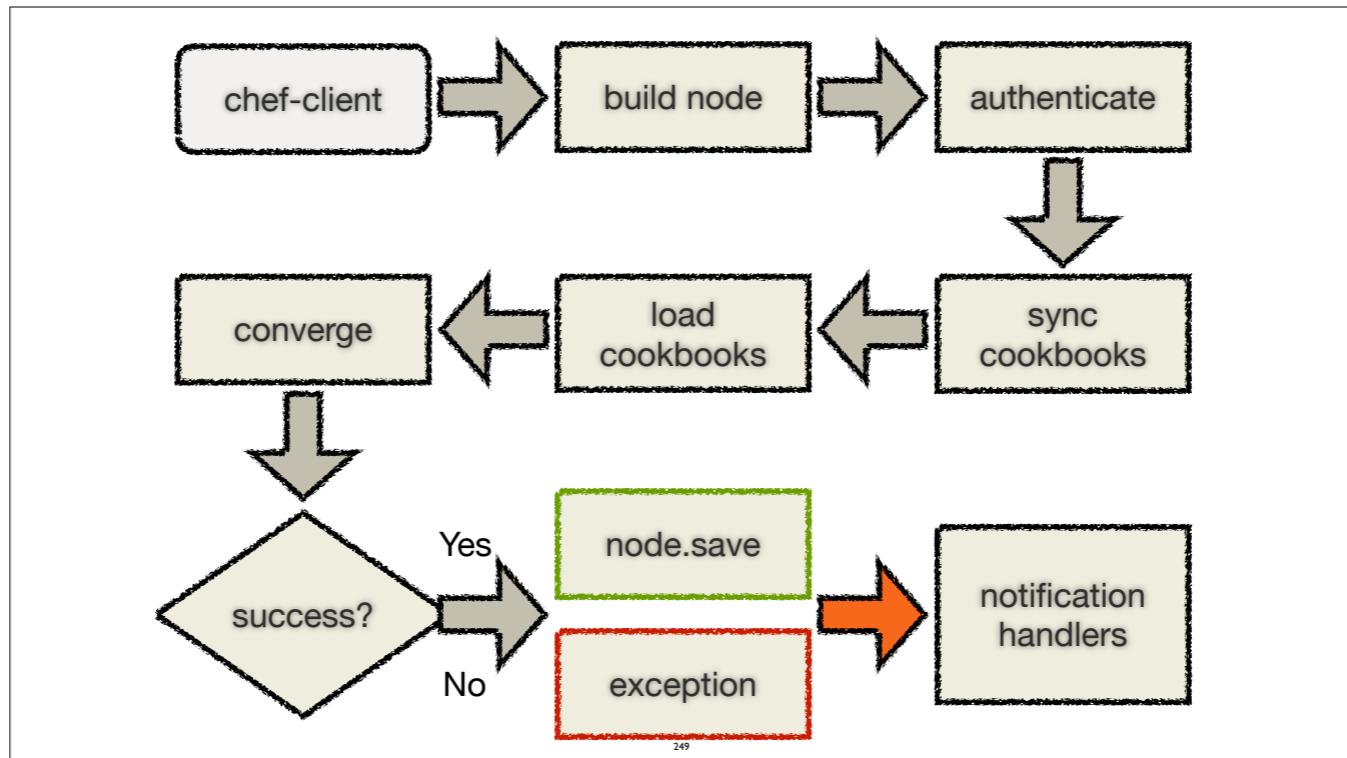


247

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.



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.



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

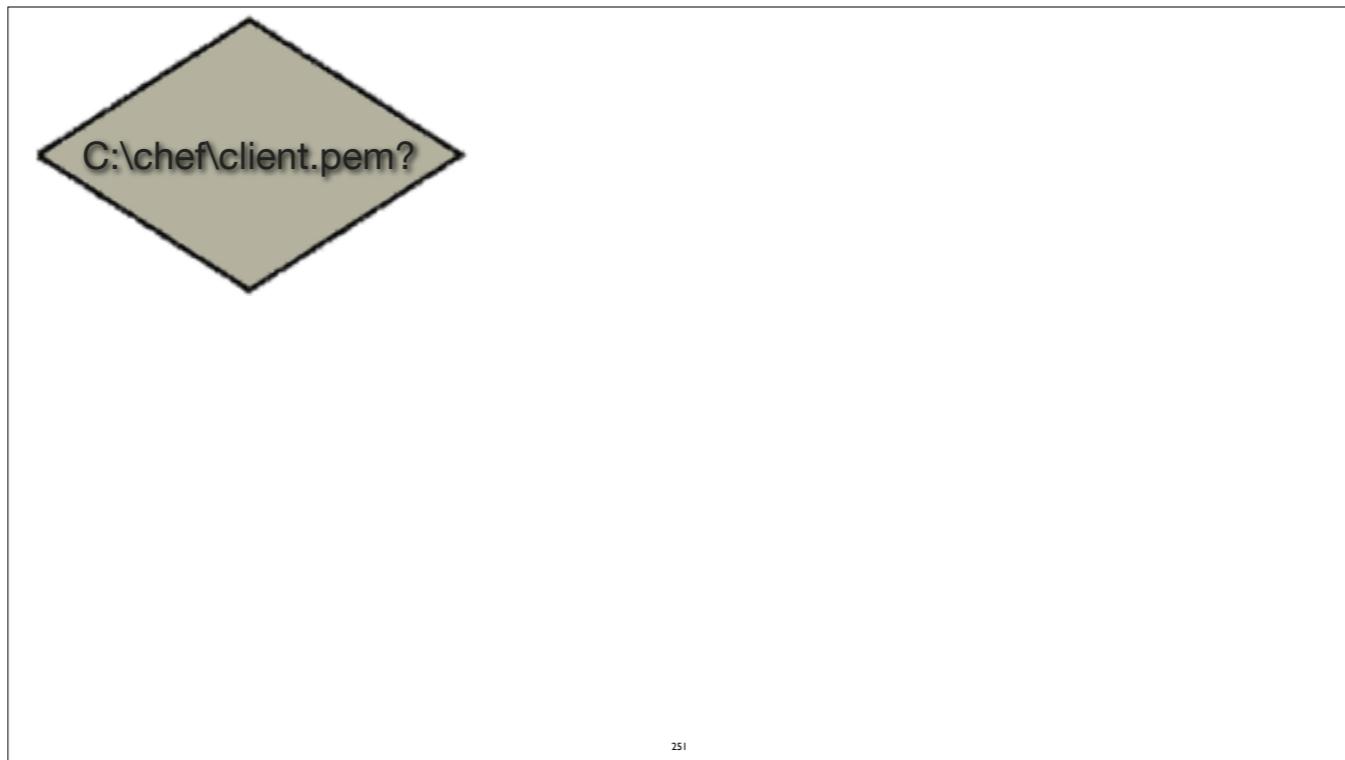
- Chef Server requires keys to authenticate.
 - `client.pem` - private key for API client
 - `validation.pem` - private key for ORGNAME-validator
- Next, let's see how those are used...

250

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.



251

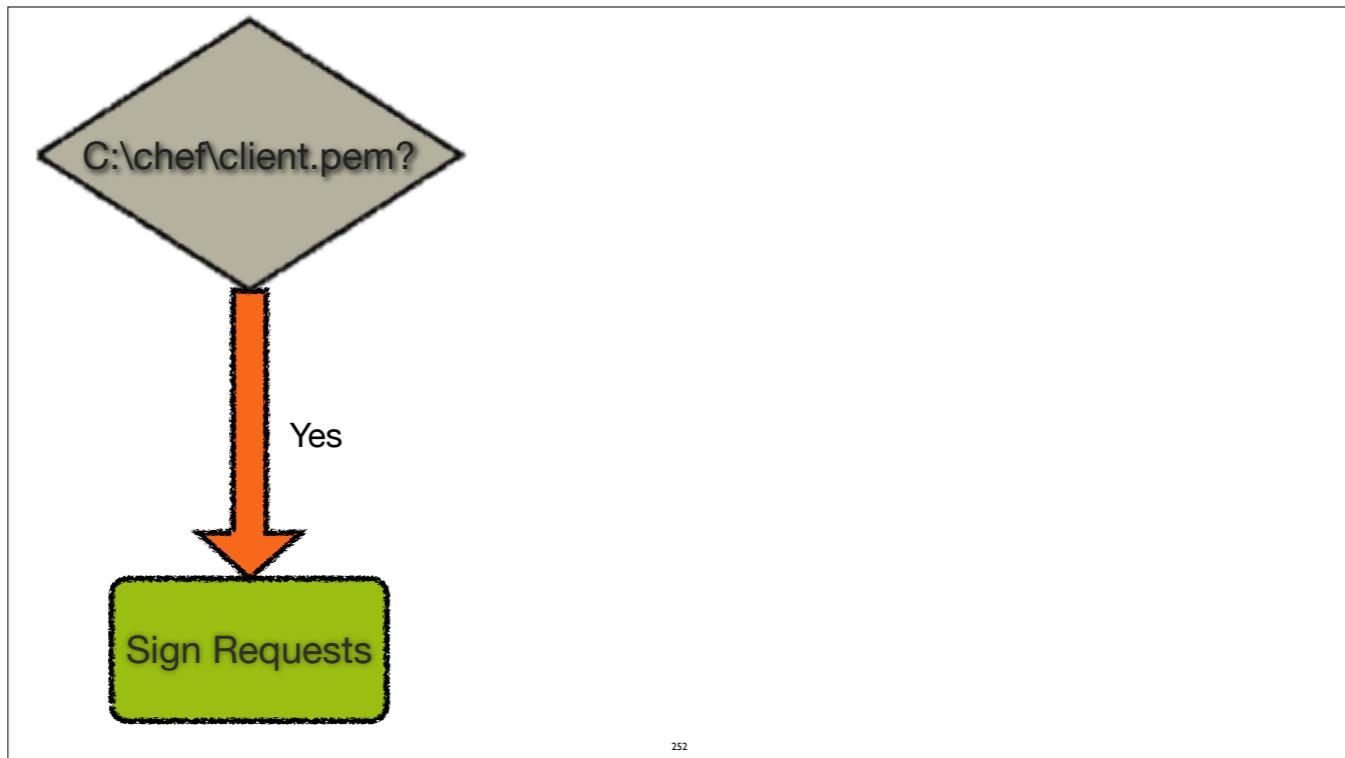
Take a minute to talk about how chef does authentication:

- * 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



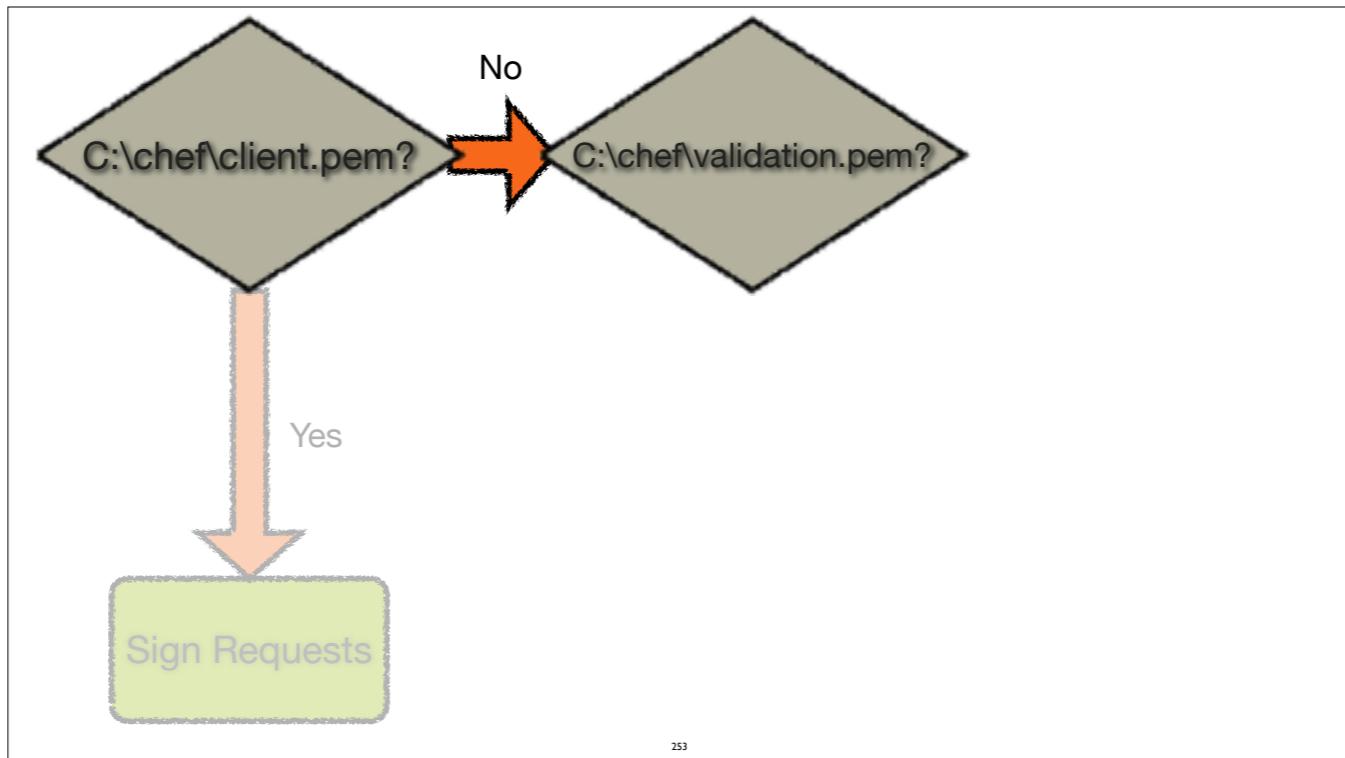
Take a minute to talk about how chef does authentication:

- * 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



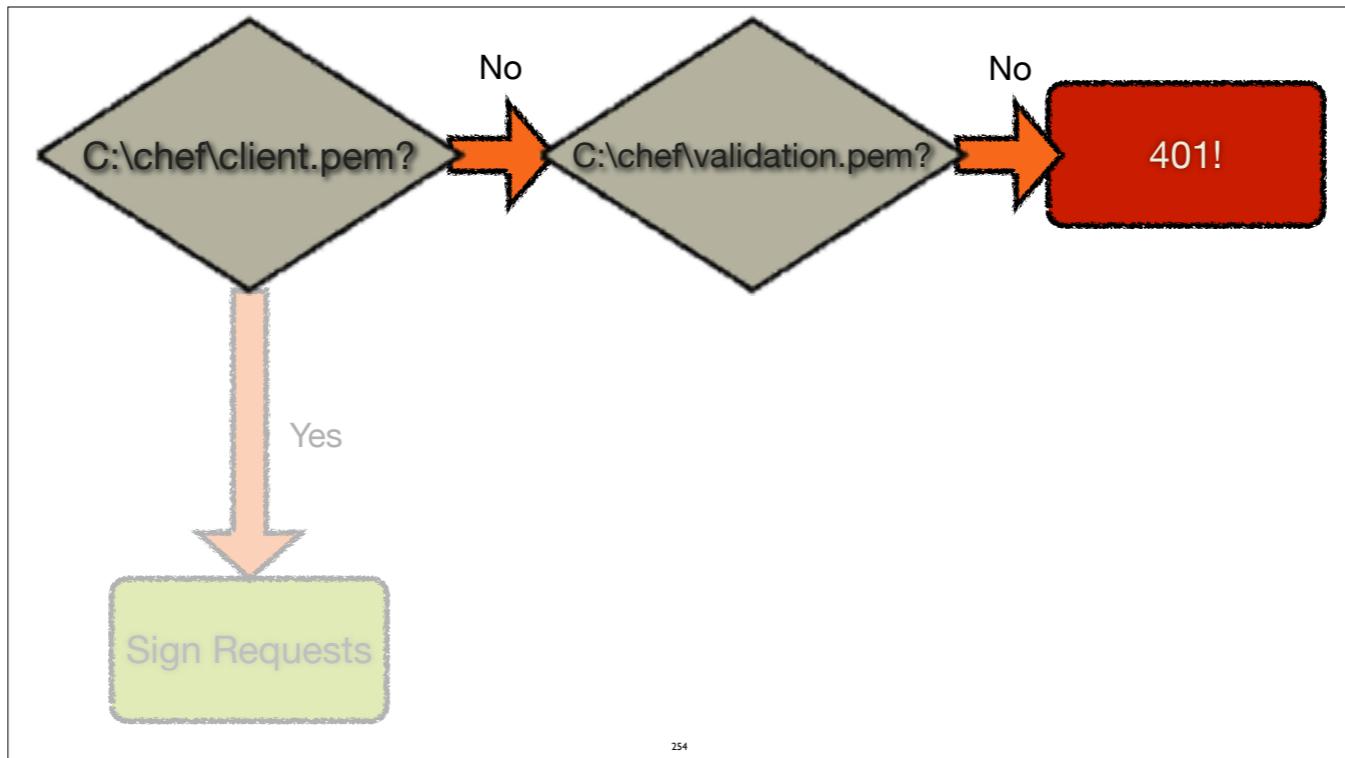
Take a minute to talk about how chef does authentication:

- * 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



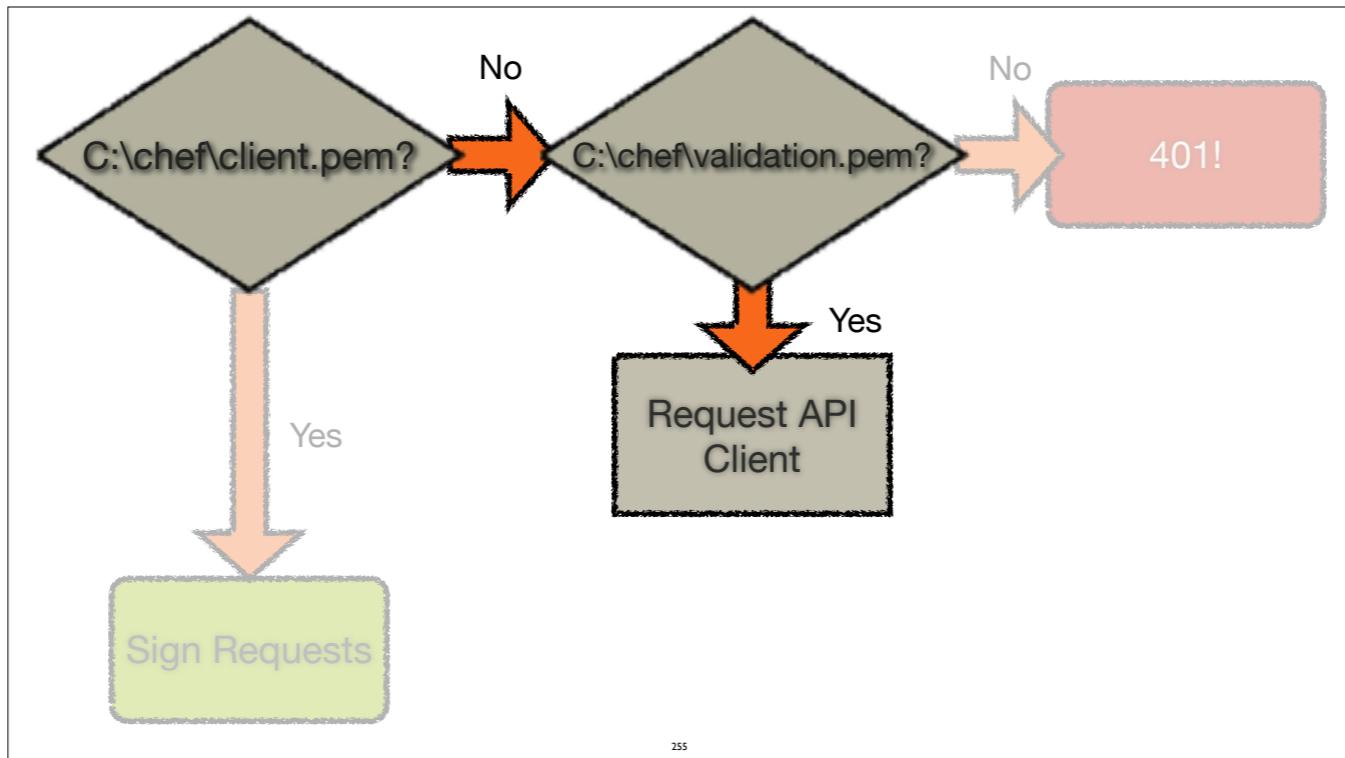
Take a minute to talk about how chef does authentication:

- * 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



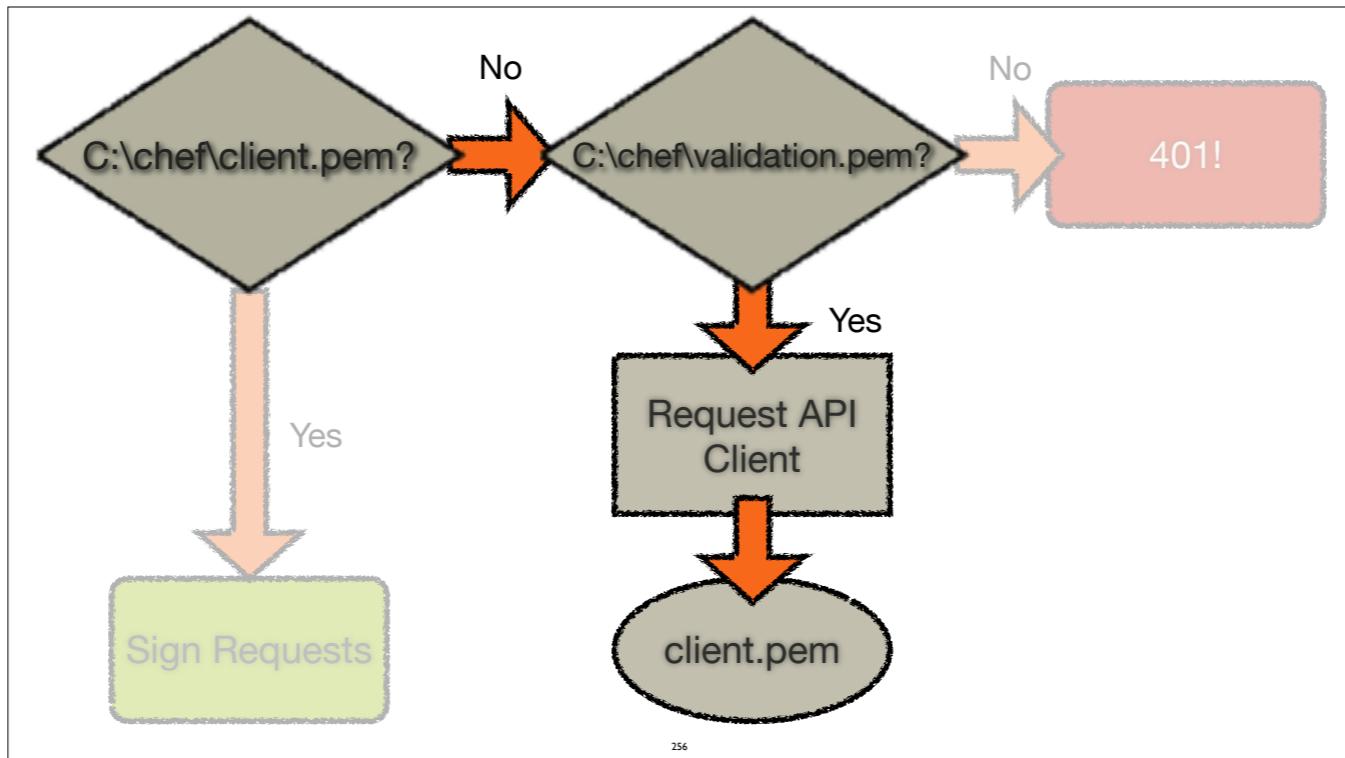
Take a minute to talk about how chef does authentication:

- * 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



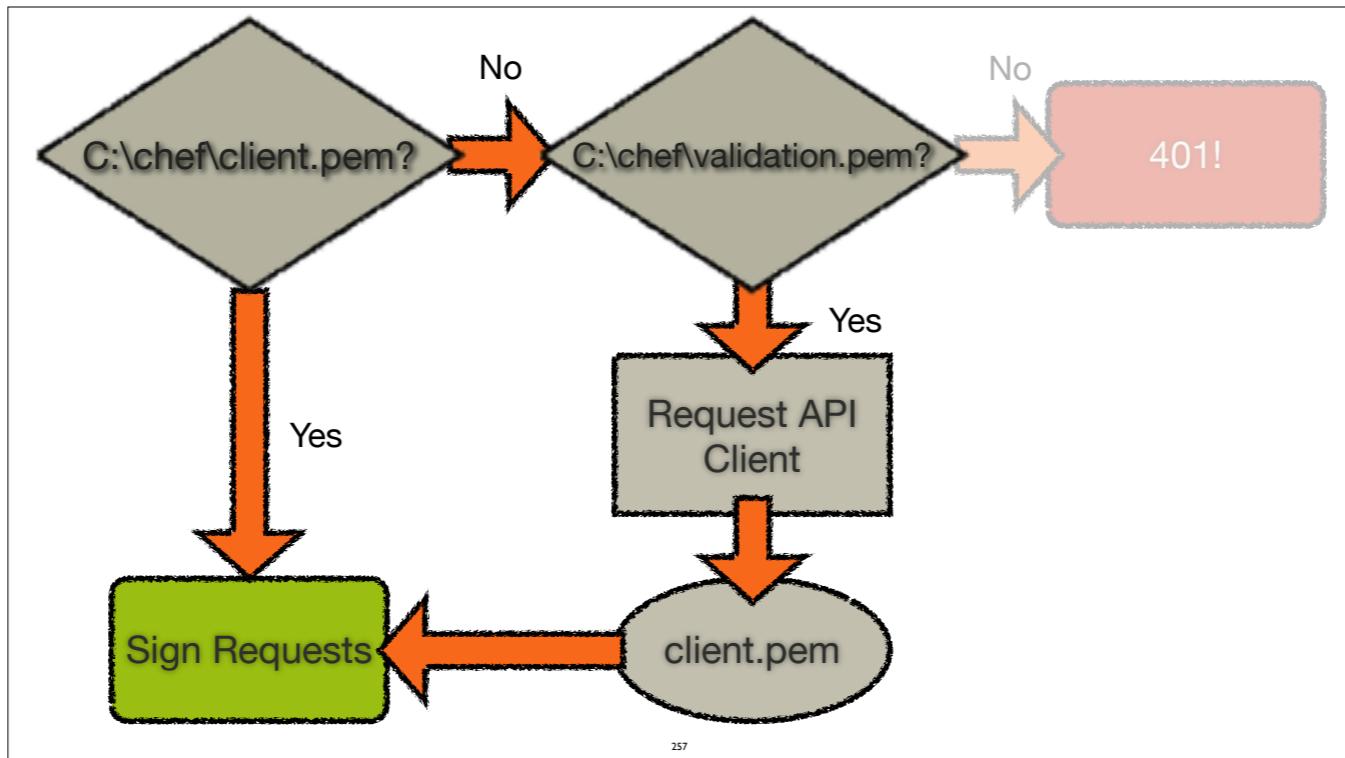
Take a minute to talk about how chef does authentication:

- * 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



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



About the resource collection

v2.1.1_WIN



Multiphase Execution - Compile Phase

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

Multiphase Execution - Execute Phase

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

Resource Collection Phase 1 - Compile Phase

Recipe

```
powershell_script "Install IIS" do
  action :run
  code "add-windowsfeature Web-Server"
end

service "w3svc" do
  action [ :enable, :start ]
end

cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

Resource Collection

```
resource_collection = [
  Powershell_script["Install IIS"],
  service ["w3svc"],
  cookbook_file["c:\inetpub\wwwroot\Default.htm"]
]
```

i.e. we end up with an array of resources

Instructor Note: This is a building slide – to illustrate the resource collection that was build in the previous section!

Important to note that this resource collection shown here is quite simple, and will become more complicated later in the course as we iterate in a recipe, and include notifications (especially immediate notifications)

Resource Collection Phase 1 - Execute Phase

Recipe

```
powershell_script "Install IIS" do
  action :run
  code "add-windowsfeature Web-Server"
end

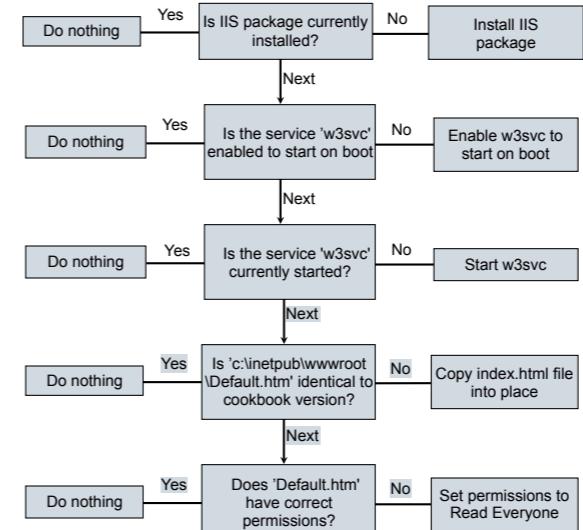
service "w3svc" do
  action [ :enable, :start ]
end

cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

Resource Collection

```
resource_collection = [
  package["Install IIS"],
  cookbook_file["c:\inetpub\wwwroot\Default.htm"],
  service ["w3svc"]
]
```

Execution



Instructor Note: This is a building slide – to illustrate the resource collection that was build in the previous section!

Instructor Note: This is a building slide!

Question: What would happen if Apache was installed on the node, but was not running?

Recipe order is important!

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

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

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

263

<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>

Resource Collection - Multiple Recipes

1. recipe[ntp::client]

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

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

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

Resource Collection

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

264

<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>
All attributes are passed to the resource collection along with the resource

NOTE: The next 3 slides are building slides and advance automatically

Resource Collection - Multiple Recipes

2. recipe[openssh::client]

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

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

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

Resource Collection

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

265

<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>
All attributes are passed to the resource collection along with the resource

Resource Collection - Multiple Recipes

3. recipe[iis_demo::default]

```
powershell_script "Install IIS" do
  action :run
  code "add-windowsfeature Web-Server"
end

service "w3svc" do
  action [ :enable, :start ]
end

cookbook_file 'c:\wwwroot\Default.htm' do
  source "Default.htm"
  rights :read, "Everyone"
end
```

Resource Collection

```
resource_collection [
  package[ntp],
  template[/etc/ntp.conf],
  service[ntp],
  package[openssh],
  template[/etc/sshd/sshd_config],
  service[openssh],
  powershell_script[Install IIS],
  service[w3svc],
  cookbook_file[c:\wwwroot\Default.htm]
]
```

266

<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>
All attributes are passed to the resource collection along with the resource

Note: cookbook_file truncated to fit the slide

The final resource collection

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

```
package[ntp]
template[/etc/ntp.conf]
service[ntp]
package[openssh]
template[/etc/sshd/sshd_config]
service[openssh]
powershell_script[Install IIS]
service[w3svc]
cookbook_file[c:\wwwroot\Default.htm]
```

267

<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>

Multiphase Execution

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



<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>

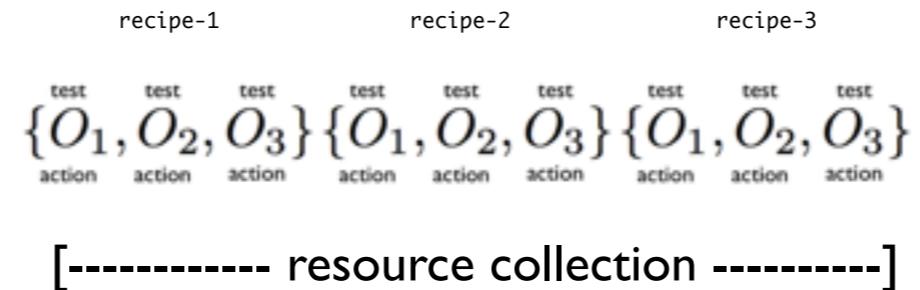
<<Review Comment: This is taken from the bottles example, but a bit more real

Point out:

- that Chef has a compile phase and an execute phase
- during the compile phase, it compiles resources into a “resource collection” and then executes them. You can use arbitrary Ruby (as seen here) to create the resources in the resource collection

Remember - Resource order is important!

- Resources are invoked in the order they appear in the recipe



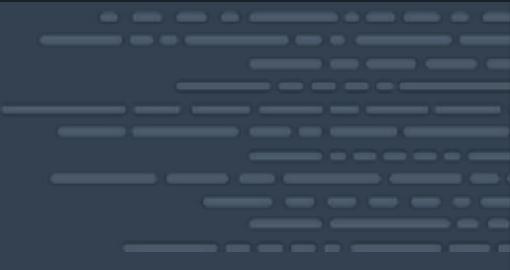
269

<<Review Comment: Added for Trello card <http://bit.ly/1c1JrZ4>
<<WIP: Do we keep this slide? Some push back on this slide in review comments

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?
- In which phase of a chef-client run do plain Ruby statements get evaluated?

-) build node, authenticate, sync cookbooks, load cookbooks, converge, save/exception, notifiers
-) If connects via org key, is issued private key for future use.
-) Chef API requests are time sensitive. Check your NTP drift.
-) Resource collection



Introducing the Node object

Attributes & Search

v2.1.1_WIN



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
- The 'node object' is the representation of that physical node within Chef (e.g. in JSON)
- When you are writing Recipes, the Node object is always available to you

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

Node

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

View Node on Chef Server

- Click the 'Details' tab

The screenshot shows the Chef Server interface for viewing a node. At the top, there's a navigation bar with tabs for Nodes, Reports, Policy, and Administration. The Nodes tab is selected. Below the navigation is a table titled "Showing All Nodes" with columns for Node Name, Platform, FQDN, IP Address, Uptime, and Last Check In. A single row is selected for "node1", which is a windows node with FQDN C1263834251, IP Address 10.160.33.236, and Uptime 44 minutes. Below the table, the node details are displayed under the heading "Node: node1". There are three tabs: Details (which is highlighted with an orange circle), Attributes, and Permissions. Under the Details tab, there are sections for "Last Check In: 24 Minutes ago" and "Uptime: 44 Minutes since 2014-02-21 14:51". To the right, there's a summary box with fields for Environment (set to "0"), Platform (windows), FQDN (C1263834251), and IP Address (10.160.33.236). At the bottom left, it says "There are no items to display" under the Tags section. On the right, there's a "Run List" section with "Expand All" and "Collapse" buttons. The footer includes copyright information (Copyright © 2012 – 2014 Chef, Inc.) and a help link ("Need help? If you have questions or are stuck, we are here to help").

View Node on Chef Server

- Click the 'Attributes' tab

The screenshot shows the Chef Server web interface. At the top, there's a navigation bar with tabs for 'Nodes', 'Reports', 'Policy', and 'Administration'. Below this, a table lists nodes, with one row for 'node1' highlighted in orange. On the left, a sidebar has options like 'Delete', 'Manage Tags', 'Reset Key', 'Edit Run List', and 'Edit Attributes'. The main content area is titled 'Node: node1' and shows three tabs: 'Details' (selected), 'Attributes' (circled in orange), and 'Permissions'. Under 'Attributes', there's a section for 'Attributes' with 'Expand All' and 'Collapse All' buttons. A list of attributes is shown, each preceded by a plus sign and a bullet point:

- tags:
 - + languages
 - + kernel
 - os: windows
 - os_version: 6.2.6000
- + chef_packages
 - hostname: C1263834251
 - ipaddr: C1263834251
 - domain:
- + network
 - + counters

Exercise: Listing nodes

```
PS\> knife node list
```

```
node1
```

Exercise: Listing clients

```
PS\> knife client list
```

```
ORGNAME-validator  
node1
```

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

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: Showing details about a node

```
PS\> knife node show node1
```

```
Node Name:    node1
Environment:  _default
FQDN:        C1263834251
IP:          10.160.33.236
Run List:    recipe[iis_demo]
Roles:
Recipes:     iis_demo, iis_demo::default
Platform:   windows 6.2.9200
Tags:
```

The FQDN is made up of the 'Computer name' (i.e. right-clicking the My Computer and choosing Properties) and the domain if added to a domain. In our case we don't have a domain configured.

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

Ohai

```
"languages": {  
  "ruby": {  
    "platform": "i386-mingw32",  
    "version": "1.9.3",  
    "target": "i386-pc-mingw32",  
    "target_cpu": "i386",  
    "target_vendor": "pc",  
    "target_os": "mingw32",  
    "host": "i686-pc-mingw32",  
    "host_cpu": "i686",  
    "host_os": "mingw32",  
    "host_vendor": "pc",  
  },  
  "perl": {  
    "version": "5.8.8",  
  },  
  "archname": "msys-64int"  
},  
...  
}
```

```
"kernel": {  
  "os_info": {  
    "boot_device": "\Device\HarddiskVolume1",  
    "build_number": "9200",  
    "build_type": "Multiprocessor Free",  
    "caption": "Microsoft Windows Server 2012 Standard",  
    "code_set": "1252",  
    "country_code": "1",  
    "creation_class_name": "Win32_OperatingSystem",  
    "cs_creation_class_name": "Win32_ComputerSystem",  
    "csd_version": null,  
    "cs_name": "C1263834251"},  
  "machine": "x86_64",  
  "pnp_drivers": {  
    "SWD\PRINTENUM\{B86F2C50-C174-459A-AE9E-0097FCE113EE":  
    ...  
  }  
}
```

```
"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"  
      }  
    }  
  }  
}
```

Exercise: Run Ohai on Target

```
remote@PS\> ohai | oh -Paging
```

```
{  
  "languages": {  
    "ruby": {  
      "platform": "i386-mingw32",  
      "version": "1.9.3",  
      "release_date": "2013-11-22",  
      "target": "i386-pc-mingw32",  
      "target_cpu": "i386",  
      "target_vendor": "pc",  
      "target_os": "mingw32",  
      "host": "i686-pc-mingw32",  
      "host_cpu": "i686",  
      "host_os": "mingw32",  
      "host_vendor": "pc",  
      ...  
    }  
  }  
}
```

Hammer home that ohai collects a ton of data

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.

Note – if someone asks what is OH – tell them it means Out-Host – then suggest they type ‘alias oh’

Exercise: Showing all the attributes of a node

```
PS\> knife node show node1 -l
```

```
Node Name:    node1
Environment:  _default
FQDN:        C1263834251
IP:          10.160.33.236
Run List:    recipe[iis_demo]
Roles:
Recipes:     iis_demo, iis_demo::default
Platform:   windows 6.2.9200
Tags:
Attributes:
...
```

knife node show target1 -l | oh -Paging WILL hang knife... I should file a bug

Exercise: Showing the raw format of the node

```
PS\> knife node show node1 -Fj
```

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

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

Exercise: Show only the fqdn attribute

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

```
node1:  
fqdn: C1263834251
```

Exercise: Use search to find the same data

```
PS\> knife search node "*:*" -a fqdn
```

```
1 items found
```

```
node1:
```

```
fqdn: C1263834251
```

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?

-) It is the 'representation' of the physical object within chef.

-) Metadata about the object. Could be FQDN, CPU type, memory, apps installed, etc.

-) knife command (knife node show <node> -l, GUI, search

-) Yes; knife search node "*" -a cpu

Setting Node Attributes

Setting attributes in recipes and attribute files

v2.1.1_WIN



Lesson Objectives

- After completing the lesson, you will be able to
 - Describe where and how attributes are set
 - Explain the attribute merge order and precedence rules
 - Declare an attribute with a recipe and sets its value

http://docs.opscode.com/chef_overview_attributes.html

What are Attributes?

- Attributes represent information about your node
- The information can be auto-detected from the node (e.g. # of CPUs, amount of RAM) & populated by Ohai
- You can also set attributes on your node using cookbook recipes & attribute files, roles, environments, etc
- Attributes keep the program code separate from data
- All attributes are set on the "node object", and are indexed for search on the server

Attributes are data points that can be static (from ohai) or variable (set in attributes file, environments, roles, and recipes)

<<WIP: Question: Is the node object stored in chef in JSON format, or stored in a backend DB in some format and just displayed in JSON?

Attribute Sources

- Attributes can be set at various levels (in increasing order of precedence)
 - Automatically on the node itself (by Ohai)
 - In roles
 - In environments
 - In cookbook recipes
 - In cookbook attribute files

[This is b](#)it of a fudge as there are 15 level of precedence, but we dont want to go into that right now!

Ohai

```
"languages": {
  "ruby": {
    "platform": "i386-mingw32",
    "version": "1.9.3",
    "target": "i386-pc-mingw32",
    "target_cpu": "i386",
    "target_vendor": "pc",
    "target_os": "mingw32",
    "host": "i686-pc-mingw32",
    "host_cpu": "i686",
    "host_os": "mingw32",
    "host_vendor": "pc",
  },
  "perl": {
    "version": "5.8.8",
  },
  "archname": "msys-64int"
},
...}

"kernel": {
  "os_info": {
    "boot_device": "\Device\HarddiskVolume1",
    "build_number": "9200",
    "build_type": "Multiprocessor Free",
    "caption": "Microsoft Windows Server 2012 Standard",
    "code_set": "1252",
    "country_code": "1",
    "creation_class_name": "Win32_OperatingSystem",
    "cs_creation_class_name": "Win32_ComputerSystem",
    "csd_version": null,
    "cs_name": "C1263834251",
    "machine": "x86_64",
    "pnp_drivers": {
      "SWD\PRINTENUM\{B86F2C50-C174-459A-AE9E-0097FCE113EE"}:
    }
  }
}

"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",
    }
  }
}
```

Setting attributes in attribute files

- Attributes can be set in the cookbook's attributes file
 - `./cookbooks/<cookbook>/attributes/default.rb`
- Format is

```
precedence           attribute name           attribute value  
↓                  ↓                         ↓  
default["iis_demo"]["indexfile"] = "Default.htm"
```

- We'll look at precedence later....

'default' filename can be replaced with 'platform' (e.g. 'centos'), or 'platform_version' (e.g. '6.3')
Note the attribute can be an array or a hash

<<WIP: I think these can also be in JSON format, yeah? If so, so we want to call that out?

Setting Attributes in Recipes

- They can also be set directly in Recipes
- Precede attribute name with 'node.' as follows

The diagram illustrates the structure of a node attribute assignment. It shows the code: `node.default["iis_demo"]["indexfile"] = "Default.htm"`. Three arrows point downwards from labels to specific parts of the code:

- An arrow labeled "precedence" points to the prefix `node.`.
- An arrow labeled "attribute name" points to the key `"indexfile"`.
- An arrow labeled "attribute value" points to the string `"Default.htm"`.

A fourth arrow points upwards from a box labeled "declares it a node attribute" to the prefix `node.`.

<<WIP: I think these can also be in JSON format, yeah? If so, so we want to call that out?

The Problem and the Success Criteria

- **The Problem:** We have defined our 'index.html' homepage in the recipe, but we may want to change that without altering the recipe
- **Success Criteria:** We can change the homepage by changing an attribute value

Exercise: Set attribute in attributes file

OPEN IN EDITOR: Cookbooks\iis_demo\attributes\default.rb

```
default["iis_demo"]["indexfile"] = "Default1.htm"
```

SAVE FILE!

- Set an attribute to a specific value in the attributes file

Exercise: Add index1.html to cookbook's files/default directory

OPEN IN EDITOR: cookbooks\iis_demo\files\default\Default1.htm

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>This is Default1.htm</h2>
    <p>We configured this in the attributes file</p>
  </body>
</html>
```

SAVE FILE!

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. :)

Exercise: Set attribute in a recipe

OPEN IN EDITOR: Cookbooks\iis_demo\recipes\default.rb

```
service "w3svc" do
  action [:enable, :start]
end

cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source node["iis_demo"]["indexfile"]
  rights :read, "Everyone"
end
```

SAVE FILE!

node[:apache][:indexfile] interpolates to the value of the attribute, which is set in two places – 'index1.html' in attributes file and 'index2.html' in recipe

Exercise: Upload the cookbook

```
PS\> knife cookbook upload iis_demo
```

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

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: Re-run Chef Client

```
remote@PS> chef-client
```

```
...
* cookbook_file[c:\inetpub\wwwroot\Default.htm] action create[2014-02-25T01:39:45-08:00] INFO: Processing cookbook_file[c:\inetpub\wwwroot\Default.htm] action create (iis_demo::default line 18)
[2014-02-25T01:39:45-08:00] INFO: cookbook_file[c:\inetpub\wwwroot\Default.htm] backed up to c:/chef/backup/inetpub/wwwroot\Default.htm.chef-20140225013945.548152
[2014-02-25T01:39:45-08:00] INFO: cookbook_file[c:\inetpub\wwwroot\Default.htm] updated file contents c:\inetpub\wwwroot\Default.htm

- update content in file c:\inetpub\wwwroot\Default.htm from 231d53 to 442a41
  --- c:\inetpub\wwwroot\Default.htm      2014-02-24 06:25:33.000000000 -0800
  +++ C:/Users/chef/AppData/Local/Temp/4/Default.htm20140225-1272-levzs75 2014-02-25 01:39:45.000000000 -0800
@@ -1,7 +1,8 @@
<html>
- <body>
-   <h1>Hello, world!</h1>
- </body>
-</html>
-
+ <body>
+   <h1>Hello, world!</h1>
+   <h2>This is Default1.htm</h2>
+   <p>We configured this in the attributes file</p>
+ </body>
+</html>
[2014-02-25T01:39:46-08:00] INFO: Chef Run complete in 3.556801 seconds
```

Which index.html was used – index1.html or index2.html?

Exercise: Verify new homepage works

- Open a web browser
- The homepage takes the attribute file value



You should see the new index.html from the attributes file.

Exercise: Set attribute in the recipe

OPEN IN EDITOR: Cookbooks\iis_demo\recipes\default.rb

```
service "w3svc" do
  action [:enable, :start]
end

node.default["iis_demo"]["indexfile"] = "Default2.htm"
cookbook_file 'c:\inetpub\wwwroot\Default.htm' do
  source node["iis_demo"]["indexfile"]
  rights :read, "Everyone"
end
```

SAVE FILE!

node["apache"]["indexfile"] is the value of the attribute, which is set in two places – 'index1.html' in attributes file and 'index2.html' in recipe

Exercise: Add index2.html to cookbook's files/default directory

OPEN IN EDITOR: cookbooks\iis_demo\files\default\Default2.htm

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>This is Default2.htm</h2>
    <p>We configured this in the recipe</p>
  </body>
</html>
```

SAVE FILE!

Exercise: Upload the cookbook

```
PS\> knife cookbook upload iis_demo
```

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

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: Re-run Chef Client

```
remote@PSI> chef-client
```

```
* cookbook_file[c:\inetpub\wwwroot\Default.htm] action create[2014-02-25T01:54:29-08:00] INFO: Processing cookbook_file[c:\inetpub\wwwroot\Default.htm] action create (iis_demo::default line 25)
[2014-02-25T01:54:30-08:00] INFO: cookbook_file[c:\inetpub\wwwroot\Default.htm] backed up to c:/chef/backup/inetpub/wwwroot\Default.htm.chef-20140225015430.025195
[2014-02-25T01:54:30-08:00] INFO: cookbook_file[c:\inetpub\wwwroot\Default.htm] updated file contents c:\inetpub\wwwroot\Default.htm

- update content in file c:\inetpub\wwwroot\Default.htm from 2d8349 to 355514
  --- c:\inetpub\wwwroot\Default.htm 2014-02-25 01:43:24.000000000 -0800
  +++ C:/Users/chef/AppData/Local/Temp/4/Default.htm20140225-4300-17e2q1j 2014-02-25 01:54:30.000000000 -0800
  @@ -1,8 +1,9 @@
<html>
  <body>
    <h1>Hello, world!</h1>
-   <h2>This is Default1.htm</h2>
-   <p>We configured this in the attributes file</p>
+   <h2>This is Default2.htm</h2>
+   <p>We configured this in the attributes file</p>
  </body>
</html>
+
[2014-02-25T01:54:30-08:00] INFO: Chef Run complete in 3.525611 seconds
[2014-02-25T01:54:30-08:00] INFO: Removing cookbooks/iis_demo/files/default/Default1.htm from the cache; it is no longer needed by chef-client.

Chef Client finished, 2/4 resources updated in 16.099269 seconds
```

Which index.html was used – index1.html or index2.html?

Exercise: Verify new homepage works

- Open a web browser
- Recipe value has taken precedence



The attribute and the file 'index1.html' is still in place – but specifying it in the recipe takes precedence!

Exercise: Viewing Attributes via WebUI

The screenshot shows the Chef WebUI interface. At the top, there is a navigation bar with tabs for Nodes, Reports, Policy, and Administration. Below the navigation bar, a sidebar on the left contains links for Nodes, Delete, Manage Tags, Reset Key, Edit Run List, and Edit Attributes. The main content area displays a table titled "Showing All Nodes" with one row for "node1". The table includes columns for Node Name, Platform, FQDN, and IP Address. Below this, a modal window is open for the node named "node1". The modal has tabs for Details, Attributes, and Permissions, with the Attributes tab selected. Under the Attributes tab, there is a "Attributes" section with "Expand All" and "Collapse All" buttons. A tree view of attributes is shown, with the "is_demo" attribute expanded, revealing its value ("is_demo: Default2.json"). Other collapsed categories include tags, languages, kernel, os, os_version, chef_packages, hostname, fqdn, domain, network, and counters.

Exercise: Viewing attributes using knife

```
PS\> knife node show node1 -l | more
```

```
...
Attributes:
tags:

Default Attributes:
iis_demo:
  indexfile: Default2.htm

Override Attributes:

Automatic Attributes (Ohai Data):
chef_packages:
  chef:
    chef_root: C:/opscode/chef/embedded/lib/ruby/gems/1.9.1/gems/chef-11.10.4-x86-mingw32/lib
    version: 11.10.4
  ohai:
    ohai_root: C:/opscode/chef/embedded/lib/ruby/gems/1.9.1/gems/ohai-6.20.0/lib/ohai
    version: 6.20.0
command:
counters:
  network:
...
...
```

Which index.html was used – index1.html or index2.html?

Setting Attributes in Roles and Environments

- Attributes can also be set in Roles and Environments
 - These use raw JSON or Ruby format

```
default_attributes({ "iis_demo" => { "port" => 80} })
```

- More later in Roles and Environments sections...

Ruby format (with '`=>`') shown.

Default Attribute Precedence



- Please note this is a simplified diagram, and the precedences shown can be overridden

Questions

- What is an attribute?
- Where can attributes be set?
- What knife command can you use to view attributes?
- How many levels of attribute precedence are there?
- Which takes precedence, a default attribute set in the attribute file or in a recipe?

-) Metadata about an object (fqdn, type, version, etc.)
-) attribute file, recipe, role, automatically, etc.
-) knife node show
-) 15 (haven't seen all of them)
-) Recipe

Attributes, Templates, and Cookbook Dependencies

Writing a Hosts Cookbook

v2.1.1_WIN



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 slide formerly known as “MOTD” :)

The example in this lab simply writes to a HOSTS file

The Problem and the Success Criteria

- **The Problem:** We need to block access to the New York Times website from our computers in China
- **How:** We will use the hosts file to set the New York Times IP address to 0.0.0.0.
- **Success Criteria:** We see the hosts file updated on the server.

We have a small problem...

- But we don't have an attribute that reflects which region the server is in
 - We could add a node attribute for Location, but that will become very tiresome at scale

Solution

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

Exercise: Create a cookbook named ‘datacenter’

```
PS\> knife cookbook create datacenter
```

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

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 in the datacenter cookbook

OPEN IN EDITOR: cookbooks\datacenter\attributes\default.rb

```
default[ "datacenter" ][ "location" ] = "china"
```

SAVE FILE!

- Creates a new Node attribute:
node["datacenter"]["location"]
- Sets the value to the Ruby string – 'china'
- Note: there is no space between 'default' and '['
- The 'datacenter' in the attribute is just convention so you know the cookbook the attribute was set in. This is not an enforced syntax

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

- * A top level attribute named 'datacenter'
- * Whose value is a hash
- * With a key named 'location'
- * Whose value is the ruby literal 'china'

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.

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. Take my word for it. :)

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: Always make your cookbooks have default values

- If a cookbook needs an attribute to exist, it should either define a default value for it in an attribute file, or depend on another cookbook that does
- **Never rely on an attribute being created manually**

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 Datacenter cookbook

```
PS\> knife cookbook upload datacenter
```

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

Reiterate what happens here:

- * Syntax check
- * Upload to Chef Server

Exercise: Create a cookbook named 'hosts'

```
PS\> knife cookbook create hosts
```

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

Exercise: Open the default recipe in your editor

OPEN IN EDITOR: cookbooks\hosts\recipes\default.rb

```
#  
# Cookbook Name:: hosts  
# 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 a template resource for hosts.erb

- Use a **template** resource
- The **name** is
`"#{ENV['windir']}\\System32\\drivers\\etc\\hosts"`
- The resource has one parameter
 - **source** "hosts.erb"

Talk about the hard way again!

Point out that within Chef you have full access to Windows Environment variables

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.

The Template Resource

OPEN IN EDITOR: cookbooks\hosts\recipes\default.rb

```
#  
# Cookbook Name:: hosts  
# Recipe:: default  
# Copyright 2013, YOUR_COMPANY_NAME  
#  
# All rights reserved - Do Not Redistribute  
  
template "#{ENV['windir']}\\System32\\drivers\\etc\\hosts" do  
  source "hosts.erb"  
end
```

SAVE FILE!

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

Congrats, you just learned some Ruby!

Exercise: Open hosts.erb in your Editor

OPEN IN EDITOR: cookbooks\host\templates\default\hosts.erb

```
#This file is managed by the server <%= node[ "fqdn" ] %>
<% case node[ "datacenter" ][ "location" ] -%>
<% when "china" -%>
  0.0.0.0      nytimes.com
<% end -%>
```

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.

Exercise: Open host.erb in your Editor

OPEN IN EDITOR: cookbooks\host\templates\default\hosts.erb

```
#This file is managed by the server <%= node[ "fqdn" ] %>
<% case node[ "datacenter" ][ "location" ] -%>
<% when "china" -%>
  0.0.0.0    nytimes.com
<% end -%>
```

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

We introduce new syntax here – node attributes, de-referenced with a square bracket.

Exercise: Open host.erb in your Editor

OPEN IN EDITOR: cookbooks\host\templates\default\hosts.erb

```
#This file is managed by the server <%= node[ "fqdn" ] %>
<% case node[ "datacenter" ][ "location" ] -%>
<% when "china" -%>
  0.0.0.0      nytimes.com
<% 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

We introduce the “case” conditional here. We will include the next line if ‘node[“datacenter”][“location”]’ is China. Ask the class whether we are going to include it or not.

Remind them that as they get more datacenters up, they can include additional ‘when’ statements

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

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 IIS?")
- Attributes contain the details. ("What port do we run IIS on?")

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 hosts cookbook

```
PS\> knife cookbook upload hosts
```

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

Testing for syntax
Uploading files to the Chef Server

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

```
PS\> knife node run_list add node1 'recipe[hosts]'
```

```
node1:  
  run_list:  
    - recipe[iis_demo]  
    - recipe[hosts]
```

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 datacenter 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

```
remote@PS\> chef-client
```

What went wrong!

Exercise: Re-run the Chef Client

```
remote@PS\> chef-client
```

```
Template Context:  
-----  
on line #2  
1: #This file is generated by the chef-client on [fqdn] %>  
2: <% case node[  
3:   'chef-client':location %>  
4:     0.0.0.0  
5:   <% end -%>  
[2014-02-25T02:49:26-08:00] INFO: Running chef-client in easy mode  
[2014-02-25T02:49:26-08:00] FATAL: Mixin::TemplateError: undefined method `[]' for nil:NilClass  
Chef Client failed. 1 resources updated in 16.114976 seconds  
[2014-02-25T02:49:26-08:00] INFO: Sending resource update report (run-id: d659518e-d981-49dc-9735-9b3cf5324be2)  
[2014-02-25T02:49:26-08:00] FATAL: Chef::Mixin::TemplateError: undefined method `[]' for nil:NilClass
```

FAIL!

What went wrong!

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

```
Chef::Mixin::Template::TemplateError
-----
undefined method `[]' for nil:NilClass
Resource Declaration:
-----
# In C:/chef/cache/cookbooks/host/recipes/default.rb
10: template "C:\\Windows\\System32\\drivers\\etc\\hosts" do
11:   source "host.erb"
12: end
Compiled Resource:
-----
# Declared in C:/chef/cache/cookbooks/host/recipes/default.rb:9:in `from_file'
template("C:\\Windows\\System32\\drivers\\etc\\hosts") do
  provider Chef::Provider::Template
  action "create"
  retries 0
  .....
  .....
Template Context:
-----
on line #2
1: #This file is managed by server at <%= node["fqdn"] %>
2: <% case node["datacenter"]["location"] %>
3: <% when 'china' %>
4:   0.0.0.0    nytimes.com
5: <% else %>
[2014-02-25T02:49:26-08:00] INFO: Running queued delayed notifications before re-raising exception
[2014-02-25T02:49:26-08:00] FATAL: Stacktrace dumped to c:/chef/cache/chef-stacktrace.out
Chef Client failed. 1 resources updated in 16.114976 seconds
[2014-02-25T02:49:26-08:00] INFO: Sending resource update report (run-id: d659518e-d981-49dc-9735-9b3cf5324be2)
[2014-02-25T02:49:26-08:00] FATAL: Chef...Mixin...Template...TemplateError: undefined method `[]' for nil:NilClass
```

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)
- * ["datacenter"] (this is the method [], with an argument 'datacenter')
- ** this returns 'nil', because there is no 'location' attribute on the node, because we haven't evaluated the datacenter 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.

```
=====
Error executing action `create` on resource 'template[C:\\Windows\\System32\\drivers\\etc\\hosts]'
=====

Chef::Mixin::Template::TemplateError
-----
undefined method `[]' for nil:NilClass

Resource Declaration:
-----
# In C:/chef/cache/cookbooks/host/recipes/default.rb

10: template "C:\\Windows\\System32\\drivers\\etc\\hosts" do
11:   source "host.erb"
12: end
```

Wait for reveal

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

```
[2014-02-25T02:49:22-08:00] INFO: Run List expands to [iis_demo, hosts]
[2014-02-25T02:49:22-08:00] INFO: Starting Chef Run for node1
[2014-02-25T02:49:22-08:00] INFO: Running start handlers
[2014-02-25T02:49:22-08:00] INFO: Start handlers complete.
resolving cookbooks for run list: ["iis_demo", "hosts"]
[2014-02-25T02:49:23-08:00] INFO: Loading cookbooks [hosts, iis_demo]
```

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

nil:NilClass error

- The bottom line is when you see this error

```
undefined method `[]' for nil:NilClass
```

- It most often means you tried to look up a node attribute that does not exist!

Exercise: Add a dependency on the datacenter cookbook to the hosts cookbook

OPEN IN EDITOR: \cookbooks\hosts\metadata.rb

```
name          'hosts'
maintainer    'YOUR_COMPANY_NAME'
maintainer_email 'YOUR_EMAIL'
License        'All rights reserved'
description    'Installs/Configures hosts'
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
Version        '0.1.0'
depends        'datacenter'
```

The cookbook metadata file contains information about the cookbook
Some of it is mainly decorative - maintainer data, description, licensing

Exercise: Add a dependency on the datacenter cookbook to the hosts cookbook

OPEN IN EDITOR: \cookbooks\hosts\metadata.rb

```
name          'hosts'
maintainer    'YOUR_COMPANY_NAME'
maintainer_email 'YOUR_EMAIL'
License        'All rights reserved'
description    'Installs/Configures hosts'
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
Version        '0.1.0'

depends        'datacenter'
```

Cookbook Metadata

OPEN IN EDITOR: \cookbooks\hosts\metadata.rb

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

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 hosts cookbook

```
PS\> knife cookbook upload hosts
```

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

Exercise: Re-run the Chef Client

```
remote@PS\> chef-client
```

```
+#This file is managed by the server C1263834251
+ 0.0.0.0    nytimes.com

-# localhost name resolution is handled within DNS itself.
-# ...
-# ...
[2014-02-25T03:14:57-08:00] INFO: Chef run complete in 3.086 seconds
[2014-02-25T03:14:58-08:00] INFO: Re-running cookbooks/iis_demo[Default] from the cache;
it is no longer needed by any environments

Running handlers:
[2014-02-25T03:14:58-08:00] INFO: Running report handlers
Running handlers complete
[2014-02-25T03:14:58-08:00] INFO: Report handlers complete
Chef Client finished, 2/5 resources updated in 15.412866 seconds
[2014-02-25T03:14:58-08:00] INFO: Sending resource update report (run-id: fcfcfd6d8-3bf6-4965-b71c-
caae1ef53ebf)
```

WIN!

Exercise: Check your work

```
remote@PS\> gc C:\Windows\System32\drivers\etc\hosts
```

```
# This file is managed by the server C1263834251  
0.0.0.0      nytimes.com
```

Exercise: Check automated backups

```
remote@PS\> gci C:\chef\backup\Windows\System32\drivers\etc
```

Mode	LastWriteTime	Length	Name
----	-----	----	----
-a--	7/25/2012 10:26 PM	824	hosts.chef-20140225031457.550034

Exercise: Show your test node's datacenter attribute

```
PS\> knife search node "*:*" -a datacenter
```

```
1 items found
target1:
  datacenter:
    location: china
```

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: Change your node's Location

OPEN IN EDITOR: cookbooks\datacenter\attributes\default.rb

```
default[ "datacenter" ][ "location" ] = "russia"
```

SAVE FILE!

Exercise: Upload the datacenter cookbook

```
PS\> knife cookbook upload datacenter
```

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

Exercise: Re-run the Chef Client

```
remote@PS\> chef-client
```

```
* template[C:\Windows\System32\drivers\etc\hosts] action create[2014-02-25T03:26:02-08:00]
INFO: Processing template[C:\Windows\System32\drivers\etc\hosts] action create (hosts::default
line 9)
[2014-02-25T03:26:02-08:00] INFO: template[C:\Windows\System32\drivers\etc\hosts] backed up to
c:/chef/backup\Windows\System32\drivers\etc\hosts.chef-20140225032602.580921
[2014-02-25T03:26:02-08:00] INFO: template[C:\Windows\System32\drivers\etc\hosts] updated file
contents C:\Windows\System32\drivers\etc\hosts

- update content in file C:\Windows\System32\drivers\etc\hosts from 94a261 to 1f8bdf
  --- C:\Windows\System32\drivers\etc\hosts      2014-02-25 03:14:57.000000000 -0800
  +++ C:/Users/chef/AppData/Local/Temp/4/chef-rendered-template20140225-4940-1y6uwwv
2014-02-25 03:26:02.000000000 -0800
@@ -1,4 +1,3 @@
 #This file is managed by server at C1263834251
- 0.0.0.0    nytimes.com

[2014-02-25T03:26:03-08:00] INFO: Chef Run complete in 3.338407 seconds
```

Notice we downloaded the datacenter cookbook, and we rendered etc\hosts

Exercise: Check your work

```
remote@PS\> gc C:\Windows\System32\drivers\etc\hosts
```

```
# This file is managed by the server C1263834251
```

Exercise: Show your test node's datacenter attribute

```
PS\> knife node show node1 -a datacenter
```

```
target1:  
  datacenter:  
    location: russia
```

Congratulations!

- You now know 3 very important resources for Windows configuration management
 - Powershell_script
 - Template
 - Service

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.

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 important resources in Windows configuration management?

Day One End

-) Configuration parameters, default values
-) default, override, automatic
-) Anytime you want values/attributes evaluated/used from another cookbook/recipe
-) Evaluate the following ruby and insert it inline (print)
-) package, template, service

Template Variables, Notifications, and Controlling Idempotency

Refactoring the iis_demo Cookbook

v2.1.1_WIN



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 powershell resources
 - Control idempotence manually with `not_if` and `only_if`
 - Navigate the Resources page on docs.opscode.com
 - Describe the Directory resource
 - Implement resource notifications
 - Explain Template Variables, 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 pages

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\iis_demo\metadata.rb

```
name          'iis_demo'
maintainer    'YOUR_COMPANY_NAME'
maintainer_email 'YOUR_EMAIL'
license        'All rights reserved'
description    'Installs/Configures iis_demo'
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/>

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 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 in the iis_demo cookbook

OPEN IN EDITOR: \cookbooks\iis_demo\attributes\default.rb

```
default["iis_demo"]["sites"]["clowns"] = { "port" => 80 }
default["iis_demo"]["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

<<Review comment: tcp/82 is closed on cloudshare's firewall, so changed this from 81 & 82 to 80 & 81

The marketing department loves clowns and bears. If we build pages with clowns and bears, the revenue will come.

Two schools of thought: start with data model and write cookbook or vice versa (model data to cookbook). This is preference. Here we start with data model.

Also note auto-vivification of hashes: These attributes are a ruby hashes of hashes with keys and values. Top level attribute apache has a hash of sites, one = clowns, one = bears, each has a hash value. Key of port with value = integer. Why nested under "iis_demo"? Doesn't *need* to be for this example. Common best practice. Nest attributes into what they belong to. Since Chef will automagically create any intermediate-level hashtables automagically if they don't exist, this is extremely useful.

Alternative: write a cookbook using data bags, but that's the next module.

Exercise: Open the default iis_demo recipe in your editor

OPEN IN EDITOR: \cookbooks\iis_demo\recipes\default.rb

```
...
#
powershell_script "Install IIS" do
  code "add-windowsfeature Web-Server"
  action :run
end

service "w3svc" do
  action [:enable, :start]
end

node.default["iis_demo"]["indexfile"] = "Default2.htm"
cookbook_file "c:\\inetpub\\wwwroot\\Default.htm" do
  source node["iis_demo"]["indexfile"]
  rights :read, "Everyone"
end
```

SAVE FILE!

<<WIP Need to list tasks on the slide to be inline w/ linux course

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

There are a few things we can pick on. Notably, our hardcoded paths. But first things first...

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

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

In this case, add-windowsfeature Web has built-in idempotency

Enter the `not_if` and `only_if` metaparameters

```
not_if "C:\\Windows\\\\System32\\\\inetsrv\\  
\\appcmd.exe list apppool #{site_name}"
```

- 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

`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

Building the resource collection - revisited

- It is important to understand the difference between these two pieces of code

```
execute "start-iis" do
  not_if <code to check if IIS is running>
  notifies :start, "service[iis]"
end
```

This code is placed in the resource collection during the 'compile' phase and executed during the 'execute' phase

```
if !<code to check if IIS is running>
  execute "start-iis" do
    notifies :start, "service[iis]"
  end
end
```

This code is executed during the 'compile' phase before any Chef DSL
Executing code during the compile phase could potentially prevent the resource from getting added to the resource collection

Resource collection is created during the compile phase.

Pure Ruby code is executed in the 'compile' phase - Chef DSL is executed in the 'execute' phase

The first one means the resource will only be invoked if the condition exists, whereas the second means the resource will only get added to the resource collection if the condition exists.

The impact of the code executing during the compile phase is that it could potentially prevent the resource from getting added to the resource collection.

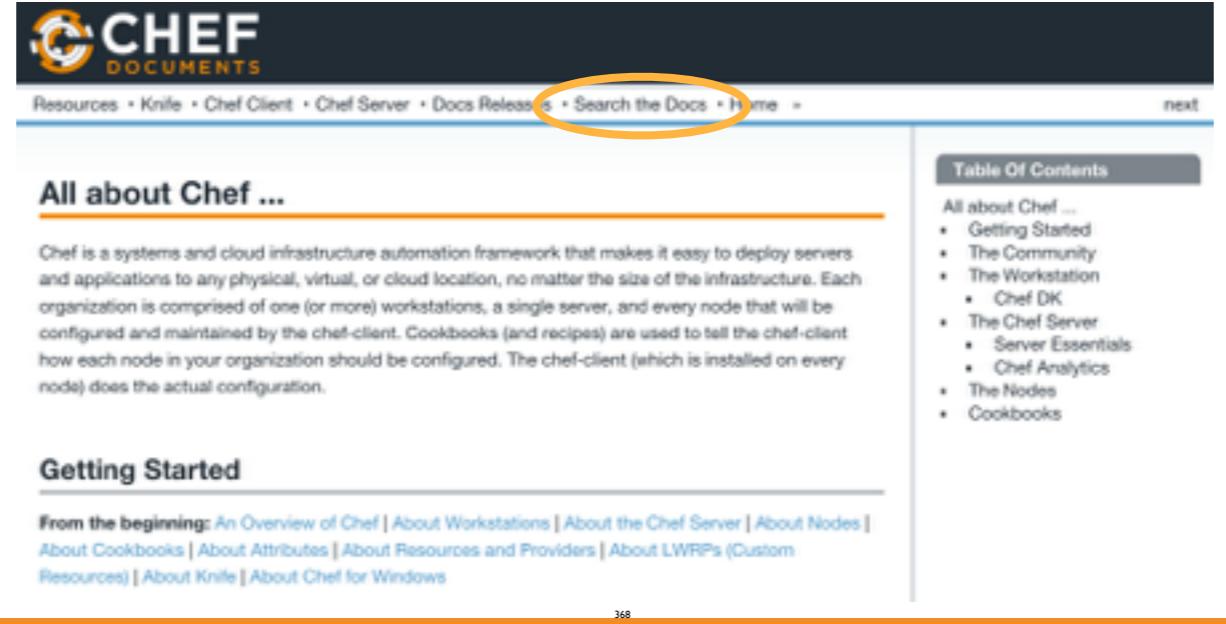
Best Practice: The Chef Docs Site

- 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.chef.io>
- Let's use the docs to learn more about **not_if** and **only_if**

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 web browser displaying the Chef Documentation homepage. The page has a dark header with the 'CHEF DOCUMENTS' logo. Below the header, there is a navigation bar with links: 'Resources', 'Knife', 'Chef Client', 'Chef Server', 'Docs Releases', 'Search the Docs' (which is circled in yellow), and 'Home'. To the right of the navigation bar, there is a 'next' link. The main content area has a section titled 'All about Chef ...' with a sub-section 'Getting Started'. Below 'Getting Started', there is a list of related topics: 'From the beginning: An Overview of Chef | About Workstations | About the Chef Server | About Nodes | About Cookbooks | About Attributes | About Resources and Providers | About LWRPs (Custom Resources) | About Knife | About Chef for Windows'. On the right side of the content area, there is a 'Table Of Contents' sidebar with a hierarchical tree structure:

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

Exercise: Search for more information about Resources

Search the Documentation for Chef

From here you can use a scoped Google search query to search all of the documentation about Chef that is located at docs.getchef.com. (This page requires JavaScript be enabled to view the search box.)

All Chef Documentation Cookbooks

About 44 results (0.31 seconds)

Sort by: Relevance

[Common Functionality — Chef Docs](#)

https://docs.getchef.com/resource_common.html

A guard attribute can be used to evaluate the state of a node during the execution phase of the chef-client run. Based on the results of this evaluation, a guard ...
Labeled Chef ...

[powershell_script — Chef Docs](#)

https://docs.getchef.com/resource_powershell_script.html

Table Of Contents. powershell_script. Syntax; Actions; Attributes; Guards; Providers; Examples ... Use

369

The Resources Page

The screenshot shows the 'Common Functionality' page from the Chef Documentation. The page includes a navigation bar with links to Resources, Knife, Chef Client, Chef Server, Docs Releases, Search the Docs, and Home. Below the navigation is a 'Table Of Contents' sidebar with sections like Actions, Examples, Attributes, Guards, and Guard Interpreters. The 'Guard Interpreters' section is circled in yellow. The main content area contains a heading 'Common Functionality' and a paragraph about common actions for resources. It also features a table titled 'Actions' with one row for 'nothing'. The page footer indicates it is page 370.

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.

Chef resources on Windows

- Resources in Chef that work on Windows:
 - `file`, `remote_file`, `cookbook_file`, `template`
 - `registry`
 - `directory`, `remote_directory`
 - `user`, `group`
 - `mount`
 - `env`
 - `service`
 - `execute`
 - `ruby_block`, `powershell_script`, `batch`
- Check the Docs...

Exercise: Stop the Default Web Site with Powershell resource

OPEN IN EDITOR: \cookbooks\iis_demo\recipes\default.rb

```
service "w3svc" do
  action [:enable, :start]
end

#node.default["iis_demo"]["indexfile"] = "Default2.htm"
#cookbook_file "c:\\inetpub\\wwwroot\\Default.htm" do
#  source node["iis_demo"]["indexfile"]
#  rights :read, "Everyone"
#end

powershell_script "disable default site" do
  code 'get-website "Default Web Site*" | where {$_.state -ne "Stopped"} | Stop-Website'
end
```

SAVE FILE!

- Comment out the cookbook_file resource from recipe
- Use powershell resource to stop Default Web Site if running.

Don't forget the * after "Default Web Site"

This is a known bug in windows, without this * get-website will return all websites instead of just the one you want.

Idempotency Guarantees

```
powershell_script "disable default site" do
  code 'get-website "Default Web Site*" | where {$_.state -ne "Stopped"} | Stop-Website'
end
```

- With `powershell_script` resource it's up to you to make resources idempotent
- Either do it in Powershell, or Chef idempotency guards (later)

There is an outstanding bug to have `not_if` run within the context of parent resource. As of now `not_if` starts as a CMD

Exercise: Iterate over each IIS site

OPEN IN EDITOR: \cookbooks\iis_demo\recipes\default.rb

```
powershell_script "disable default site" do
  code 'get-website "Default Web Site*" | where {$_.state -ne "Stopped"} | Stop-Website'
end

node["iis_demo"]["sites"].each do |site_name, site_data|
  site_dir = "#{ENV['SYSTEMDRIVE']}\\inetpub\\wwwroot\\#{site_name}"
  directory site_dir

```

SAVE FILE!

- node["iis_demo"]["sites"] is the Ruby hash, with keys and values we defined in our default attributes

We will explain the new syntax in the next few slides

Exercise: Iterate over each IIS site

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

- Calling `.each` loops over each site

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

- **First pass**

- `site_name = "clowns"`
- `site_data = { "port" => 80 }`

- **Second pass**

- `site_name = "bears"`
- `site_data = { "port" => 81 }`

More new syntax, this time “`.each`”. 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.

Exercise: Iterate over each IIS site

```
node["iis_demo"]["sites"].each do |site_name, site_data|
  site_dir = "#{ENV['SYSTEMDRIVE']}\\inetpub\\wwwroot\\#{site_name}"
  directory site_dir
```

- Store directory path in a variable called **site_dir**
- **directory** resource creates a directory
- **#{ENV['SYSTEMDRIVE']}** – insert local environment variable
- **#{site_name}** means “insert the value of **site_name** here”

Call out that `site_dir` is already a string so we don't have to interpolate it.

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

`#{..}` in a string

Scoping: var exists on this block and only this block. Globals have \$ for global. In java, php, perl: \$ just means variable. In ruby, it means global var. (vs. local _, instance @, or class @@)

Exercise: Iterate over each IIS site

```
node["iis_demo"]["sites"].each do |site_name, site_data|
  site_dir = "#{ENV['SYSTEMDRIVE']}\\inetpub\\wwwroot\\#{site_name}"
  directory site_dir
```

- First pass
 - The value of `site_dir` is the string “`c:\\inetpub\\wwwroot\\clowns`”
- Second pass
 - The value of `site_dir` is the string “`c:\\inetpub\\wwwroot\\bears`”

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

`#{} in a string`

Scoping: var exists on this block and only this block. Globals have \$ for global. In java, php, perl: \$ just means variable. In ruby, it means global var. (vs. local _, instance @, or class @@)

Pop Quiz: Variables and interpolation

- Given the following variable for 'document_root'

```
site_dir="c:\foo\foobar"
```

- What directory will be created in each of the following?

Directory <code>site_dir</code> do ...	c:\foo\bar
<code>directory "site_dir"</code> do ...	c:\site_dir
<code>directory "#{site_dir}"</code> do ...	c:\foo\bar

This is an optional slide – 1/3!

Note this is a building slide for student participation – the answers reveal one at a time on click

Wait for the big reveal!

This is a building slide. Ask the class for answer to each one, and reveal one at a time

Answer:-

Case1 – /srv/apache/foobar

Case2 – /document_root

Case3 – /srv/apache/foobar

Exercise: Add an iis_site resource to install the new sites

```
node[ "iis_demo" ][ "sites" ].each do |site_name, site_data|
  site_dir = "#{ENV['SYSTEMDRIVE']}\\inetpub\\wwwroot\\#{site_name}"

  directory site_dir

  powershell_script "create app pool for #{site_name}" do
    code "New-WebAppPool #{site_name}"
    not_if "C:\\Windows\\System32\\inetsrv\\appcmd.exe list apppool #{site_name}"
  end
end
```

SAVE FILE!

- Create App Pool, but only if one doesn't already exist.

1st Pass – we create an iis_site resource for the “clowns” site

2nd Pass – we create an iis_site resource for the “bears” site

We encounter 3 new pieces of syntax here:

1) do..end being used as a block (and sub-block). depending on the crowd, explain anonymous functions and lambdas, or just explain that it means “execute this code and return the result”.

iis_site "string" plus whole block. Checks valid syntax. But code block gets called later when we are ready to call the function. It gets read in very early. But not_if is executed when the function is called. If executed outside the block, it would execute at compile time not at action time (and would always fail). So when you're wrapping functions, you are declaring when they should be executed. We create objects and then do stuff with them conditionally. Nesting is for deferring execution. Which is why we need special nonruby syntax to defer.

2) File – this is a class name

3) File.exists? 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 few slides go into greater detail.

Exercise: Add a Web Site with Powershell_script resource

```
powershell_script "create app pool for #{site_name}" do
  code "New-WebAppPool #{site_name}"
  not_if "C:\\Windows\\System32\\inetsrv\\appcmd.exe list apppool #{site_name}"
end

powershell_script "new website for #{site_name}" do
  code <<-EOH
  Import-Module WebAdministration
  if (-not(test-path IIS:\\Sites\\#{site_name})){
    $NewWebsiteParams = @｛Name= '#{site_name}';Port= #{site_data["port"]};PhysicalPath= '#{site_dir}';ApplicationPool= '#{site_name}'｝
    New-Website @NewWebsiteParams
  }
  elseif ((Get-WebBinding -Name #{site_name}).bindingInformation -ne '*:#{{site_data["port"]}}') {
    $CurrentBinding = (Get-WebBinding -Name #{site_name}).bindingInformation
    $BindingParameters = @｛Name= '#{site_name}';Binding= $CurrentBinding;PropertyName= 'Port';Value = #{site_data["port"]}｝
    Set-WebBinding @BindingParameters
  }
  Get-Website -Name #{site_name} | Where {$_.state -like 'Stopped'} | Start-Website
  EOH
end
```

SAVE FILE!

- Create Web Site, but only if one doesn't already exist.

Note: You see "Index out of bound" error - chances are student manually deleted Default Website. Simply recreate the default website and the error will go away.

* **Import-Module webadministration** – need to load this because IIS: provider is not loaded by default. Quirks of windows.

* **if(-not(test-path....** – same as ! in ruby. Can be supplemented with a !, –not makes it slightly easier for the noobies.

New-Website takes all the same parameters as IIS_Site except that you have to explicitly specify Application Pool.

<<-EOH stuff is known as a Here Document - https://en.wikipedia.org/wiki/Here_document in case people ask.

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

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

In this class, within the Template block we only ever use node attributes. Not all your data may be attributes (e.g. data from your own DBS – chef is good at integrating with other data sources). But let's walk before we run.

Notifications

```
notifies :restart, "service[w3svc]"
```

- 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 iis”
- 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

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: Use a template to create Default.htm for each site

```
.... #{site_dir} -ApplicationPool #{site_name}
      }
EOH
end

template "#{site_dir}\\Default.htm" do
  source "Default.htm.erb"
  rights :read, "Everyone"
  variables(
    :site_name => site_name,
    :port => site_data["port"]
  )
  notifies :restart, "service[w3svc]"
end
end
```

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

Exercise: Add a file resource to set the permissions on the Default.htm files

```
template "#{site_dir}\\Default.htm" do
  source "Default.htm.erb"
  rights :read, "Everyone"
  variables(
    :site_name => site_name,
    :port => site_data["port"]
  )
  notifies :restart, "service[w3svc]"
end
```

end

- Don't forget the last `end`
- Delete the rest of the lines below

I don't think this slide is needed...

The entire recipe



<http://tinyurl.com/pelqotr>

Gotta do something about making this slide pretty

Full URL: <https://gist.github.com/smurawski/ae9187fb8a5fcfedc5ee2>

Exercise: Add Default.htm.erb to your templates directory

OPEN IN EDITOR: cookbooks\iis_demo\templates\default\Default.htm.erb

```
<html>
  <body>
    <h2>We love <%= @site_name %></h2>
    <%= node["ipaddress"] %>:<%= @port %>
  </body>
</html>
```

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

<<Review comment: 'Company' attribute is no longer set (was a facet of 'knife node edit'), so removed the line <h1>Welcome to <%= node["company"] %></h1>

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.

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

Gist: <https://gist.github.com/2866421>

Exercise: Upload the iis_demo cookbook

```
PS\> knife cookbook upload iis_demo
```

```
Uploading iis_demo [ 0.2.0 ]
Uploaded 1 cookbook.
```

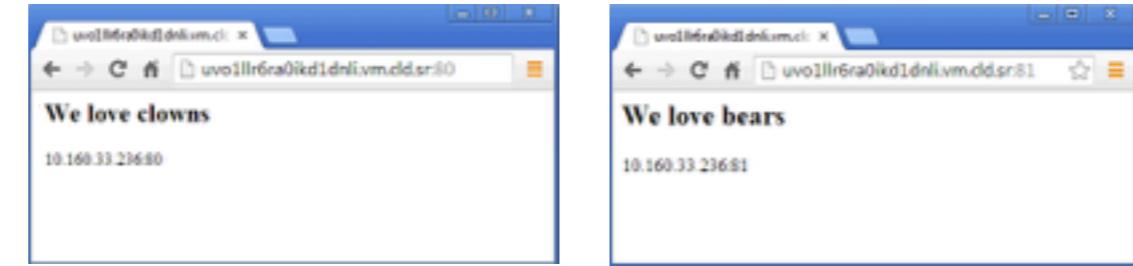
Exercise: Re-run the Chef Client

```
remote@PS> chef-client
```

```
"C:/Users/ADMINI~1/AppData/Local/Temp/2/chef-script20130918-1312-ebgdli.ps1"
  * template[C:/inetpub/wwwroot/bears/Default.htm] action create
INFO: Processing template[C:/inetpub/wwwroot/bears/Default.htm] action create
(iis_demo::default line 45)
  (up to date)
Recipe: hosts::default
  * template[C:/Windows\System32\drivers\etc\hosts] action create
INFO: Processing template[C:/Windows\System32\drivers\etc\hosts] action create
(hosts::default line 11)
  (up to date)
[2013-09-18T21:02:52+00:00] INFO: Chef Run complete in 13.1508 seconds
[2013-09-18T21:02:52+00:00] INFO: Running report handlers
[2013-09-18T21:02:52+00:00] INFO: Report handlers complete
Chef Client finished, 6 resources updated
```

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

Exercise: Verify our two sites are working!



Again, if they have trouble getting to them with a web browser, do it from within the VM

<http://localhost:80>

<http://localhost:81>

I

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

- Recipes contain the pattern for how to do something. (“How we deploy IIS virtual hosts?”)
- Attributes contain the details. (“What virtual hosts should we deploy?”)

This is a re-iteration from earlier, but it's one of the most important things to understand about Chef.

Questions

- How do you control the idempotence of an `powershell_script` 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?

-) You need to code it into the commands you write or are executing.. Chef DOES NOT do this.

-) docs.chef.io

-) Used to trigger an action on another resource when the current resource executes successfully (eg. edit a config file then restart service)

-) Let's you pass in custom data as a variable; data that isn't a node attribute.

-) Insert value of variable 'foo' here.

Search

A deeper dive....

v2.1.1_WIN



This lesson is entirely optional, and depends if you've time or running behind.

Lesson Objectives

- After completing the lesson, you will be able to
 - Describe the query syntax used in search
 - Invoke a search from command line
 - Build a search into your recipe code

http://docs.opscode.com/essentials_search.html

What is search?

- Use search to query data indexed on the chef server
- Syntax is

```
knife search INDEX SEARCH_QUERY
```

- Where index can be client, environment, node, role, (or the name of a data bag)
- Search query **runs on** the server and is **invoked from** within a recipe, using knife or via WebUI

More in data bags later...

<http://wiki.apache.org/solr/SolrQuerySyntax>

Exercise: Use knife search

```
PS\> knife search node "*:*"
```

```
Node Name:    node1
Environment:  _default
FQDN:        C1263834251
IP:          10.160.33.236
Run List:    recipe[iis_demo], recipe[hosts]
Roles:
Recipes:     iis_demo, hosts, iis_demo::default, hosts::default
Platform:   windows 6.2.9200
Tags:
```

Search for all nodes

Query Syntax

- Chef search uses the Solr 'key:search_pattern' syntax

```
knife search node "ipaddress:10.20.30.40"
```

- Use an asterisk ("*") for >=0 chars in a wildcard search

```
knife search node "ipaddress:10.*"
```

```
knife search node "platfo*:windows"
```

- Use a question mark ("?") to replace a single character

```
knife search node "platform_version:6.2.920?"
```

platfor* could match "platform" or "platform_family"

Indexing and search delay

- Search is based on an index, so if you write to a node it may not be immediately available

```
node.default["serveradmin"] = "Jane"
search("node", "*:*").each do |server|
  user servers["serveradmin"]
end
```

BAD PRACTICE!

- Here, since the attribute value has just been written the search will not pick it up until the index is rebuilt
- Also, node attributes changed during a Chef run aren't posted to the server until the entire Chef run is successful
- But the value is stored in memory - so no need to search!

397

It's all about eventual consistency -- individual Chef runs are immediately convergent, but your infrastructure as a whole is eventually consistent

Also, node attributes changed during a Chef run aren't posted to the server until the entire Chef run is successful

Remember what you're searching for!

```
use knife          for nodes  
↓                ↓  
knife search node 'ipaddress:10.1.1.*'  
      ↑          ↑  
    to search     with this search criteria
```

- A successful search returns **all objects**, not just the attributes, that satisfy the search criteria
- In other words 'search for nodes containing this attribute', **NOT** 'search for this attribute across nodes'!

So

'knife search node 'ipaddress:10.1.1.*'"
means:

"find all nodes that contain the attribute 'ipaddress:10.1.1.*', and return those nodes"

It **DOES NOT** mean

"find 'ipaddress:10.1.1.*' across all nodes and return those attribute values"

i.e. It returns **nodes** 'node1', 'node2' etc, and not a list of **attributes** 'ipaddress:10.1.1.1', 'ipaddress:10.1.1.2' etc

Exercise: Find a node with the specific attribute value

```
PS\> knife search node "ipaddress:10.*"
```

```
1 items found

Node Name:    node1
Environment:  default
FQDN:        C1263834251
IP:          10.160.33.236
Run List:    recipe[iis_demo], recipe[hosts]
Roles:
Recipes:     iis_demo, hosts, iis_demo::default, hosts::default
Platform:   windows 6.2.9200
Tags:
```

Here we are searching for nodes. Output is shortened by default – to see all attributes for the node use the '-l' switch

Note student IP may be different – use the result on previous slide

Exercise: Using Knife to find an attribute value

```
PS\> knife search node "*:*" -a ipaddress
```

```
1 items found

node1:
  ipaddress: 10.160.33.236
```

Here we are trying to find the IP Address for our nodes, i.e. the value of the 'ipaddress' attribute

Exercise: Return just the attribute value

```
PS\> knife search node "ipaddress:10.*" -a ipaddress
```

```
1 items found

node1:
  ipaddress: 10.160.33.236
```

Here we are searching for a node. Output is shortened by default – to see all attributes for the node use the '-l' switch

Note student IP may be different – use the result on previous slide

Exercise: Boolean matching

```
PS\> knife search node "ipaddress:10* AND platform:windows"
```

```
1 items found

Node Name:    node1
Environment:  _default
FQDN:        C1263834251
IP:          10.160.33.236
Run List:    recipe[iis_demo], recipe[hosts]
Roles:
Recipes:     iis_demo, hosts, iis_demo::default, hosts::default
Platform:   windows 6.2.9200
Tags:
```

Make sure 'AND' is in upper case

Exercise: Boolean matching

```
PS\> knife search node "ipaddress:[10.0.* TO 10.2.*]"
```

```
1 items found

Node Name:    node1
Environment: _default
FQDN:        C1263834251
IP:          10.160.33.236
Run List:    recipe[iis_demo], recipe[hosts]
Roles:
Recipes:     iis_demo, hosts, iis_demo::default, hosts::default
Platform:   windows 6.2.9200
Tags:
```

For more details see http://docs.chef.io/knife_search.html

Search from within a recipe

- You can invoke a search within a recipe, and use the results to apply further Chef primitives

```
search(INDEX, SEARCH_QUERY).each do |result|
  foo
end
```

- For example

```
search("node", "role:webserver").each do |webserver|
  [register webserver with load balancer]
  or
  [configure firewall so webserver can access database]
end
```

Exercise: Search for an attribute in a recipe

OPEN IN EDITOR: cookbooks\iis_demo\recipes\ip-logger.rb

```
search("node","platform:windows").each do |server|
  log "The Windows servers in your organization have the following
FQDN/IP Addresses:- #{server['fqdn']}/#{server['ipaddress']}"
end
```

SAVE FILE!

Exercise: Upload the iis_demo cookbook

```
PS\> knife cookbook upload iis_demo
```

```
Uploading iis_demo      [ 0.2.0 ]
Uploaded 1 cookbook.
```

Recipe Naming

- Recipes are referenced using the notation '**cookbook::recipe**', where 'recipe' is the name of the '.rb' file in the "cookbook/recipe" directory
- The "default.rb" recipe for a given cookbook may be referred to just the cookbook name (e.g. 'mysql')
- If we added another recipe to this cookbook named 'backup', we would refer to it as 'mysql::backup'

407

Repeating slide – review

Sometimes, you might have to manage a complex application that may have different functions (server vs. client) or an extensive plugin system (httpd server modules). It is often helpful to break apart the logic used to manage these different aspects from the default recipe into additional recipes that can be added as necessary.

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

```
PS> knife node run_list add node1 'recipe[iis_demo::ip-logger]'
```

```
node1:
  run_list:
    - recipe[iis_demo]
    - recipe[hosts]
    - recipe[iis_demo::ip-logger]
```

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

```
remote@PS:> chef-client
```

```
Recipe: iis_demo::ip-logger
  * log[The Windows servers in your organization have the following FQDN/
    IP Addresses:- C1263834251/10.160.33.236] action
  write[2014-02-25T06:19:43-08:00] INFO: Processing log[The Windows servers
    in your organization have the following FQDN
    /IP Addresses:- C1263834251/10.160.33.236] action write (iis_demo::ip-
    logger line 2)
  [2014-02-25T06:19:43-08:00] INFO: The Windows servers in your
    organization have the following FQDN/IP Addresses:-
    C1263834251/10.160.33.236
```

Now run chef-client on the node

Exercise: Remove the recipe to your test node's run list

```
PS\> knife node run_list remove node1 'recipe[iis_demo::ip-logger]'
```

```
node1:  
  run_list:  
    recipe[iis_demo]  
    recipe[hosts]
```

Just remove the recipe from the run_list again to tidy up after the lab.

Questions

- What search engine is chef search built upon?
- What is searchable in chef?
- What character can you use for a wildcard search in SOLR?
- Why should you not set an attribute and then immediately search for it?

-)

- Solr
- client, environment, node, role and name of databag(s)
- *
- Delay in index generation

Recipe Inclusion, Data Bags, and Search

Writing a Users cookbook

v2.1.1_WIN



Lesson Objectives

- After completing the lesson, you will be able to
 - Explain what Data Bags are, and how they are used
 - Describe 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

These “containers” take JSON. Anything you can do in JSON you can do in a databag.

Make a data_bags directory

```
PS \> md data_bags
```

```
Directory: C:\windows\temp\chef-repo
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	25-Feb-14 3:23 PM		data_bags

Exercise: Create a data bag named users

```
PS \> md data_bags\users  
PS \> knife data_bag create users
```

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d---	25-Feb-14 3:24 PM		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",  
  "password": "iAm@Cl0wN!!!"  
}
```

SAVE FILE!

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.

<https://gist.github.com/vinvar/6616139>

Exercise: Create the data bag item

```
PS\> knife data_bag from file users bobo.json
```

```
Updated data_bag_item[users::bobo]
```

Exercise: Create a user item in the users data bag

OPEN IN EDITOR: data_bags\users\frank.json

```
{  
  "id": "frank",  
  "comment": "Frank Belson",  
  "password": "sP3nC3r4Hire!"  
}
```

SAVE FILE!

Frank Belson is the sergeant of the police precinct in the Spenser novels by Robert B. Parker (later: Spenser for Hire TV series)

<https://gist.github.com/vinyar/6616289>

Exercise: Create the data bag item

```
PS\> knife data_bag from file users frank.json
```

```
Updated data_bag_item[users::frank]
```

Exercise: Show all the users in Chef

```
PS\> knife search users '*:*'
```

```
2 items found

chef_type: data_bag_item
comment: Frank Belson
data_bag: users
id: frank
password: sP3nC3r4Hire!

chef_type: data_bag_item
comment: Bobo T. Clown
data_bag: users
id: bobo
password: iAm@Clo0wN!!!
```

What's important here is that more than just the nodes get indexed for search – it is everything in Chef!

If they don't see this right away after uploading the data bags, ask the students to wait a few seconds and try again. Sometimes there can be lag after uploading a data bag, because the information is being indexed.

Exercise: Find Bobo's id in Chef

```
PS\> knife search users 'comment:Bobo T. Clown' -a id
```

```
1 items found
```

```
data_bag_item_users_bobo:  
  id: bobo
```

Encourage students to try searches: knife help search
You can use boolean values e.g. knife search INDEX 'field:value OR field:value'

Exercise: Create a data bag named groups

```
PS\> mkdir data_bags\groups  
PS\> knife data_bag create groups
```

Mode	LastWriteTime	Length	Name
d----	9/18/2013 3:07 PM		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",  
  "members": [ "bobo", "frank" ]  
}
```

SAVE FILE!

JSON syntax again.
Arrays in JSON have brackets.
Quotes around strings.
No trailing commas.

<https://gist.github.com/vinyar/6616444>

Exercise: Create the data bag item

```
PS\> knife data_bag from file groups clowns.json
```

```
Updated data_bag_item[groups::clowns]
```

Exercise: Show all the groups in Chef

```
PS\> knife search groups '*:*'
```

```
1 items found

chef_type: data_bag_item
data_bag:   groups
id:        clowns
members:
  bobo
  frank
```

Exercise: Create a cookbook named 'users'

```
PS\> 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: \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" ]
    password user_data[ "password" ]
  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

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"]
    password user_data["password"]
  end
end

include_recipe "users::groups"
```

- `include_recipe` ensures another recipes resources are complete before we continue
- Chef will only include each recipe once

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 ASP.NET 4.5 runtime existing for your ASP.NET application, for example)
- Always use `include_recipe` to include it specifically, even if you put it in a run list

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
    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

```
PS\> knife cookbook upload users
```

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

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

```
PS\> knife node run_list add node1 'recipe[users]'
```

```
node1:  
  run_list:  
    - recipe[iis_demo]  
    - recipe[hosts]  
    - recipe[users]
```

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

```
remote@PS\> chef-client

...
Recipe: users::default
  * user[bobo] action create[2014-02-25T07:56:24-08:00] INFO: Processing user[bobo] action
create (users::default line 10)
[2014-02-25T07:56:24-08:00] INFO: user[bobo] created

  - create user user[bobo]
  * user[frank] action create[2014-02-25T07:56:24-08:00] INFO: Processing user[frank] action
create (users::default line 10)
[2014-02-25T07:56:24-08:00] INFO: user[frank] created

  - create user user[frank]
Recipe: users::groups
  * group[circus-monkeys] action create[2014-02-25T07:56:24-08:00] INFO: Processing group[circus-monkeys]
action create (users::groups line 2)
[2014-02-25T07:56:24-08:00] INFO: group[circus-monkeys] created

  - create group[circus-monkeys]
[2014-02-25T07:56:25-08:00] INFO: Chef Run complete in 6.203709 seconds
```

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

```
remote@PS\> net users
```

```
User accounts for \\C1263834251
-----
Administrator          bobo
frank                  Guest
The command completed successfully.
```

```
remote@PS\> net localgroup
```

```
Aliases for \\WIN-BBNT36Q0RB2
*Administrators
...
*circus
*Performance Log Users
```

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 in about ~40 lines.
 - (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

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!

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 IIS recipe?
- Where would you go to find out more?

-) Container for data that is one to many relationship with nodes (not 1:1)
-) Once defined and applied to data bag objects in chef, search/query against them.
-) Creates and manages system user accounts
-) Instructs chef to 'get' and execute another recipe at this position before continuing; recipe reuse (not duplicating work)
-) Works by parsing results via a loop and setting desired results to values
-) Find list of servers in a particular role and add to load balancer configuration?
-) Instead of attributes, could have set to values in data bags
-) [docs.chef.io](#)

Roles

Role-based Attributes and Merge Order Precedence

v2.1.1_WIN



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
 - Explain how 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

Source control history can act as a documentation trail. Helpful if you need to unentangle your infrastructure.

Exercise: Create a role named webserver

```
PS\> md roles
```

Mode	LastWriteTime	Length	Name
d----	25-Feb-14 3:07 PM	-----	roles

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[iis_demo]"
default_attributes({
  "iis_demo" => {
    "sites" => {
      "admin" => {
        "port" => 8000
      }
    }
  }
})
```

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[iis_demo]"
default_attributes({
  "iis_demo" => {
    "sites" => {
      "admin" => {
        "port" => 8000
      }
    }
  }
})
```

SAVE FILE!

Exercise: Create the role

```
PS\> knife role from file webserver.rb
```

```
Updated Role webserver!
```

Exercise: Show the role with knife

```
PS\> knife role show webserver
```

```
chef_type:          role
default_attributes:
  iis_demo:
    sites:
      admin:
        port: 8000
  description:      Web Server
  env_run_lists:
  json_class:       Chef::Role
  name:             webserver
  override_attributes:
  run_list:
```

Exercise: Search for the roles that have recipe[iis_demo] in their run list

```
PS\> knife search role 'run_list:recipe\[iis_demo\]'
```

```
1 items found

chef_type:          role
default_attributes:
  iis_demo:
    sites:
      admin:
        port: 8000
    description:  Web Server
    env_run_lists:
    json_class:   Chef::Role
    name:         webserver
    override_attributes:
      run_list:    recipe[iis_demo]
```

note: knife command on the next (**hidden**) slide

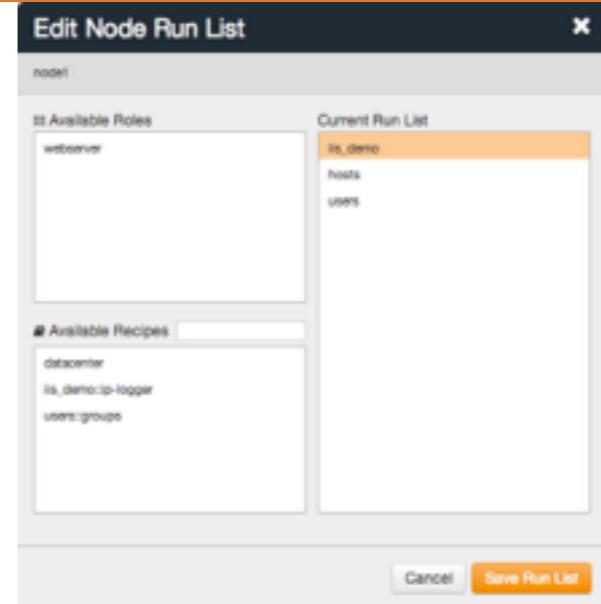
This is a hard one to do without the key above, but it's worth it. Make them try and figure it out from the docs's "search" page and the hint that it is "solr syntax".

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[iis_demo] with role[webserver] in run list

- Click the ‘Nodes’ tab then select node ‘node1’
- Click ‘Edit Run List’ from left navigation bar
- Drag ‘iis_demo’ 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’



Exercise: Re-run the Chef Client

```
Remote@PS\> chef-client
```

```
[2014-02-25T09:44:14-08:00] INFO: Run List is [role[webserver],  
recipe[hosts], recipe[users]]  
[2014-02-25T09:44:14-08:00] INFO: Run List expands to [iis_demo, hosts,  
users]  
[2014-02-25T09:44:14-08:00] INFO: Starting Chef Run for node1  
[2014-02-25T09:44:14-08:00] INFO: Running start handlers  
[2014-02-25T09:44:14-08:00] INFO: Start handlers complete.  
resolving cookbooks for run list: ["iis_demo", "hosts", "users"]  
[2014-02-25T09:44:15-08:00] INFO: Loading cookbooks [datacenter, hosts,  
iis_demo, users]  
Synchronizing Cookbooks:  
  - iis_demo  
  - hosts
```

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

Remote@PS\> chef-client

```
[2014-02-25T09:44:18-08:00] INFO: directory[C:\inetpub\wwwroot\admin] created directory C:\inetpub\wwwroot\admin
  - create new directory C:\inetpub\wwwroot\admin
    * powershell_script[create app pool for admin] action run[2014-02-25T09:44:18-08:00] INFO: Processing
powershell_script[create app pool for admin] action run (iis_demo::default line 24)

Name          State      Applications
----          ----      -----
admin         Started

[2014-02-25T09:44:19-08:00] INFO: powershell_script[create app pool for admin] ran successfully

  - execute "powershell.exe" -NoLogo -NonInteractive -NoProfile -ExecutionPolicy RemoteSigned -InputFormat None -
File"C:/Users/chef/AppData/Local/Temp/2/chef-script20140225-2140-1dltsrt.ps1"
    * powershell_script[new website for admin] action run[2014-02-25T09:44:19-08:00] INFO: Processing
powershell_script[new website for admin] action run (iis_demo::default line 29)

Name          ID  State      Physical Path           Bindings
----          --  -----      -----           -----
admin         8718 Started   C:\inetpub\wwwroot\admin     http *:8000:
7409
7
[2014-02-25T09:44:20-08:00] INFO: powershell_script[new website for admin] ran successfully
```

We added the admin site to the existing set

Node Attributes that are hashes are merged

- The `iis_demo` cookbooks attribute file

```
Default["iis_demo"]["sites"]["clowns"] = { "port" => 80 }
Default["iis_demo"]["sites"]["bears"] = { "port" => 81 }
```

- While our role has...

```
default_attributes({
  "iis_demo" => {
    "sites" => {
      "admin" => {
        "port" => 8000
      }
    }
  }
})
```

And the results are merged together.

We will get to merge order in just a few more slides...

Exercise: Search for the IIS sites attribute on all nodes with the role webserver

```
PS\> knife search node 'role:webserver' -a iis_demo.sites
```

```
1 items found
```

```
node1:  
  iis_demo.sites:  
    admin:  
      port: 8000  
    bears:  
      port: 81  
    clowns:  
      port: 80
```

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({  
  "iis_demo" => {  
    "sites" => {  
      "admin" => {  
        "port" => 8000  
      },  
      "bears" => {  
        "port" => 8081  
      }  
    }  
  }  
)
```

SAVE FILE!

Exercise: Create the role

```
PS\> knife role from file webserver.rb
```

```
Updated Role webserver!
```

Exercise: Re-run the Chef Client

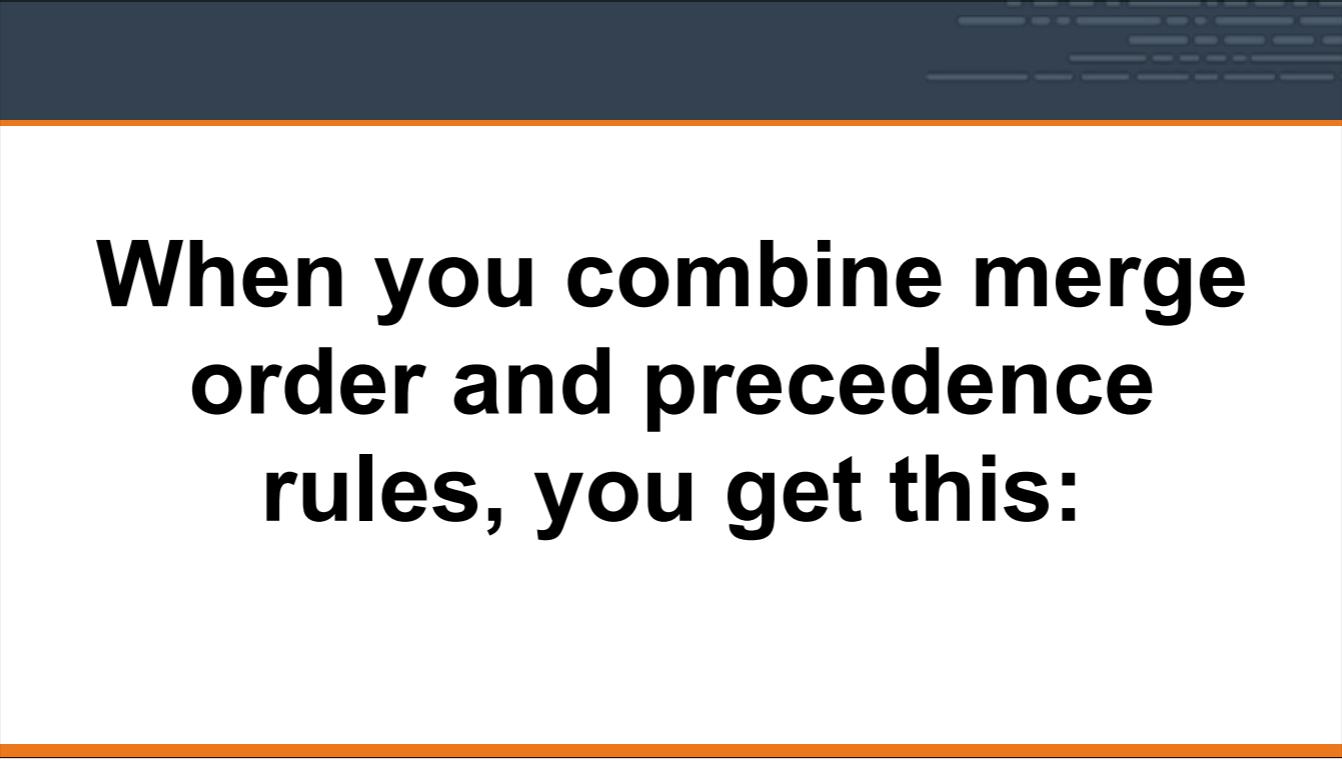
```
remote@PS\> chef-client
```

Exercise: Search for the apache sites attribute on all nodes with the role webserver

```
PS\> knife search node 'role:webserver' -a iis_demo.sites
```

```
1 items found

node1:
  iis_demo.sites:
    admin:
      port: 8000
    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		

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

Drive home that people should use defaults!

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

Almost every org has this

Exercise: Create the base role

OPEN IN EDITOR: roles\base.rb

```
name "base"
description "Base Server Role"
run_list "recipe[hosts]", "recipe[users]"
```

SAVE FILE!

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

```
PS\> knife role from file base.rb
```

```
Updated Role base!
```

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[iis_demo]"
default_attributes({
  "iis_demo" => {
```

- Put `role[base]` at the front of the `run_list`

SAVE FILE!

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

```
PS\> knife role from file webserver.rb
```

```
Updated Role webserver!
```

Exercise: Re-run the Chef Client

```
remote@PS\> chef-client
```

```
...
[2014-02-25T10:05:36-08:00] INFO: Run List is [role[webserver], recipe[hosts],
recipe[users]]
[2014-02-25T10:05:36-08:00] INFO: Run List expands to [hosts, users, iis_demo]
[2014-02-25T10:05:36-08:00] INFO: Starting Chef Run for node1
[2014-02-25T10:05:36-08:00] INFO: Running start handlers
[2014-02-25T10:05:36-08:00] INFO: Start handlers complete.
resolving cookbooks for run list: ["hosts", "users", "iis_demo"]
[2014-02-25T10:05:37-08:00] INFO: Loading cookbooks [datacenter, hosts, iis_demo,
users]
Synchronizing Cookbooks:
- hosts
- datacenter
- users
- iis_demo
Compiling Cookbooks...
...
```

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**

REPEAT THESE POINTS OFTEN -- drive this home

Exercise: Set the run list to just role[webserver]

- Remove all the entries in the run list other than `role[webserver]`, then **save** and **close**.

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.

Questions

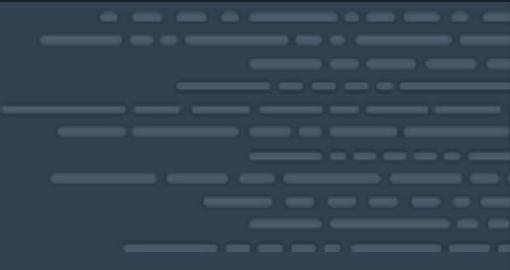
- What is a Role?
- What makes for a “good” role?
- How do you search for roles with a given recipe in their run list?
- How many times will Chef execute a recipe in the same run?

-) layer of abstraction that allows you to encapsulate the run lists and attributes required for a server to "be" what you think it is

-) Allowing you to configure multiple nodes identically with one definition

-) \$ knife search role "run_list:recipe\[apache\]"

-) 1



Environments

Cookbook Version Constraints, Override Attributes, and Per Environment Run Lists

v2.1.1_WIN

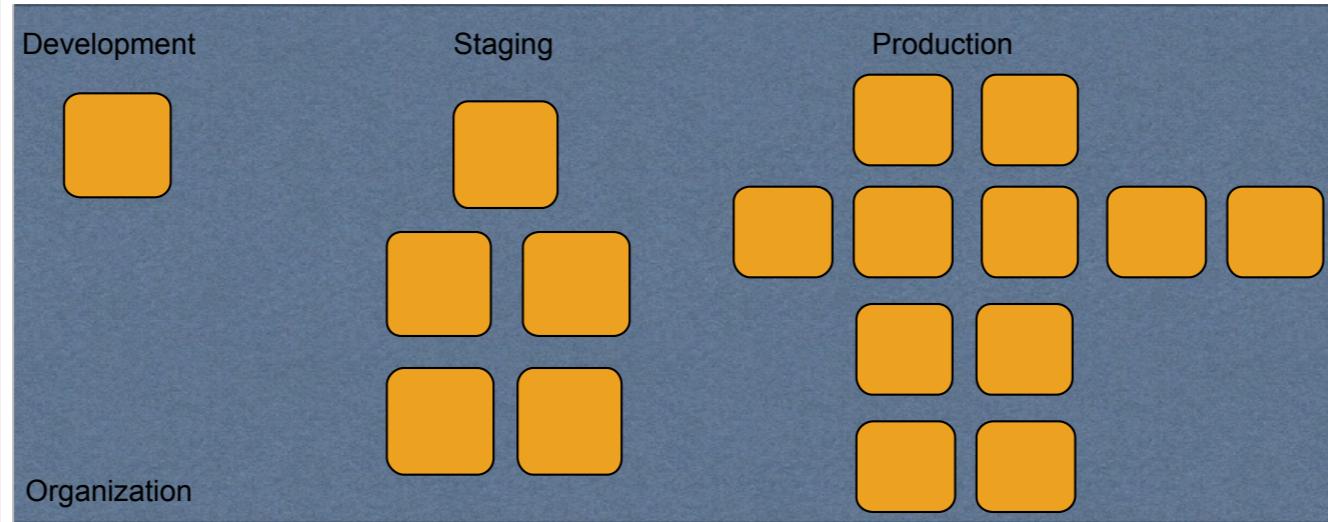


Lesson Objectives

- 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

Note: when this module is finished, we will have learned every core primitive

Environments



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

- We cannot share cookbooks between organizations
- 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

Cookbooks can be shared within environments, as opposed to organizations. (Organizations *can* share cookbooks through work arounds)

Exercise:

Use knife to show the available cookbook versions

```
PS\> knife cookbook show iis_demo
```

```
iis_demo      0.2.0      0.1.0
```

Remember the version bump from earlier? This was why.

Exercise: List the current environments

```
PS\> knife environment list
```

```
_default
```

- The `_default` environment is read-only, and sets no policy at all

Make an environments directory

```
PS\> mkdir environments
```

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d----	26-Feb-14 9:36 AM		environments

Another manual directory creation - like an animal

Exercise: Create a dev environment

OPEN IN EDITOR: environments\dev.rb

```
name "dev"
description "For developers!"
cookbook "iis_demo", "= 0.2.0"
```

SAVE FILE!

- Environments have **names**
- Environments have a **description**
- Environments *can* have one or more **cookbook** constraints

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.chef.io/chef/essentials_cookbook_versions.html

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`

Exercise: Create the dev environment

```
PS\> knife environment from file dev.rb
```

```
Updated Environment dev
```

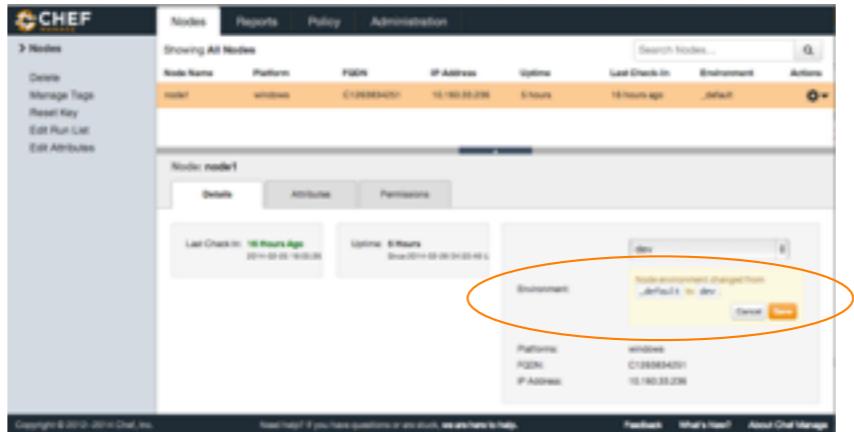
Exercise: Show your Chef dev environment

```
PS\> knife environment show dev
```

```
chef_type:          environment
cookbook_versions:
  iis_demo: = 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'



Note: Next slide has the knife command via knife environment

Click the Node tab then select node 'node1'

Might look at knife flip plugin – <https://github.com/jonlives/knife-flip>

Exercise: Change your node's environment to "dev"

```
PS:\> knife node environment set node1 dev
```

```
node1:  
  chef_environment: dev
```

487

This is new in 11.14

Exercise: Re-run the Chef Client

```
Remote@PS\> chef-client
```

```
[2014-02-26T01:46:16-08:00] INFO: Chef Run complete in  
7.988069 seconds
```

```
Running handlers:
```

```
[2014-02-26T01:46:16-08:00] INFO: Running report handlers  
Running handlers complete  
[2014-02-26T01:46:16-08:00] INFO: Report handlers complete
```

We removed PCI data, so we did modify the MOTD

Exercise: Create a production environment

OPEN IN EDITOR: environments\production.rb

- Make sure the iis_demo cookbook is set to version 0.1.0
- Set an override attribute for being in china for our DC environment no matter what.

```
name "production"
description "For Prods!"
cookbook "iis_demo", "= 0.1.0"
override_attributes({
  "datacenter" => {
    "location" => "china"
  }
})
```

SAVE FILE!

Exercise: Create the production environment

```
PS\> knife environment from file production.rb
```

```
Updated Environment production
```

Exercise: Change your node's environment to "production"

```
PS:\> knife node environment set node1 production
```

```
node1:  
  chef_environment: production
```

Exercise: Re-run the Chef Client

Remote@PS\> chef-client

```
* template[C:\Windows\System32\drivers\etc\hosts] action create[2014-02-26T01:57:44-08:00]
INFO: Processing template[C:\Windows\System32\drivers\etc\hosts] action create
(hosts::default line 9)
[2014-02-26T01:57:44-08:00] INFO: template[C:\Windows\System32\drivers\etc\hosts] backed up
to c:/chef/backup/Windows\System32\drivers\etc\hosts.chef-20140226015744.972901
[2014-02-26T01:57:44-08:00] INFO: template[C:\Windows\System32\drivers\etc\hosts] updated
file contents C:\Windows\System32\drivers\etc\hosts

- update content in file C:\Windows\System32\drivers\etc\hosts from 1f8bdf to 94a261
  --- C:\Windows\System32\drivers\etc\hosts      2014-02-25 03:26:02.000000000 -0800
  +++ C:/Users/chef/AppData/Local/Temp/2/chef-rendered-template20140226-3540-w4vntg
2014-02-26 01:57:44.000000000 -0800
@@ -1,3 +1,4 @@
 #This file is managed by server at C1263834251
+ 0.0.0.0    nytimes.com
```

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 `iis_demo` cookbook
- While the recipe applied fine, investigating the system will reveal IIS 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.

Exercise: Create a production environment

OPEN IN EDITOR: environments\production.rb

- Make sure the iis_demo cookbook is set to version 0.2.0

No example!

Exercise: Upload the updated production environment

```
PS\> knife environment from file production.rb
```

```
Updated Environment production
```

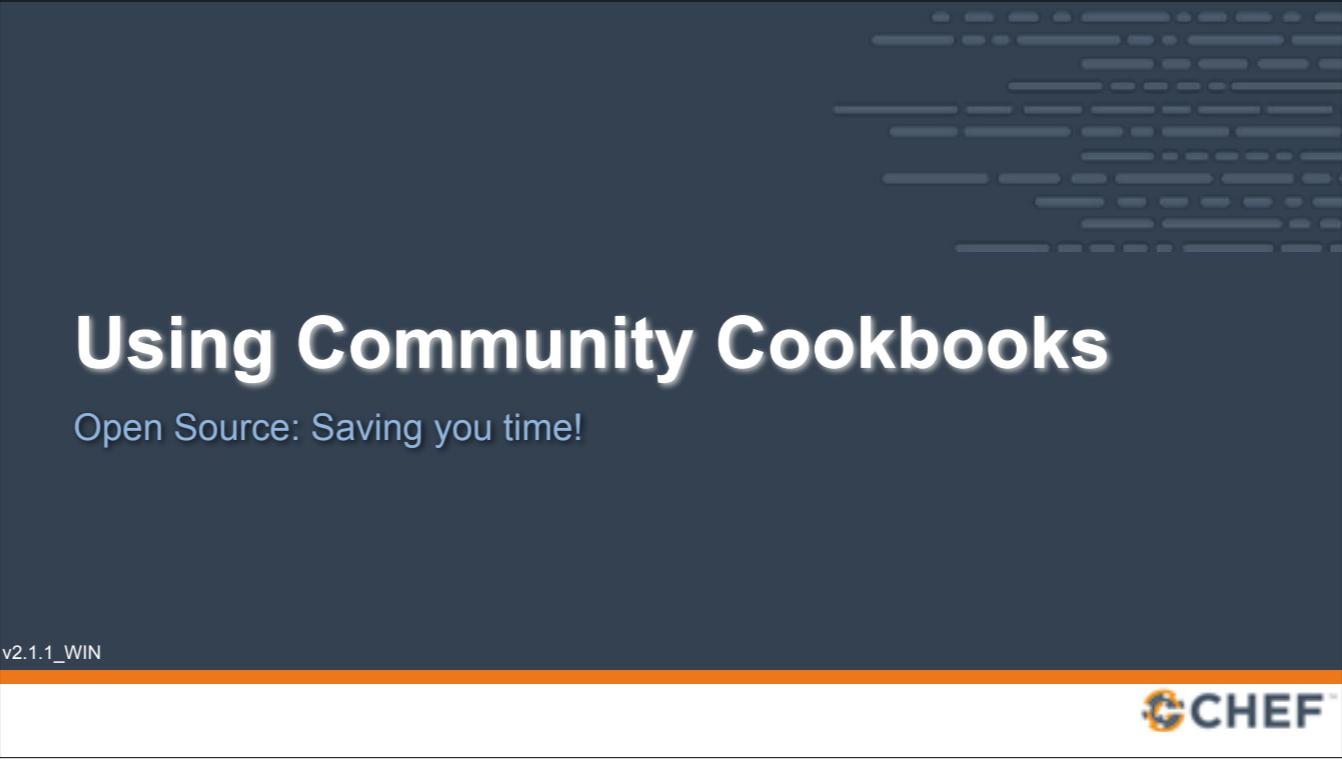
Questions?

- What is an Environment?
- How is it different from an Organization?
- What is a cookbook version constraint?
- Which cookbook constraint should we most likely be using?
- What kind of node attributes do you typically set from an Environment?
- What is a per-environment run list?

-) Abstraction that allows you to align objects based on common attributes or 'real' associations
-) Can share information across environments.
-) API strings, database, etc.

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!

v2.1.1_WIN



note: screenshots in this section will always be outdated (supermarket is getting a lot of love)

Lesson Objectives

- After completing the lesson, you will be able to
 - Find, preview and download cookbooks from the Chef Supermarket site
 - Use knife to work with the Supermarket API
 - Download, extract, examine and implement cookbooks from the Supermarket 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 Chef.
- Think of it like RubyGems.org, CPAN.org, or other focused plugin-style distribution sites.

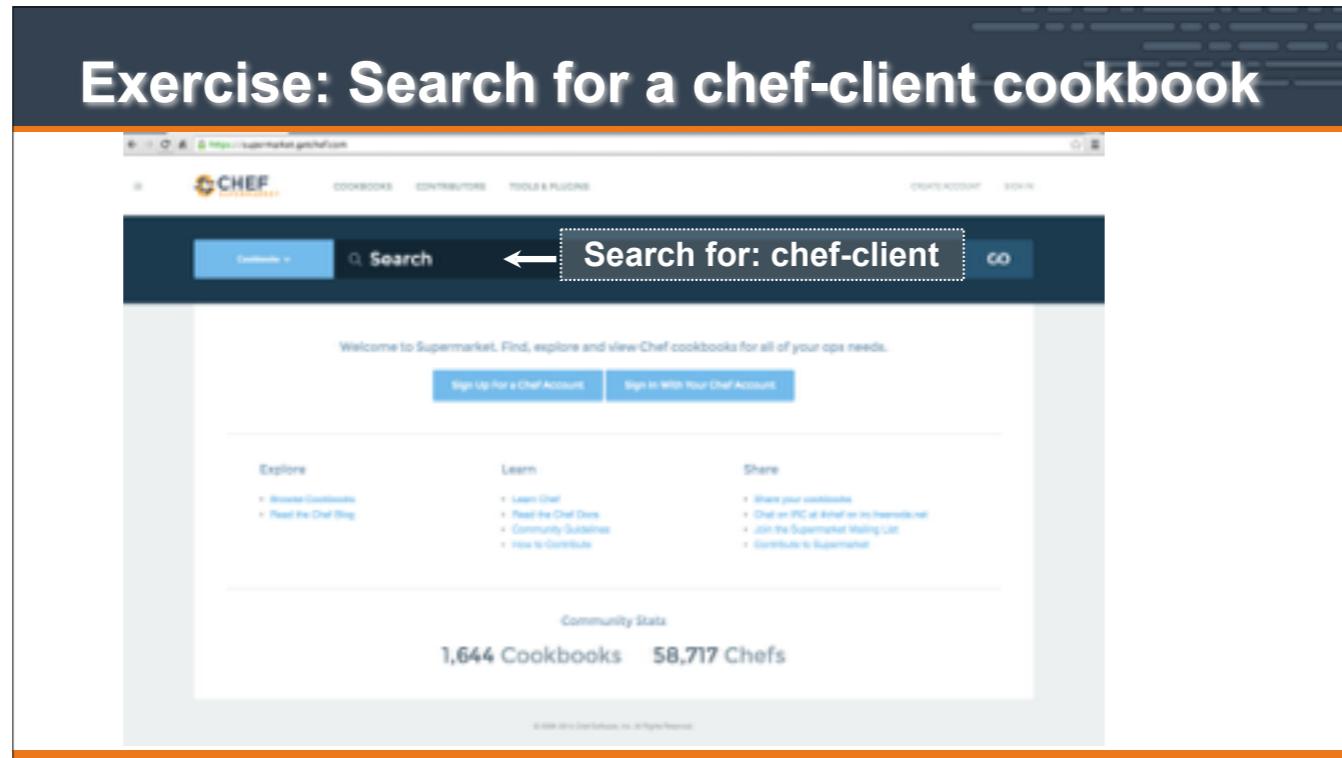
Exercise: Find and preview cookbooks on the site

The screenshot shows the homepage of the Supermarket Chef website. At the top, there's a dark header bar with the title "Exercise: Find and preview cookbooks on the site". Below the header is a navigation bar with links for "COOKBOOKS", "CONTRIBUTORS", "TOOLS & PLUGINS", "CREATE ACCOUNT", and "SIGN IN". A search bar with a placeholder "Search" and a "GO" button is prominently displayed. The main content area features a welcome message: "Welcome to Supermarket. Find, explore and view Chef cookbooks for all of your ops needs." Below this are three sections: "Explore" (with links to "Browse Cookbooks" and "Read the Chef Blog"), "Learn" (with links to "Learn Chef", "Read the Chef Docs", "Community Guidelines", and "How to Contribute"), and "Share" (with links to "Share your cookbooks", "Chef on IRC at #chef on irc.freenode.net", "Join the Supermarket Mailing List", and "Contribute to Supermarket"). At the bottom, there's a "Community Stats" section showing "1,644 Cookbooks" and "58,717 Chefs". The footer contains a small copyright notice: "© 2008-2011 Chef Software, Inc. All Rights Reserved".

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 cookbooks main page.

Perform a search using the search box. Simply type in 'chef-client' and hit enter.

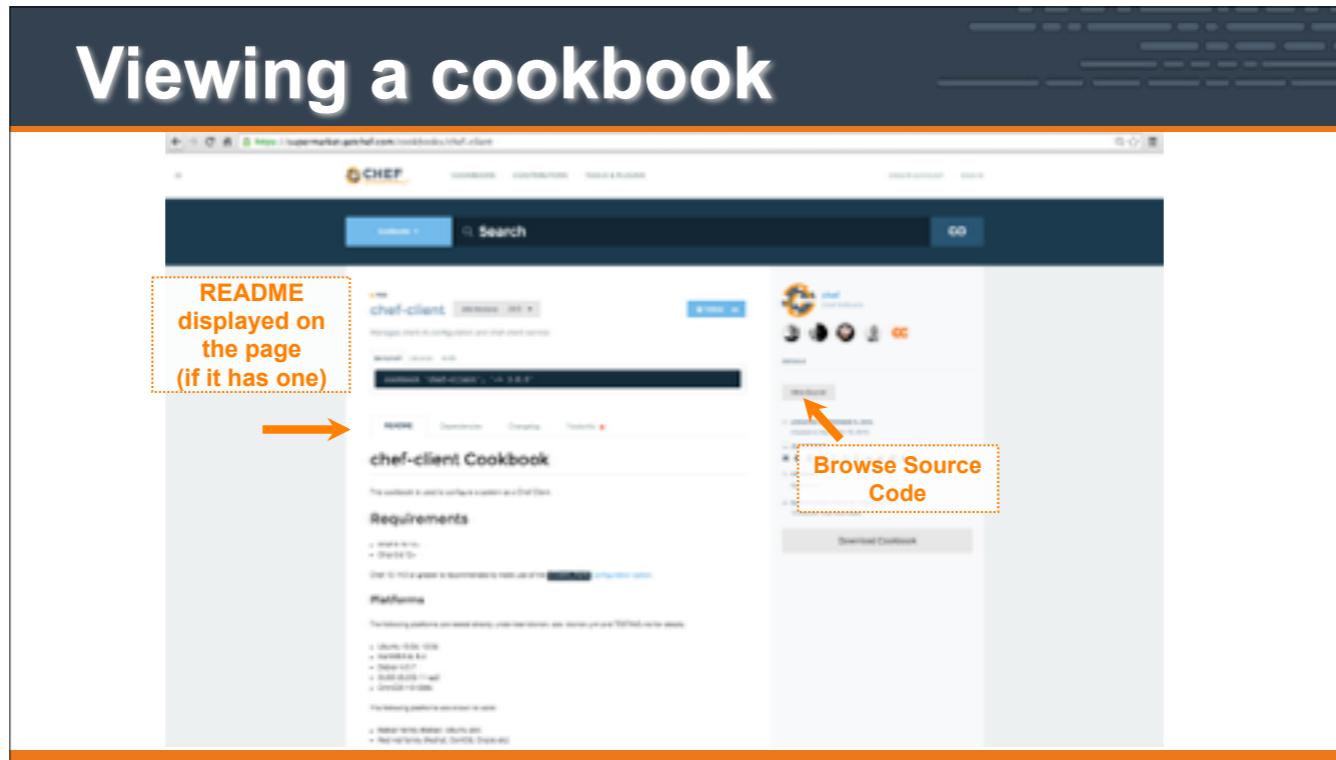
Search Results...

The screenshot shows a web browser displaying the Chef Market search results for the query "chef-client". The search bar at the top contains "chef-client". Below the search bar, there is a header with the Chef logo and navigation links for COOKBOOKS, CONTRIBUTE, TOOLS & PLUGINS, CREATE ACCOUNT, and SIGN IN.

The main content area displays a list of cookbooks under the heading "26 Cookbooks". The first result is "chef-client" (0.1.0), which has a description: "Manage client's configuration and chef-client service". A screenshot of the cookbook page shows a line of code: `loadpath 'chef-client'`. An orange callout box with the text "We're probably looking for this one" points to this line of code. The second result is "chef-client_syslog" (0.1.0), with a description: "chef-client log to syslog". A screenshot of its page shows a line of code: `loadpath 'chef-client_syslog'`.

Here's the results, there are other cookbooks, for purposes of this class we want the first one.

Viewing a cookbook



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

v2.1.1_WIN



Exercise: Use knife to search the community site

```
PS\> 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: chef
  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: chef
  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

chef-client_syslog:
  ....
```

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

```
PS\> knife cookbook site show chef-client
```

```
average_rating:
category:          Other
created_at:        2010-12-16T23:00:45.000Z
deprecated:        false
description:       Manages client.rb configuration and chef-client service
external_url:      http://github.com/opscode-cookbooks/chef-client
foodcritic_failure: true
latest_version:    http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/3.9.0
maintainer:         chef
metrics:
downloads:
  total: 24629844
  versions:
    0.99.0: 562401
    ...
    3.9.0: 132395
```

Exercise: Download the chef-client cookbook

```
PS\> knife cookbook site download chef-client
```

```
Downloading chef-client from the cookbooks site at  
version 3.9.0 to /Users/YOU/chef-repo/chef-  
client-3.9.0.tar.gz
```

```
Cookbook saved: /Users/YOU/chef-repo/chef-  
client-3.9.0.tar.gz
```

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

```
PS\> tar -zvxf chef-client-3.9.0.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
x chef-client/recipes/service.rb
x chef-client/attributes/default.rb
```

Note: Windows users cannot use *, they must pass in the full file name
(they can use tab completion in PS)

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.

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!

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 Administrator!

- 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!

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: cookbooks\chef-client\recipes\delete_validation.rb

```
class ::Chef::Recipe
  include ::Opscode::ChefClient::Helpers
end

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!

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: roles\base.rb

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

SAVE FILE!

- Add the **delete_validation** recipe

Put the delete validation recipe first in the node's run list.

Best Practice: Delete the validation certificate when it isn't required

- Once Chef enters the actual run, synchronizing cookbooks, it has register 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: 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
```

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

```
PS\> knife cookbook upload chef-client
```

```
Uploading ch... [3/3] [100%]
ERROR: Cookbook 'chef-client' depends on cookbook 'cron' version '>= 1.2.0',
ERROR: which is not currently available for upload and can't be found on the
server.
```

FAIL!

Need to grab the cron cookbook too

Exercise: Download the cron cookbook

```
PS\> knife cookbook site download cron
```

```
Downloading cron from the cookbooks site at version  
1.2.8 to /Users/YOU/SOMEFOLDER/cron-1.2.8.tar.gz  
Cookbook saved: /Users/YOU/chef-repo/  
cron-1.2.8.tar.gz
```

Exercise: Extract cron cookbook tarball

```
PS\> tar -zxvf cron-1.3.0.tar.gz -C cookbooks\
```

```
x cron/
x cron/CHANGELOG.md
x cron/CONTRIBUTING
x cron/Gemfile
x cron/LICENSE
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/resources/
x cron/resources/d.rb
x cron/templates/
x cron/templates/default/
x cron/templates/default/cron.d.erb
```

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.

```
tar -zxvf cron-1.2.8.tar.gz -C cookbooks
```

Exercise: Upload the cron cookbook

```
PS\> knife cookbook upload cron
```

```
Uploading cron [1.2.8]  
Uploaded 1 cookbook.
```

Exercise: Upload the chef-client cookbook

```
PS\> knife cookbook upload chef-client
```

```
Uploading chef-client [30%]
ERROR: Cookbook 'chef-client' depends on cookbook 'curl' version '1.2.0',
ERROR: which is not currently available for upload and can't be found on the
server.
```

FAIL!

Exercise: Download the logrotate cookbook

```
PS\> knife cookbook site download logrotate
```

```
Downloading logrotate from the cookbooks site at
version 1.5.0 to C:/Users/somefolder/
logrotate-1.5.0.tar.gz
Cookbook saved: C:/Users/somefolder/
logrotate-1.5.0.tar.gz
```

Exercise: Extract logrotate cookbook tarball

```
PS\> tar -zxvf logrotate-1.5.0.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`
```

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

```
PS\> knife cookbook upload logrotate
```

```
Uploading logrotate [1.3.0]
Uploaded 1 cookbook.
```

Exercise: ...windows

```
PS\> knife ...
```

```
Downloading ...
Extracting ...
Uploading ...
```

knife cookbook site download windows

tar -zxvf windows-1.34.8.tar.gz -C cookbooks\

knife cookbook upload windows

Exercise: Upload the chef-client cookbook

```
PS\> knife cookbook upload chef-client
```



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[hosts]", "recipe[users]"
```

SAVE FILE!

Exercise: upload the base role

```
PS\> knife role from file base.rb
```

```
Updated Role base!
```

Exercise: Re-run the Chef Client

```
remote@PS\> chef-client
```

```
Starting Chef Client, version 11.10.4
[2014-02-26T04:08:21-08:00] INFO: *** Chef 11.10.4 ***
[2014-02-26T04:08:21-08:00] INFO: Chef-client pid: 3412
[2014-02-26T04:08:36-08:00] INFO: Run List is [role[webserver], recipe[hosts], recipe[users]]
[2014-02-26T04:08:36-08:00] INFO: Run List expands to [chef-client::delete_validation, chef-client, hosts, users, iis_de
mo]
[2014-02-26T04:08:36-08:00] INFO: Starting Chef Run for node1
[2014-02-26T04:08:36-08:00] INFO: Running start handlers
[2014-02-26T04:08:36-08:00] INFO: Start handlers complete.
resolving cookbooks for run list: ["chef-client::delete_validation", "chef-client", "hosts", "users", "iis_demo"]
[2014-02-26T04:08:37-08:00] INFO: Loading cookbooks [chef-client, cron, datacenter, hosts, iis_demo, logrotate, users]
Synchronizing Cookbooks:
[2014-02-26T04:08:38-08:00] INFO: Storing updated cookbooks/chef-client/recipes/arch_service.rb in the cache. ipes/config.rb in the cache.
...
...
```

Examine chef-client output

```
...
Recipe: chef-client::delete_validation
  * file[c:/chef/validation.pem] action delete[2014-02-26T04:08:55-08:00] INFO:
Processing file[c:/chef/validation.pem]
action delete (chef-client::delete_validation line 25)
[2014-02-26T04:08:55-08:00] INFO: file[c:/chef/validation.pem] deleted file at c:/chef/validation.pem

  - delete file c:/chef/validation.pem
...
...
```

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)

Examine chef-client output

```
...
Recipe: chef-client::windows_service
...
  * execute[register-chef-service] action run[2014-02-26T04:08:55-08:00] INFO:
Processing execute[register-chef-service]
  action run (chef-client::windows_service line 34)
Service 'chef-client' has successfully been installed.
[2014-02-26T04:08:59-08:00] INFO: execute[register-chef-service] ran successfully

    - execute chef-service-manager -a install
      * service[chef-client] action enable[2014-02-26T04:08:59-08:00] INFO: Processing
service[chef-client] action enable (chef-client::windows_service line 39)
      (up to date)
      * service[chef-client] action start[2014-02-26T04:08:59-08:00] INFO: Processing
service[chef-client] action start (chef-client::windows_service line 39)
[2014-02-26T04:09:07-08:00] INFO: service[chef-client] started

    - start service service[chef-client]
...
...
```

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

```
Remote@PS\> get-service chef*
```

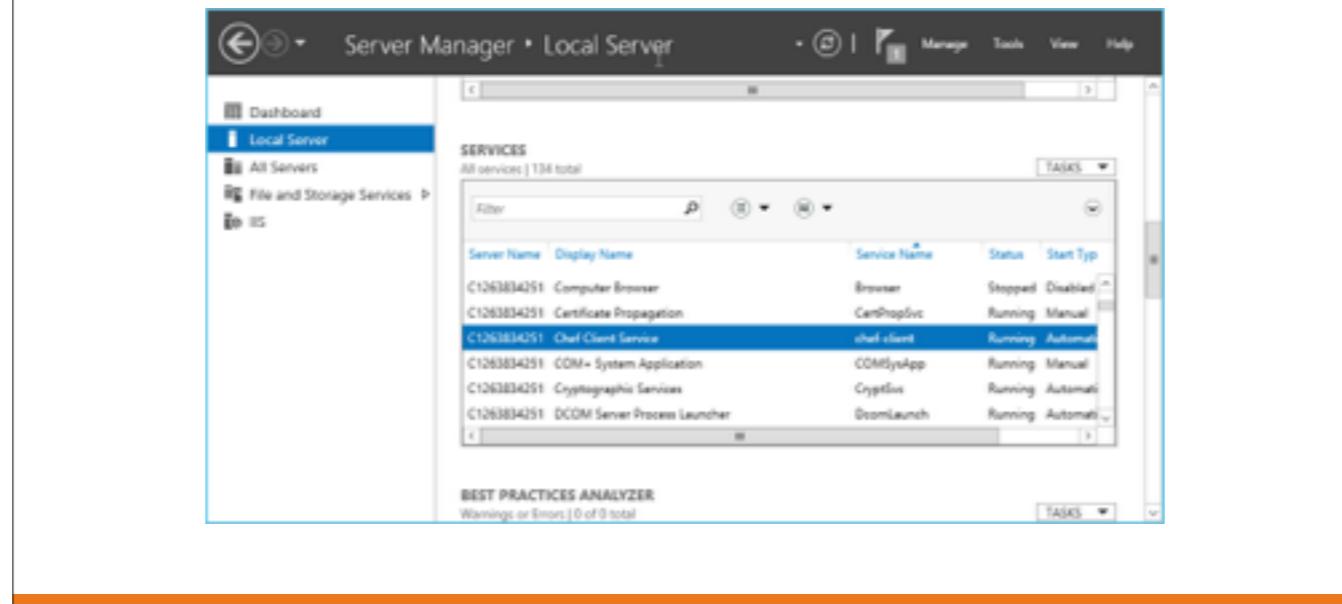
Status	Name	DisplayName
-----	----	-----
Running	chef-client	Chef Client Service

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



Convergent infrastructure

- Our node is now running chef-client as a service, 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.

Get nerdy about convergence!

chef-client daemon can be signaled with USR1 to wake it up

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?

-) supermarket.chef.io
-) GUI and knife cookbook site download <name>
-) Readme, followed by the entire cookbook/recipe
-) I do
-) I do



WORKING WITH CHEF, THE COMPANY

v2.1.1_WIN



Chef Status

- Status for Chef related systems can be found by browsing to
 - <http://status.chef.io>, or
 - https://twitter.com/opscode_status

542

<<Review Comment: This slide added for this Trello card <https://trello.com/c/OewKCBAz>

<<WIP: At time of writing status.getchef.com wasn't config'd yet

How do I interact with Chef support?

- There are two ways to raise a ticket with Chef Support
 - Via the Web UI at <http://www.getchef.com/support/>
 - By sending an email to support@chef.io
- The Web UI allows you to submit any severity level ticket, however emailed tickets default to 'severity 3'
- The Web UI allows you to view previously submitted tickets for an org

543

<<Review Comment: This slide added for this Trello card <https://trello.com/c/OewKCBAz>

The web interface is the best method of submitting tickets to the Support team. From this interface you may submit any severity level ticket, as well as view previously submitted tickets. Previous tickets are shared for all members of the same organization (see below for more information). This interface is the only method of submitting a Severity 1 ticket; after submission you may use email as the method of updating the ticket.

To re-open a closed ticket

- To re-open a closed ticket either
 - Click on the closed ticket in the web interface, and reply in the box provided, or
 - Reply to the email thread
- This will restart the ticket and move it back into the active queue

544

<<Review Comment: This slide added for this Trello card <https://trello.com/c/OewKCBAz>

Dealing with Support

- When an org signs up for a support contract, the email domain of the primary user is used as a key, and anyone with an email in that domain can submit tickets
- E.g. user1@domain.com starts a support contract for an org, then user2@domain.com and user3@domain.com can also submit tickets on behalf of "domain.com" for that org
- Any user email address be added explicitly to the Support Tool for your org by emailing sales@chef.io or submitting a ticket

545

<<Review Comment: This slide added for this Trello card <https://trello.com/c/OewKCBAz>

New releases

- Release Notes can be found here
 - <http://docs.chef.io/>

546

<<Review Comment: This slide added for this Trello card https://trello.com/c/OewKCBAZ

Further Resources

v2.1.1_WIN



Chef Fundamentals - Q&A Forum

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

548

Points to <https://groups.google.com/forum/#!forum/chef-fundamentals-workshop>

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

Further Resources

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

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





CHEF PRESENTS
#ChefConf 2015

March 31 – April 2 Santa Clara, CA
Santa Clara Convention Center



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

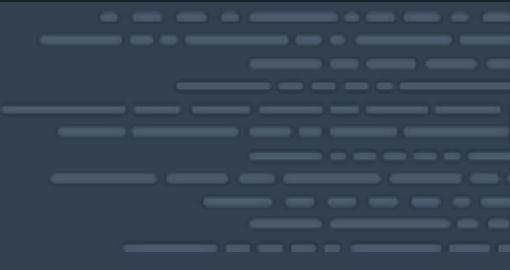
The definitive assembly for the tens of thousands-strong Chef community will convene IT leaders representing some of the most dynamic companies and thinking in the world.

Local Meetup Groups

- Fill in local meetup groups here
<http://opscode.meetup.com>
- <https://wiki.opscode.com/display/chef/Resources+for+Community+Organizers>

553

ref0038



Just enough Ruby for Chef

A quick & dirty crash course

v2.1.1_WIN



Objectives

- Know what irb is, and how to use it.
- Become familiar with:
 - Variable Assignment
 - Basic Arithmetic
 - Strings
 - Truthiness
 - Operators
 - Arrays
 - 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

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

```
remote@PS> C:\opscode\chef\embedded\bin\irb
```

```
irb(main):001:0>
```

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.

Exercise: Variable assignment

```
remote@PS> C:\opscode\chef\embedded\bin\irb
```

```
irb(main):001:0> x = "hello"
=> "hello"
irb(main):002:0> puts x
Hello
=> nil
```

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

Exercise: Arithmetic

```
irb(main):003:0> 1 + 2  
=> 3
```

Plus is addition – numbers are unquoted

Note the return code!

Note if strings: "1" + "2" = "12"

Exercise: Arithmetic

```
irb(main):004:0> 18 - 5  
=> 13
```

Subtraction works like you think it does

Exercise: Arithmetic

```
irb(main):005:0> 2 * 7  
=> 14
```

* is multiplication

Exercise: Arithmetic

```
irb(main):006:0> 5 / 2  
=> 2
```

5 / 2 is... 2?

Division on whole numbers means no decimal points

Exercise: Arithmetic

```
irb(main):007:0> 5 / 2.0  
=> 2.5
```

Division on floating point numbers means decimal points

Exercise: Arithmetic

```
irb(main):008:0> 5.class  
=> Fixnum  
irb(main):009:0> 5.0.class  
=> Float
```

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.

Exercise: Arithmetic

```
irb(main):010:0> 1 + (2 * 3)  
=> 7
```

You can group expressions with parenthesis

Exercise: Strings

```
irb(main):011:0> 'jungle'  
=> "jungle"
```

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.

Exercise: Strings

```
irb(main):012:0> 'it\'s alive'  
=> "it's alive"
```

You escape the quote delimiter with a forward slash

REPL gives double quoted version, so no escape character

Exercise: Strings

```
irb(main):013:0> "animal"  
=> "animal"
```

Strings in ruby can use double quotes

Exercise: Strings

```
irb(main):014:0> \"so easy\"  
=> "so easy"  
irb(main):015:0> puts \"so easy\"  
"so easy"  
=> nil
```

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

Exercise: Strings

```
irb(main):016:0> x = "pretty"
=> "pretty"
irb(main):017:0> "#{x} nice"
=> "pretty nice"
irb(main):018:0> '#{x} nice'
=> "\#{x} nice"
```

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

Exercise: 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
```

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

Exercise: 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
```

`==` tests for equality
`1` evaluates to true *when pressed* – but it is not true

Exercise: 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
```

Greater than, less than, greater than or equal to, less than or equal to

Exercise: Operators

```
irb(main):035:0> 5 <=> 5  
=> 0  
irb(main):036:0> 5 <=> 6  
=> -1  
irb(main):037:0> 5 <=> 4  
=> 1
```

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!

Exercise: 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"
```

They have methods just like everything else – first and last, for instance

Exercise: 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" ]
```

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

Exercise: Arrays

```
irb(main):046:0> x << "e"  
=> ["a", "b", "c", "d", "e"]  
irb(main):047:0> x  
=> ["a", "b", "c", "d", "e"]
```

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

Exercise: 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"]
```

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.

Exercise: 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"]
```

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!

Exercise: 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"}
```

A key value pair – like a dict in python, or a hash in perl

Exercise: 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"]
```

We can add items to the hash by putting the key in [], and using assignment on that key name

Exercise: Hashes

```
irb(main):060:0> h.values  
=> [ "Gary", "Gygax", 33 ]
```

We can get all the items in the hash with .values

Exercise: 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}
```

We can iterate over the hash with .each – here we print out the keys and values

We all love Regular Expressions



www.rubular.com

<http://xkcd.com/1171/>

Exercise: 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
```

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!

Exercise: 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
```

String substitution (ala sed)

Exercise: 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
```

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

Exercise: 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
```

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 ≥ 3 , use if and elsif. If more than 3, use case. Style choice.

Exercise: 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
```

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 **ddddddeeeeeeeeeeeepppppppp**

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.

Exercise: 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
```

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

Exercise: 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
```

Create a new instance of your class with ".new"
Use the accessors with .name = ""

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>