

Chef Fundamentals

training@opscode.com



Introductions





Instructor Introduction



Introduce yourselves

Course Objectives and Style



- Write cookbooks to automate common infrastructure tasks
- Understand Chef's architecture
- Be familiar with Chef's various tools
- Understand how to apply Chef's primitives to solve your problems

- 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

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

Training is really a discussion

- I'll post objectives at the beginning of a section - if everyone gets it already, we'll skip it!
- Ask questions when they come to you
- Ask for help when you need it

Agenda



- Overview of Chef
- Workstation Setup
- Test Node Setup
- Dissecting your first Chef run
- Introducing the Node object
- Writing an Apache cookbook
- Writing an MOTD cookbook
- Refactoring the Apache Cookbook
- Writing a Users Cookbook

- Roles
- Environments
- Using community cookbooks
- If we have time
 - Just enough Ruby for Chef
 - Building LWRPs
 - Shef
 - Working a use case

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

Overview of Chef

What is this thing again?



- Understand how Chef thinks about Infrastructure Automation
- Understand the following concepts:
 - Node
 - Resource
 - Recipe
 - Cookbook
 - Run List
 - Roles
 - Search

Applications



<http://www.flickr.com/photos/steffenz/337700069/>
<http://www.flickr.com/photos/kky/704056791/>



Infrastructure

<http://www.flickr.com/photos/sbh/462754460/>

Collection of Resources

- Nodes
- Networking
- Files
- Directories
- Symlinks
- Mounts
- Routes
- Users
- Groups
- Tasks
- Packages
- Software
- Services
- Configurations
- Stuff

<http://www.flickr.com/photos/stevekeys/3123167585/>



Acting in concert



RULE THE

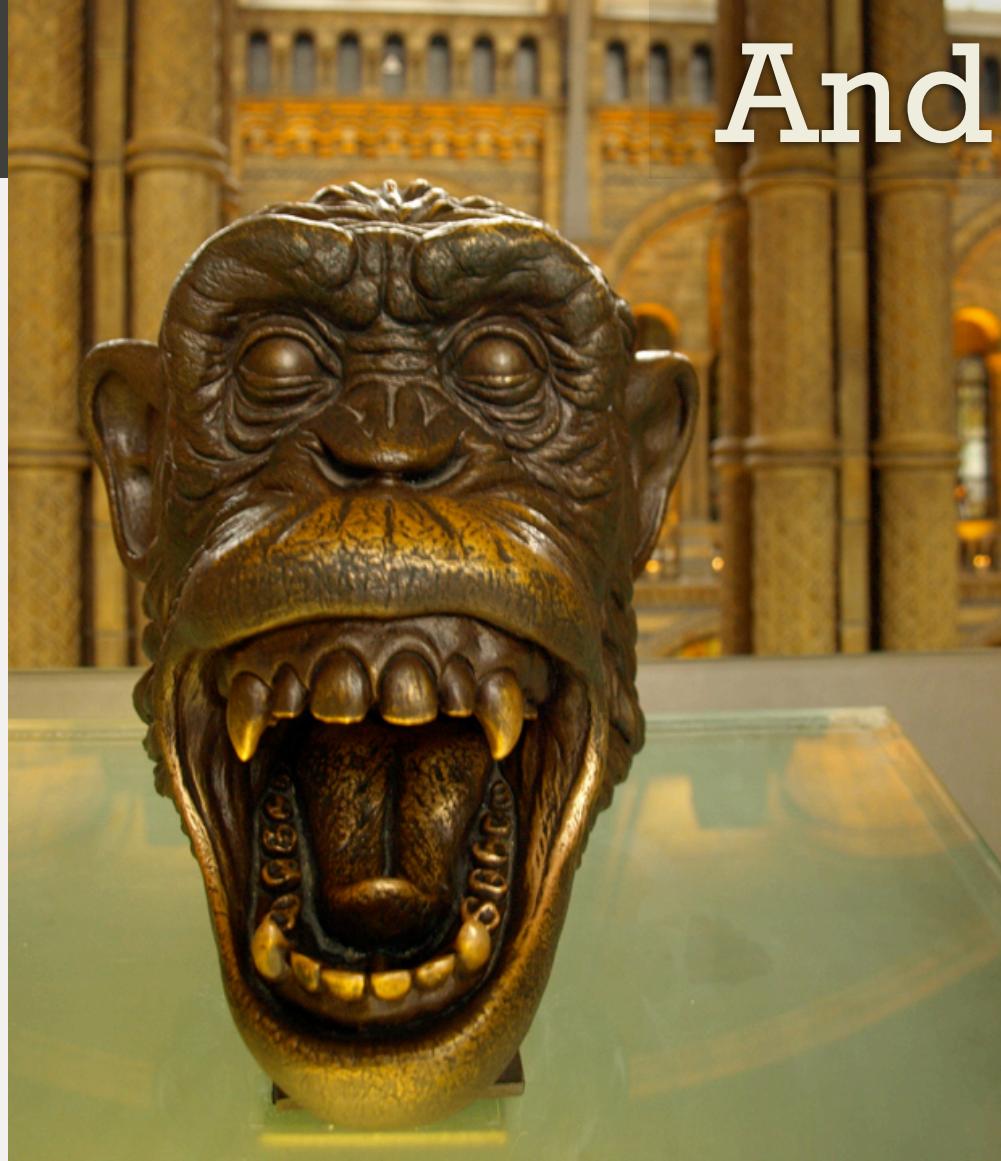
<http://www.flickr.com/photos/glowjangles/4081048126/>



To provide a Service



<http://www.flickr.com/photos/28309157@N08/3743455858/>

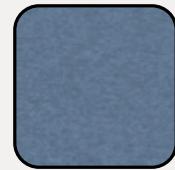


And it evolves

<http://www.flickr.com/photos/16339684@N00/2681435235/>



See Node



Application

RULE THE CLOUD

See Nodes



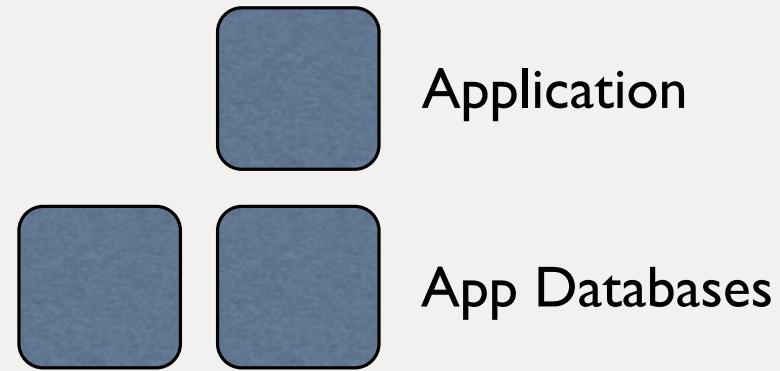
Application



Application Database

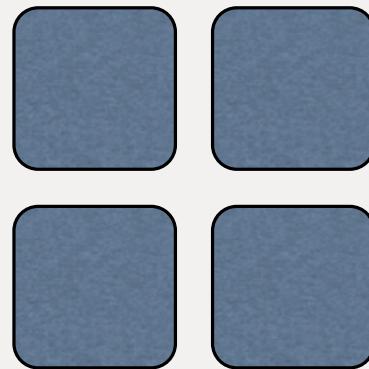


See Nodes Grow



RULE THE CLOUD

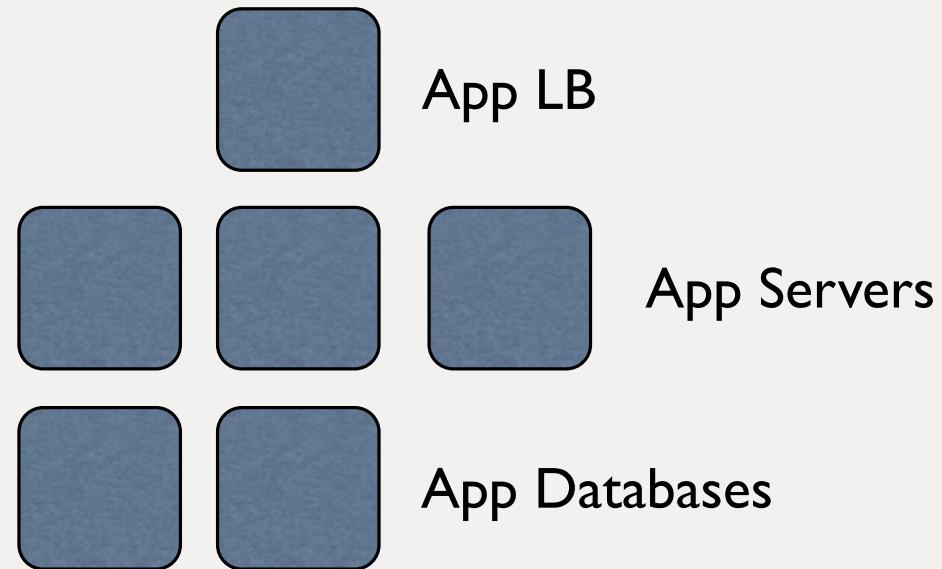
See Nodes Grow



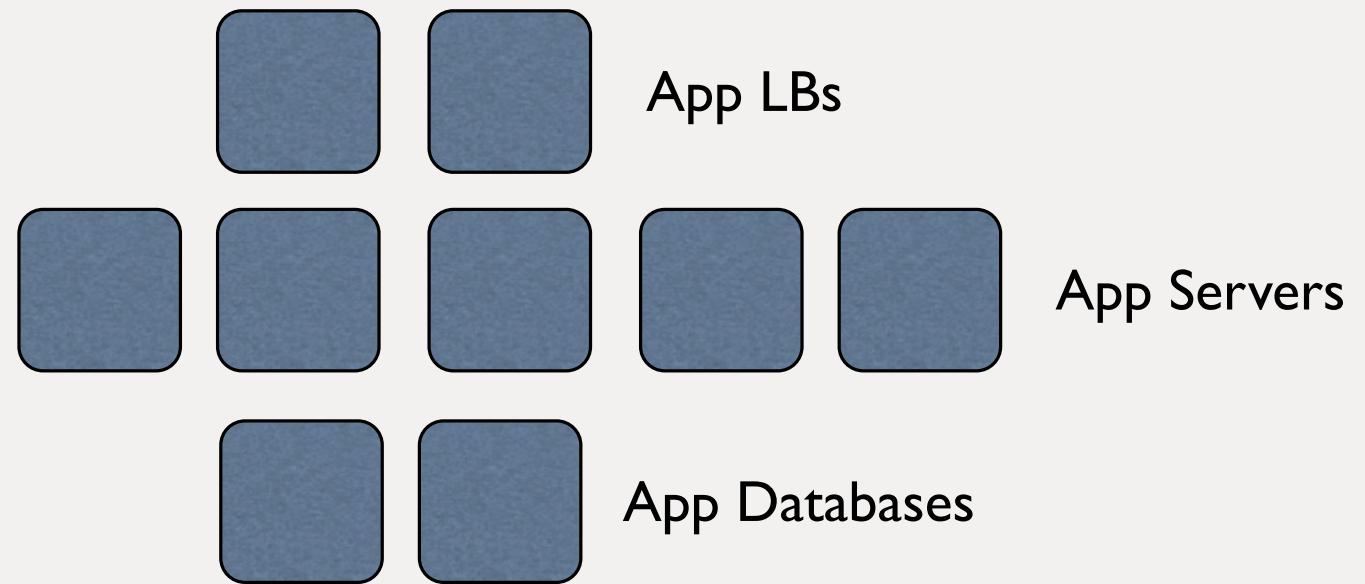
App Servers

App Databases

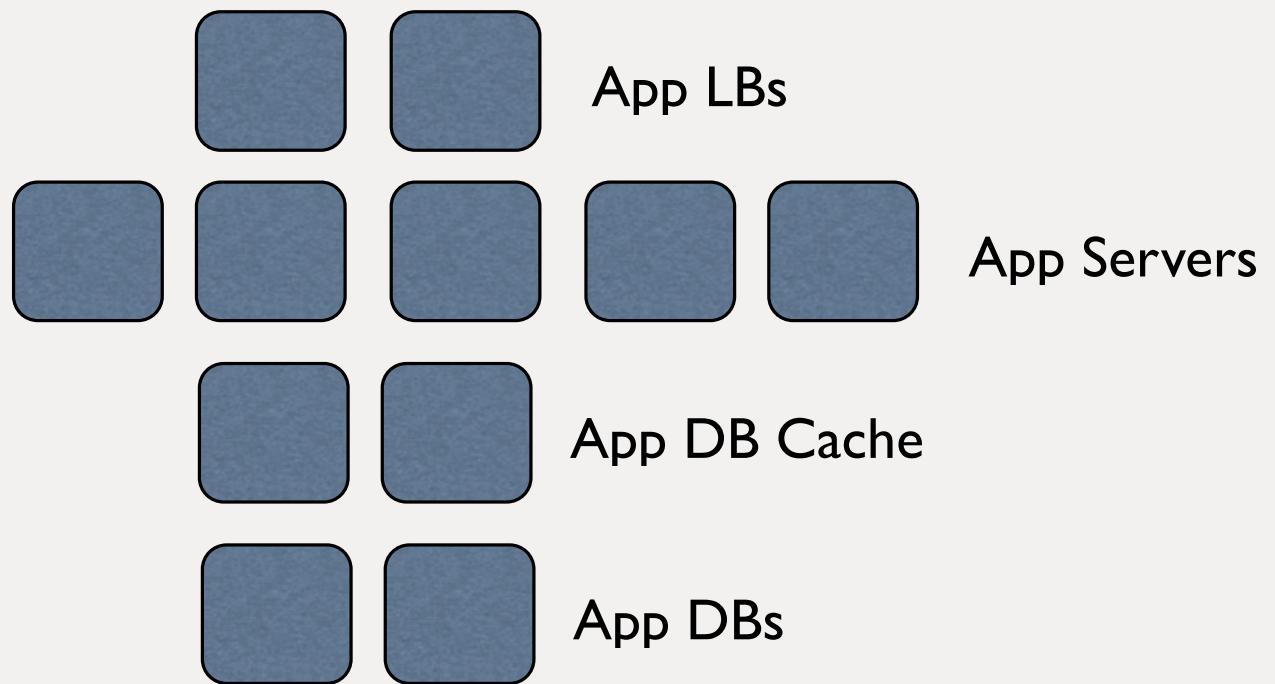
See Nodes Grow



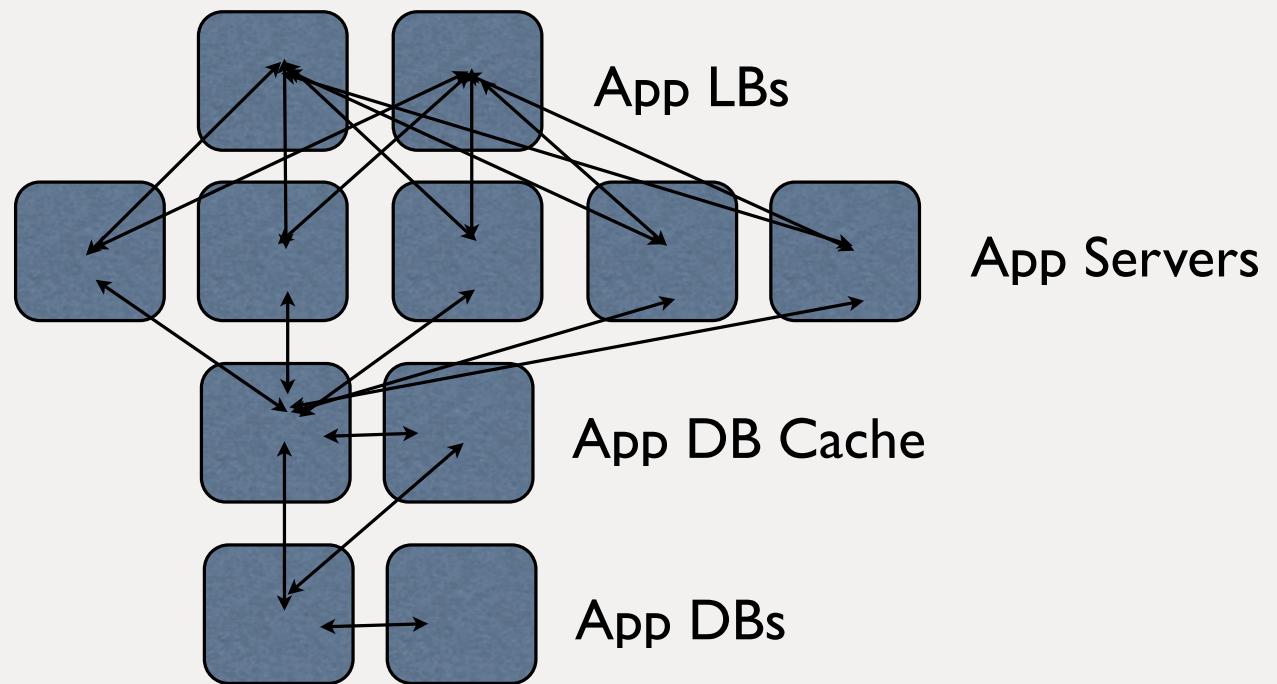
See Nodes Grow



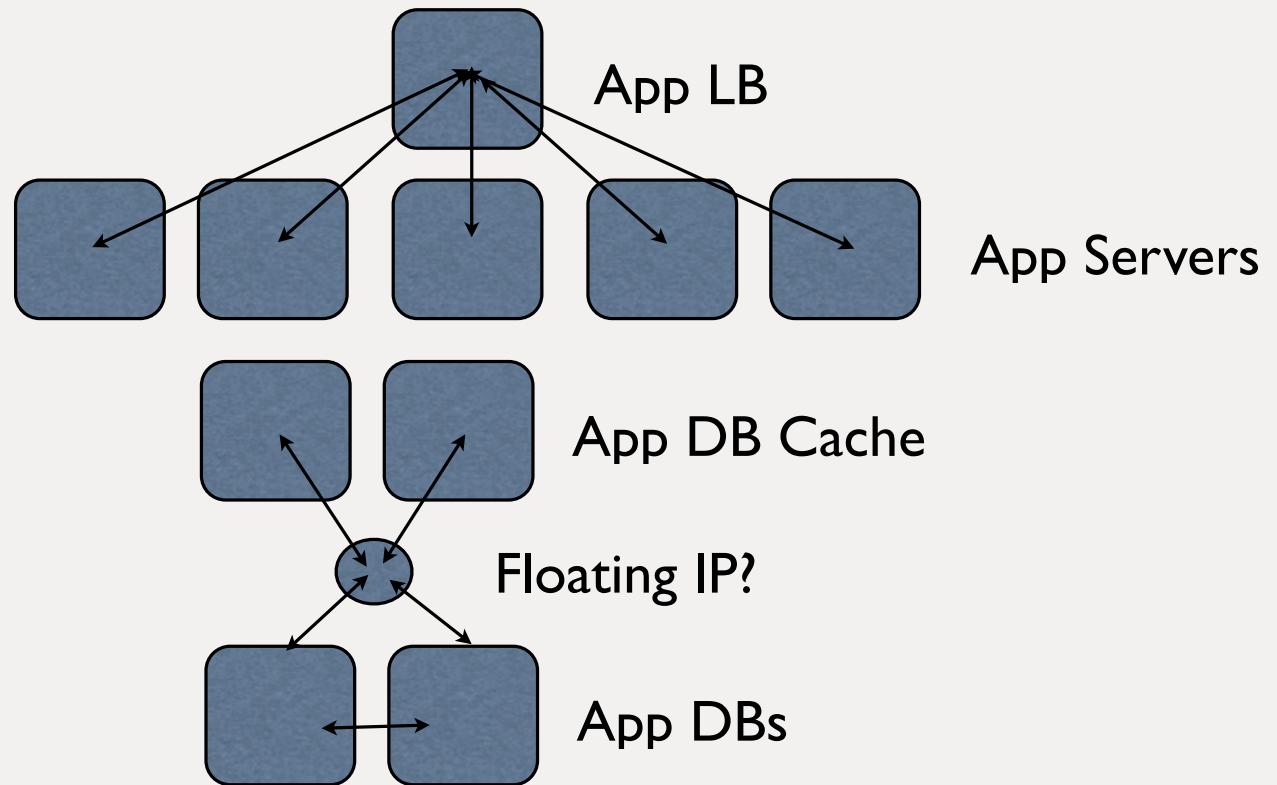
See Nodes Grow



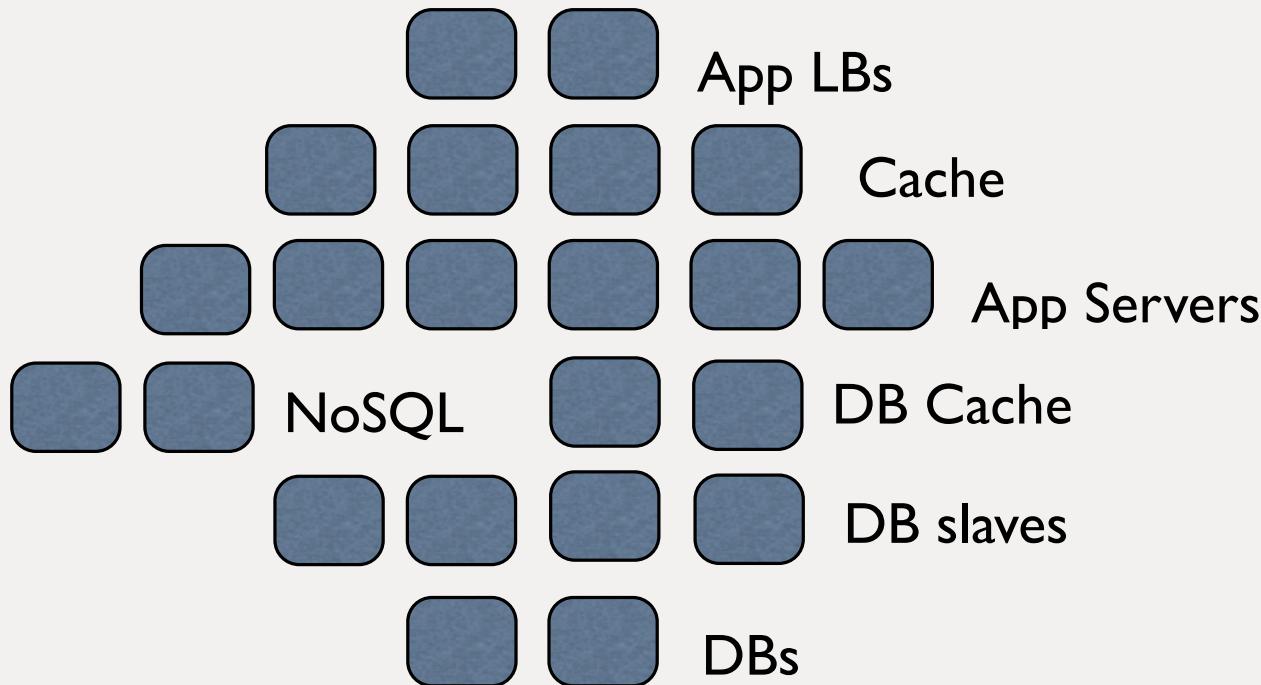
Stitched together with configs



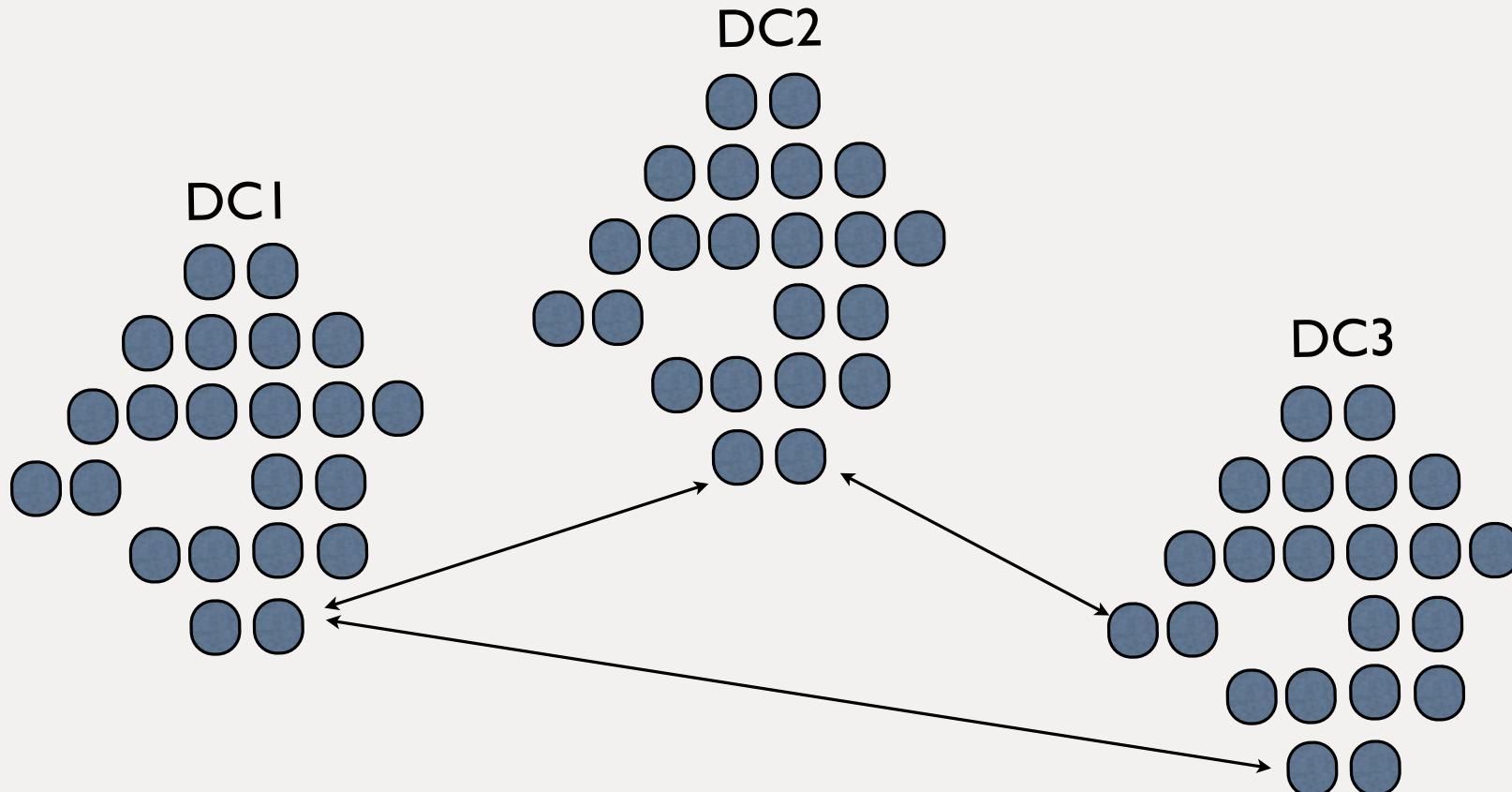
Your Infrastructure is a snow flake

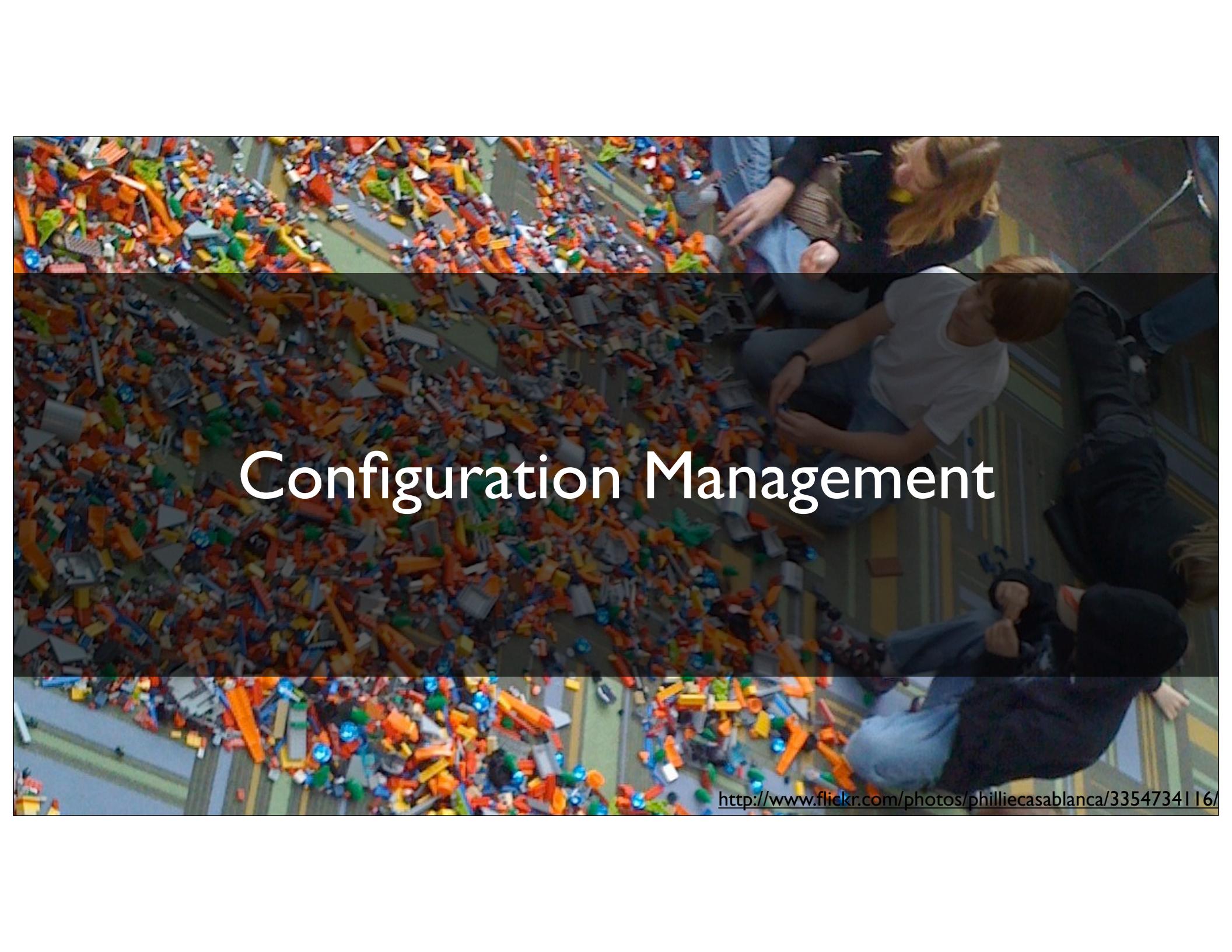


Complexity increases quickly



Complexity increases very quickly



A wide-angle photograph showing a massive pile of colorful LEGO bricks filling most of the frame. Several children are scattered across the surface of the bricks, some sitting and some lying down, appearing to play or search through the pieces. The scene is set indoors with a carpeted floor and walls visible in the background.

Configuration Management

<http://www.flickr.com/photos/philliecasablanca/3354734116/>

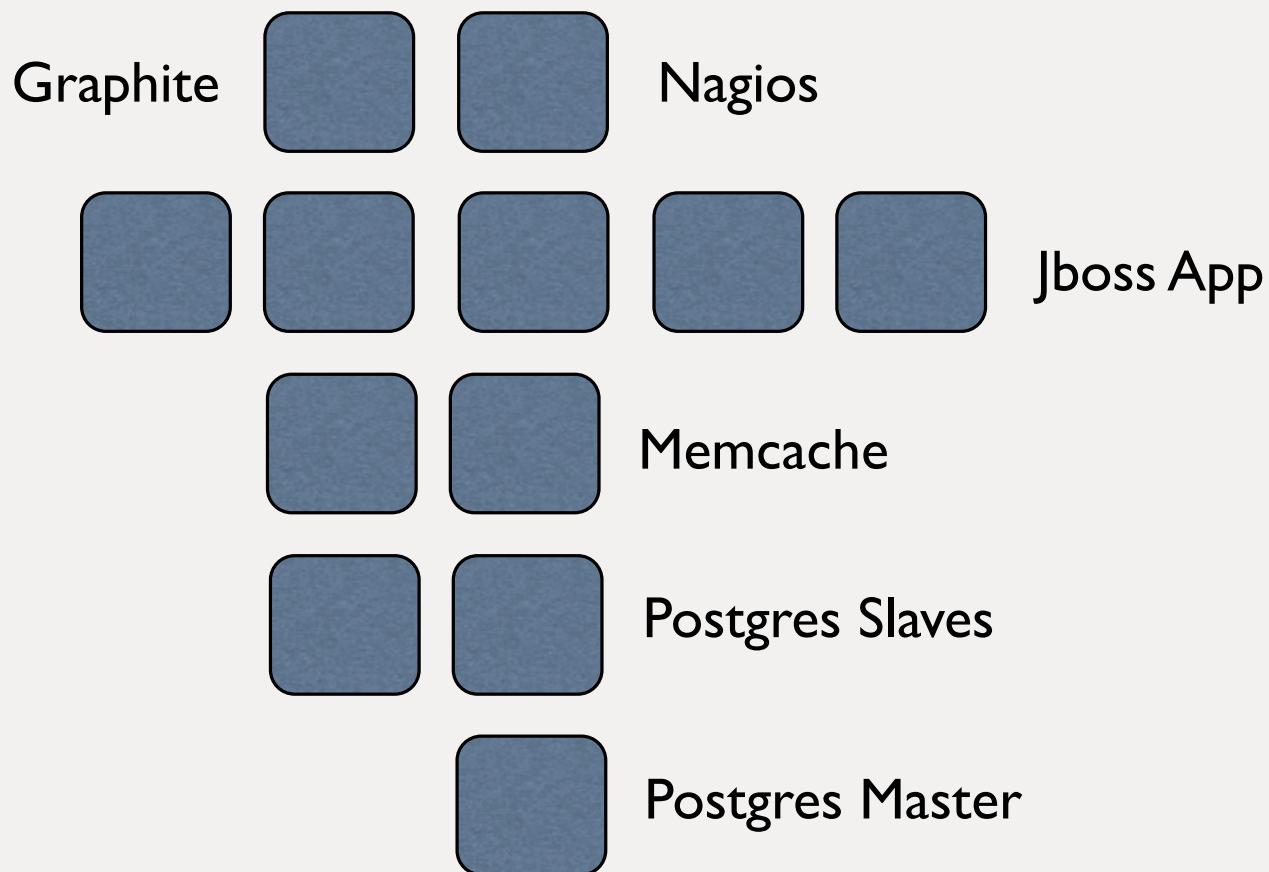
Golden Images are not the answer

- Gold is heavy
- Hard to transport
- Hard to mold
- Easy to lose configuration detail

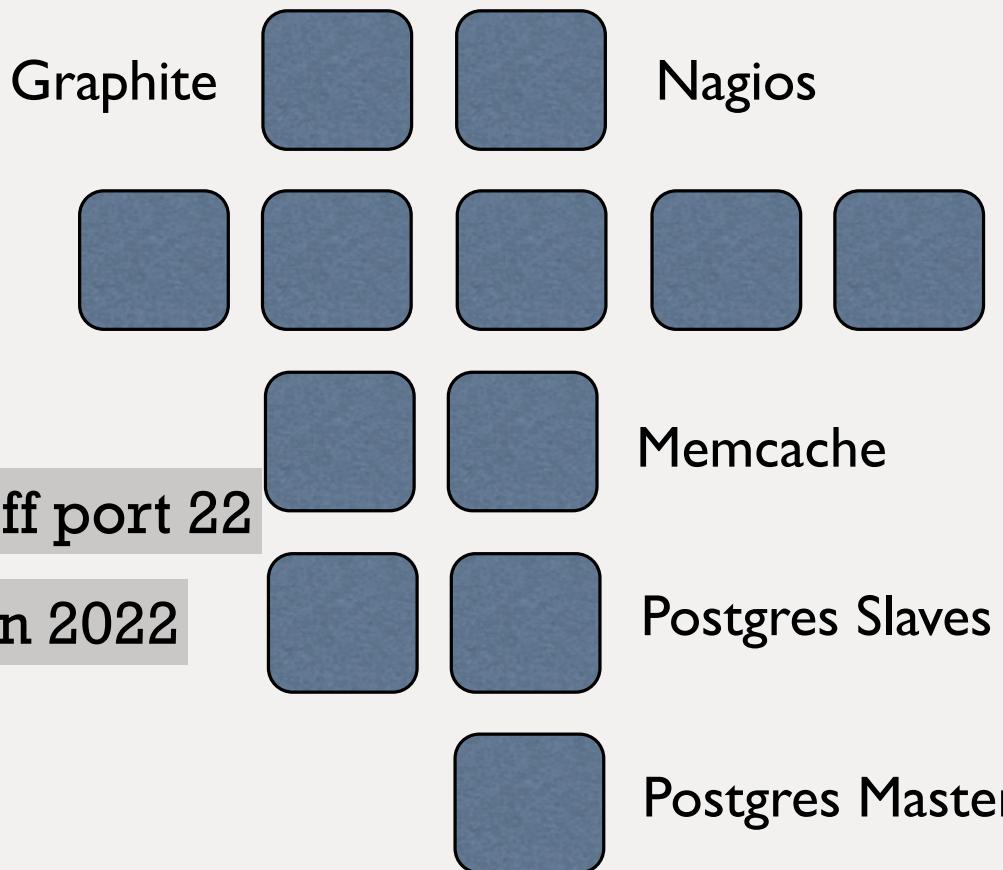


<http://www.flickr.com/photos/garysoup/2977173063/>

Typical Boring Infrastructure



New Compliance Mandate



- Move SSH off port 22
- Lets put it on 2022

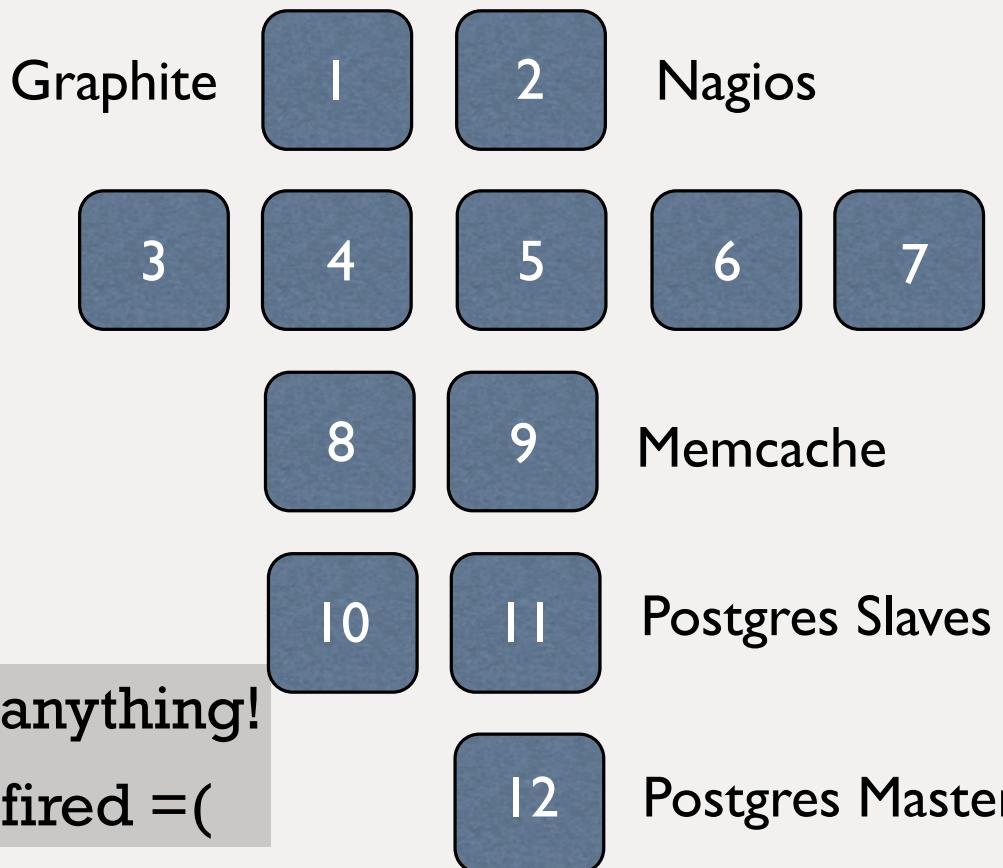
6 Golden Image Updates



12 Instance Replacements

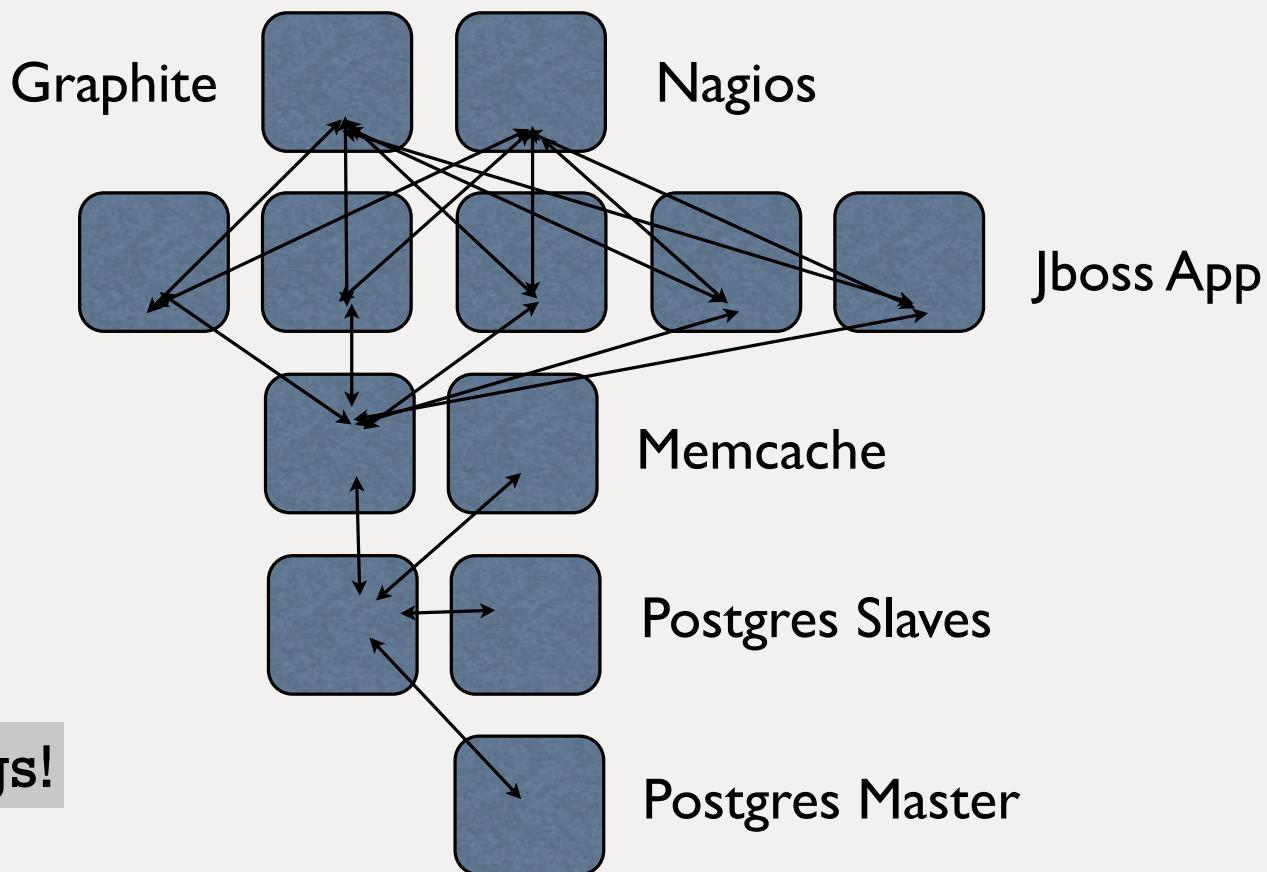


Done in Maintenance Windows



- Don't break anything!
- Bob just got fired =(

Different IP Addresses?



- Invalid configs!



Configuration Desperation



RULE THE

<http://www.flickr.com/photos/francoforeshock/5716969942/>



Chef Solves This Problem



Chef

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



Programs!

- Generate configurations directly on nodes
- Reduce management complexity
- Version control the programs

<http://www.flickr.com/photos/ssoosay/5126146763/>



Declarative Interface to Resources

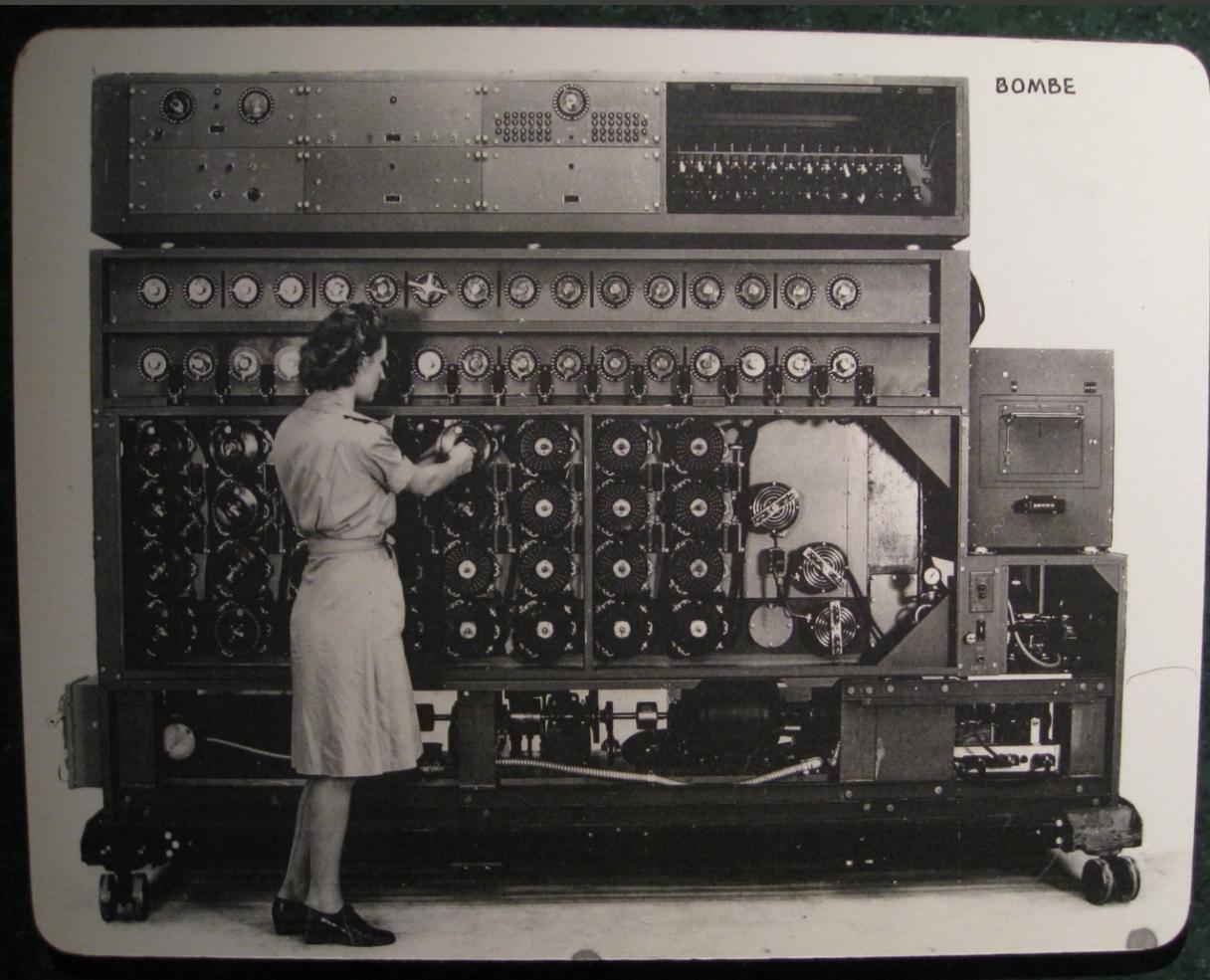
- Define policy
- Say what, not how
- Pull not Push



<http://www.flickr.com/photos/bixentro/2591838509/>



Chef is Infrastructure as Code



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

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



That looks like this

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

```
template "/etc/ntp.conf" do
  source "ntp.conf.erb"
  owner "root"
  group "root"
  mode 0644
  action :create
  variables(:time_server => "time.example.com")
  notifies :restart, "service[ntp]"
end
```

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



Or this

```
package "net-snmp" do
  action :install
end
```

```
template "/etc/snmpd.conf" do
  source "snmpd.conf.erb"
  owner "root"
  group "root"
  mode 0644
  action :create
  variables(:community_string => "not_public")
  notifies :restart, "service[snmpd]"
end
```

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

Ohai!

```
"hostname": "server-1",
"fqdn": "server-1.example.com",
"domain": "example.com",
"network": {
  "interfaces": {
    "eth0": {
      "type": "eth",
      "number": "0",
      "encapsulation": "Ethernet",
      "addresses": {
        "00:0C:29:43:26:C5": {
          "family": "lladdr"
        },
        "192.168.177.138": {
          "family": "inet",
          "broadcast": "192.168.177.255",
          "netmask": "255.255.255.0"
        },
        "fe80::20c:29ff:fe43:26c5": {
          "family": "inet6",
          "prefixlen": "64",
          "scope": "Link"
        }
      }
    }
  }
},
```

```
"memory": {
  "swap": {
    "cached": "0kB",
    "total": "4128760kB",
    "free": "4128760kB"
  },
  "total": "2055676kB",
  "free": "1646524kB",
  "buffers": "35032kB",
  "cached": "210276kB",
  "active": "125336kB",
  "inactive": "142884kB",
  "dirty": "8kB",
  "writeback": "0kB",
  "anon_pages": "22976kB",
  "mapped": "8416kB",
  "slab": "121512kB",
  "slab_reclaimable": "41148kB",
  "slab_unreclaim": "80364kB",
  "page_tables": "1784kB",
  "nfs_unstable": "0kB",
  "bounce": "0kB",
  "commit_limit": "5156596kB",
  "committed_as": "74980kB",
  "vmalloc_total": "34359738367kB",
  "vmalloc_used": "274512kB",
  "vmalloc_chunk": "34359449936kB"
},
```

```
"block_device": {
  "ram0": {
    "size": "32768",
    "removable": "0"
  },
  "ram1": {
    "size": "32768",
    "removable": "0"
  },
  "ram2": {
    "size": "32768",
    "removable": "0"
  },
}
```

Decide what to declare

```
execute "load sysctl" do
  command "/sbin/sysctl -p"
  action :nothing
end

bytes = node['memory']['total'].split("kB")[0].to_i * 1024 / 3,
pages = node['memory']['total'].split("kB")[0].to_i * 1024 / 3 / 2048

# adjust shared memory and semaphores
template "/etc/sysctl.conf" do
  source "sysctl.conf.erb"
  variables(
    :shmmax_in_bytes => bytes,
    :shmall_in_pages => pages
  )
  notifies :run, "execute[load sysctl]", :immediately
end
```

Multiphase Execution

```
size = ((2 * 3) * 4) / 2

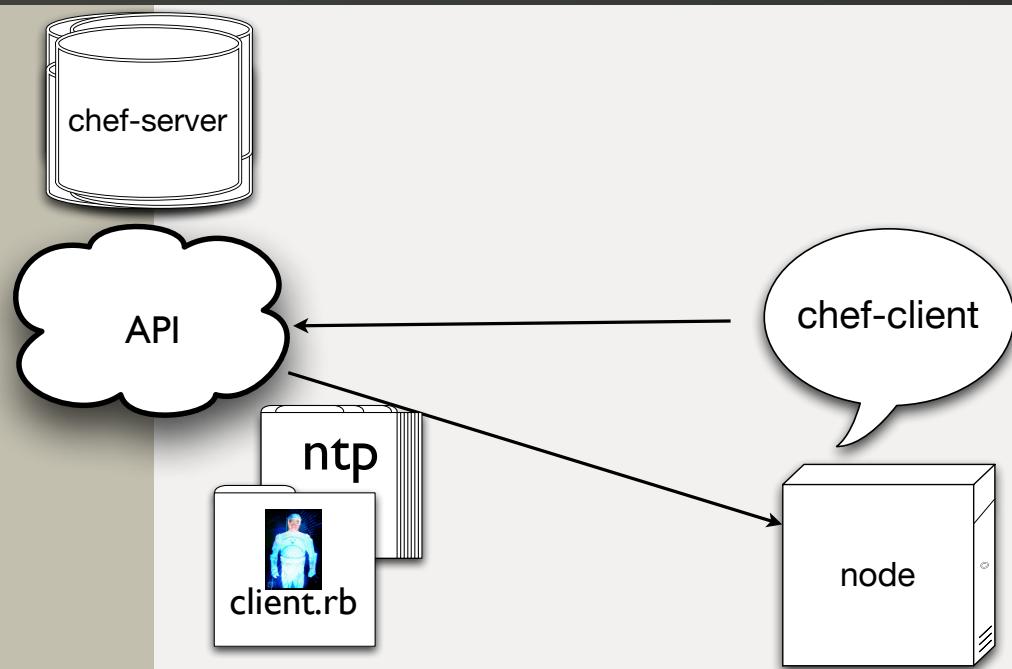
99.downto(1) do |i|
  beer_bottle "bottle-#{i}" do
    oz size
    action [ :take_down, :pass_around ]
  end
end
```

Recipes and Cookbooks

- Recipes are collections of Resources
- Cookbooks contain recipes, templates, files, custom resources, etc
- Code re-use and modularity

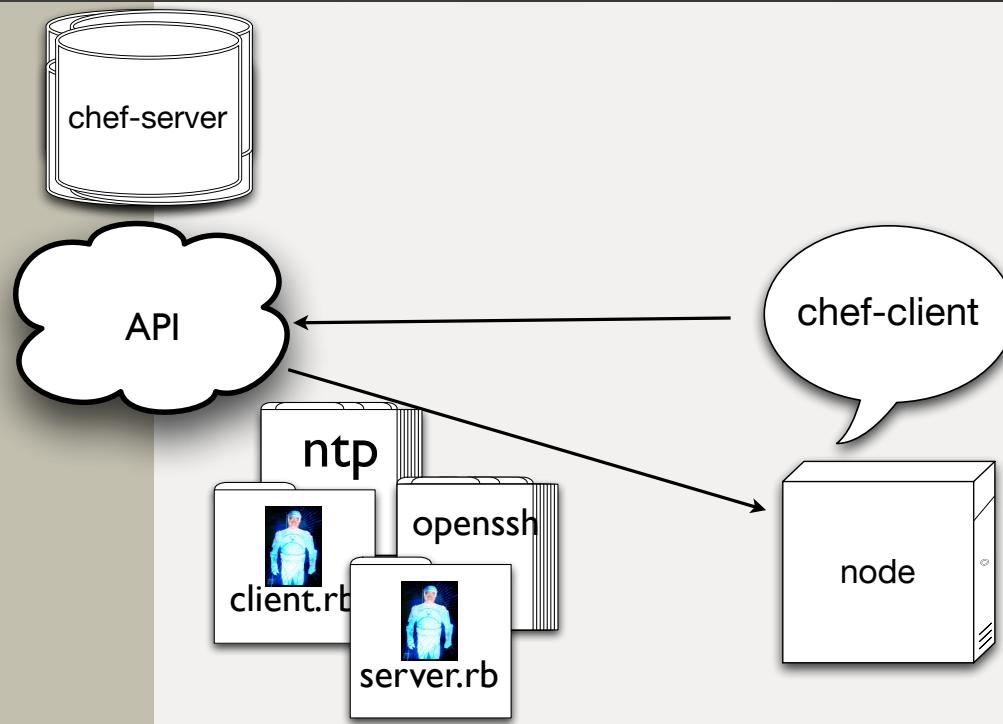


Run Lists



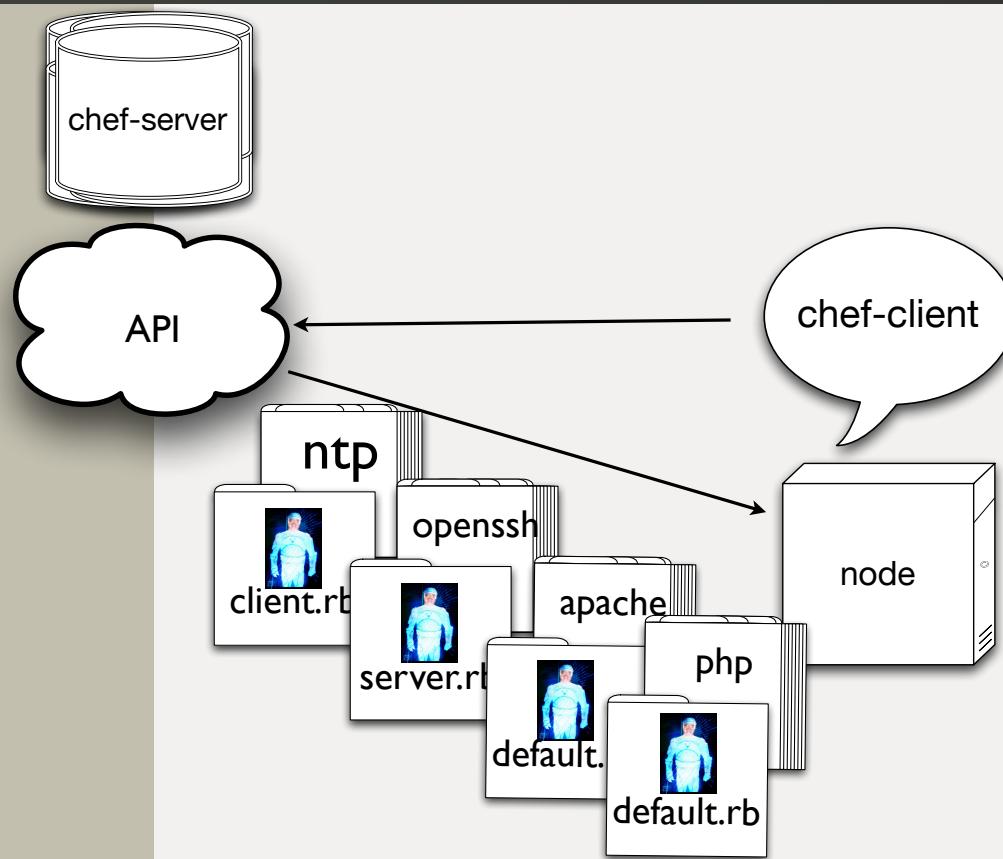
`recipe[ntp::client]`

Run Lists



“recipe[ntp::client]”,
“recipe[openssh::server]”

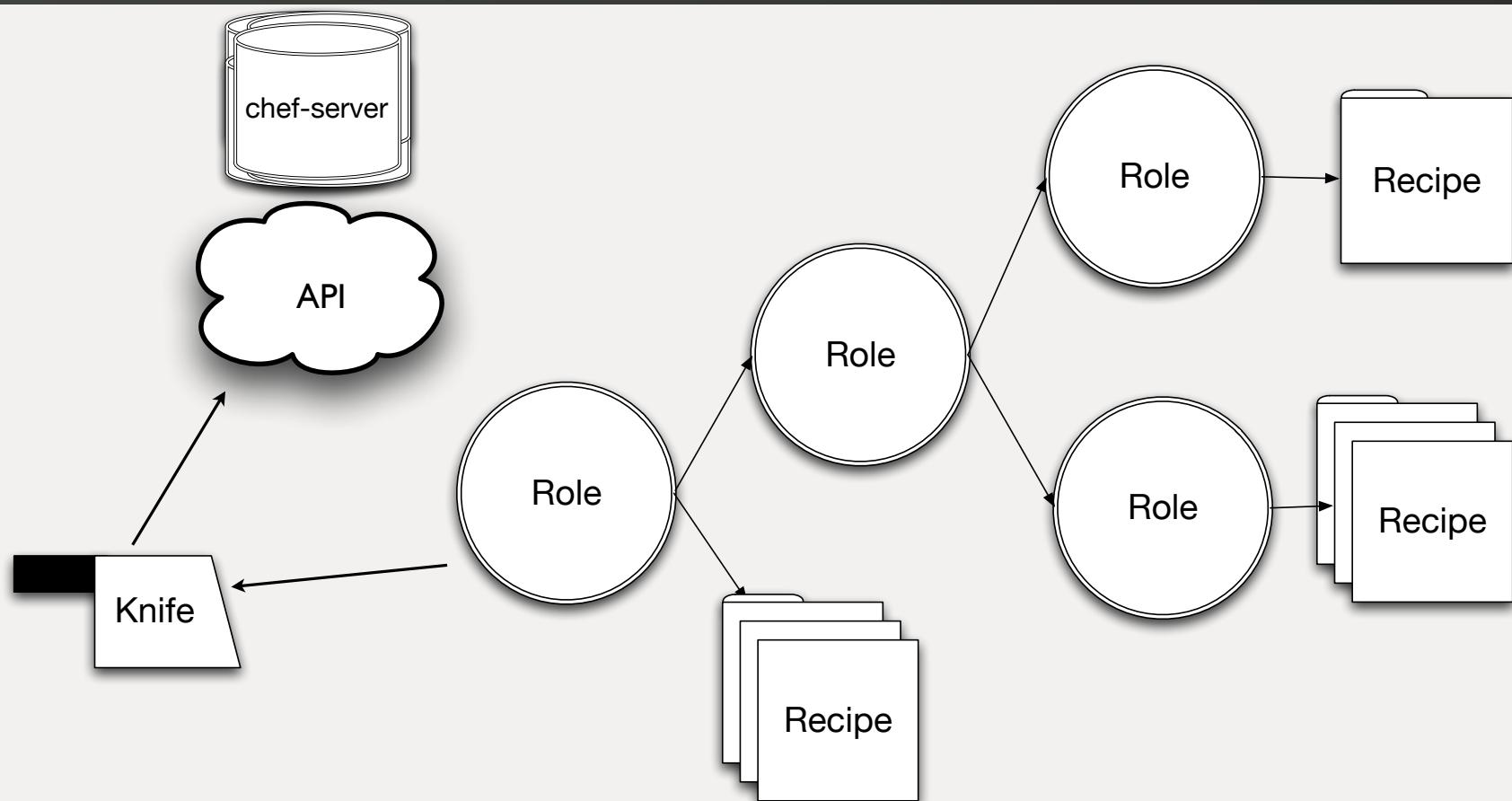
Run Lists



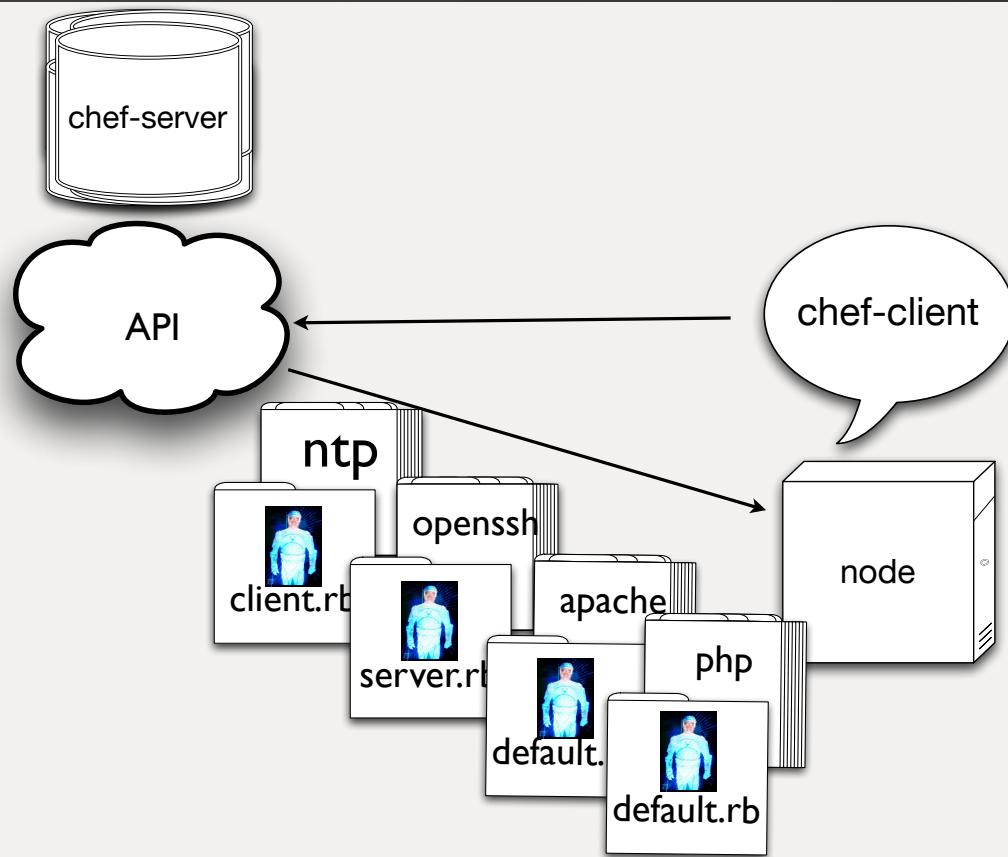
“`recipe[ntp::client]`”,
“`recipe[openssh::server]`”,
“`recipe[apache]`”,
“`recipe[php]`”

```
name "webserver"
description "webserver server"
run_list [
  "role[base]",
  "recipe[nginx::server]"
]
```

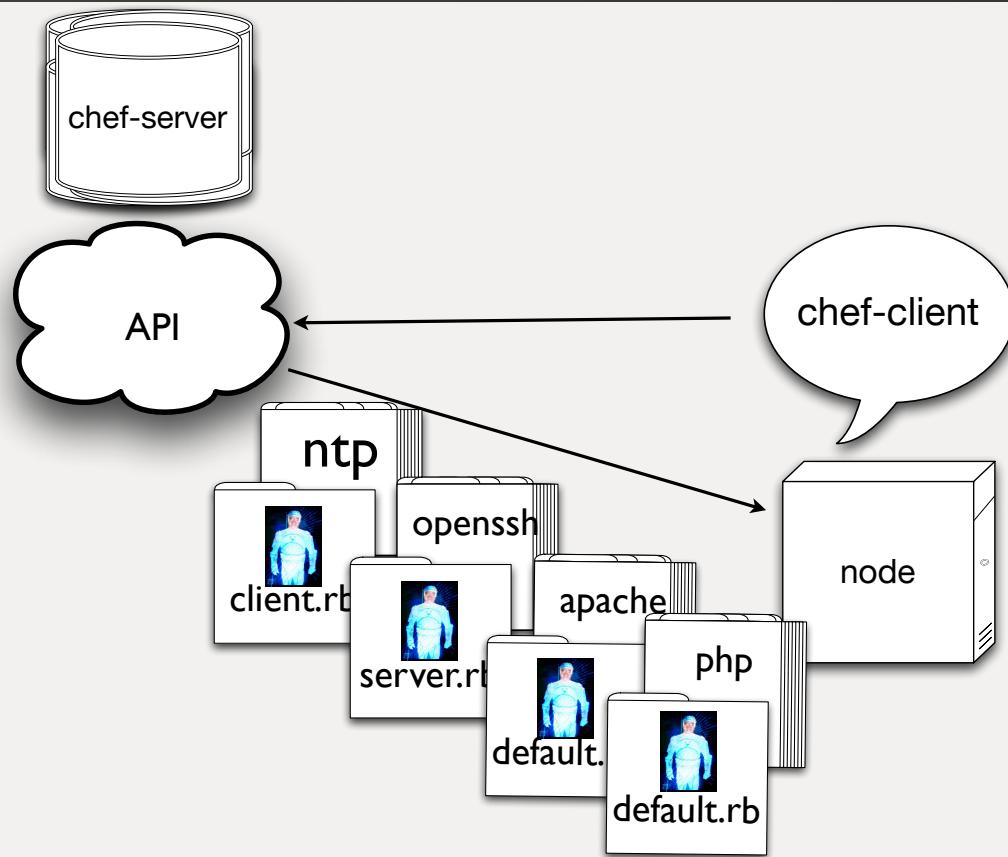
```
name "base"
description "base"
run_list [
  "recipe[selinux::disabled]",
  "recipe[etctools]",
  "recipe[yum::epel]",
  "recipe[debugtools]"
]
```



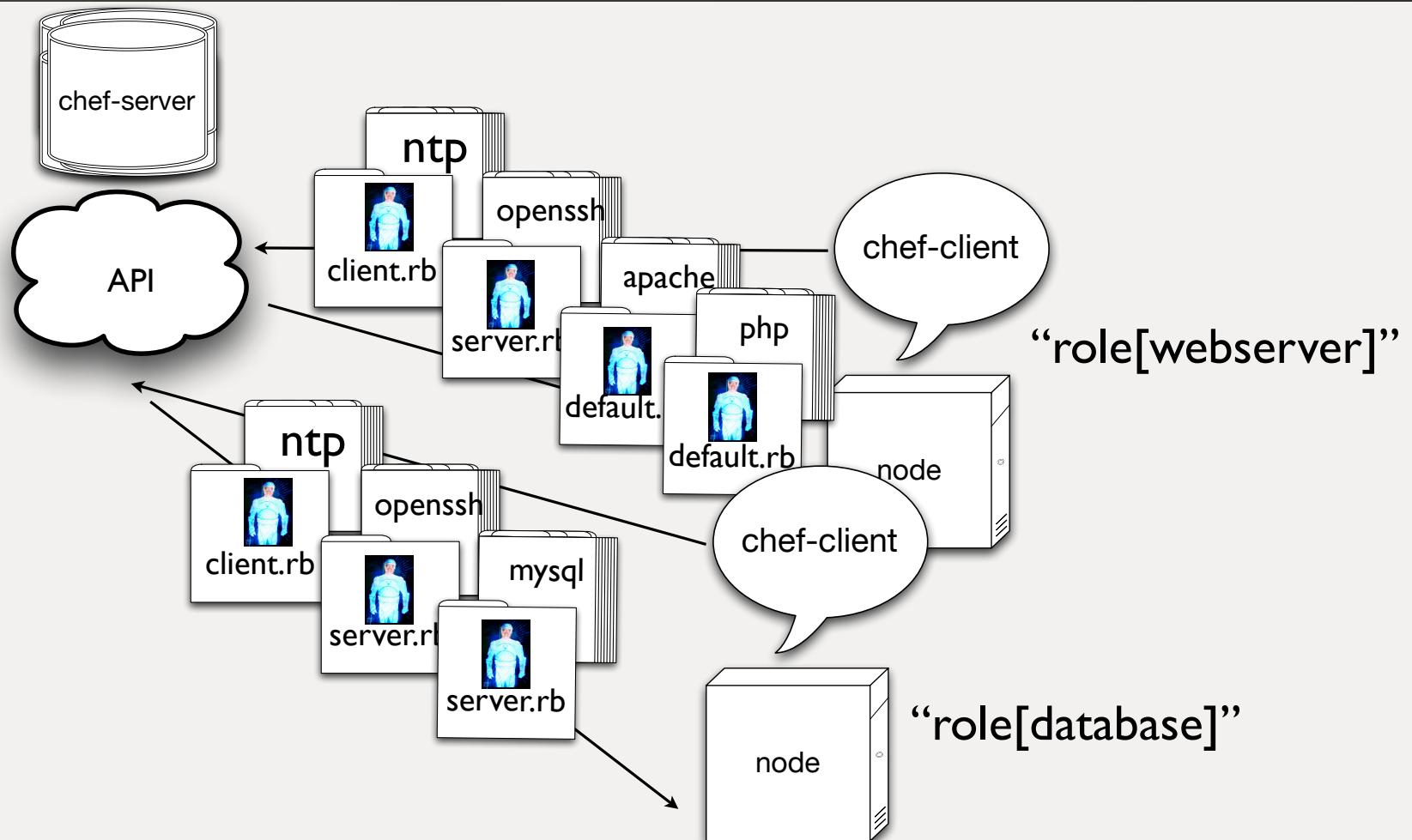
Run Lists



“recipe[ntp::client]”,
“recipe[openssh::server]”,
“recipe[apache]”,
“recipe[php]”



**“role[base]”,
“role[webserver]”**



Search



- Search for nodes with Roles
- Find configuration data
- IP addresses
- Hostnames
- FQDNs

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

Search for nodes

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

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

Pass results into Templates

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

munin::server example

```
node.set[:munin][:server] = true
munin_clients = search(:node, "munin_client:true")

cookbook_file "/etc/cron.d/munin" do
  source "munin-cron"
  mode "0644"
  owner "root"
  group "root"
end

template "/etc/munin/munin.conf" do
  source "munin.conf.erb"
  mode 0644
  variables(:munin_clients => munin_clients)
end
```

munin::client example

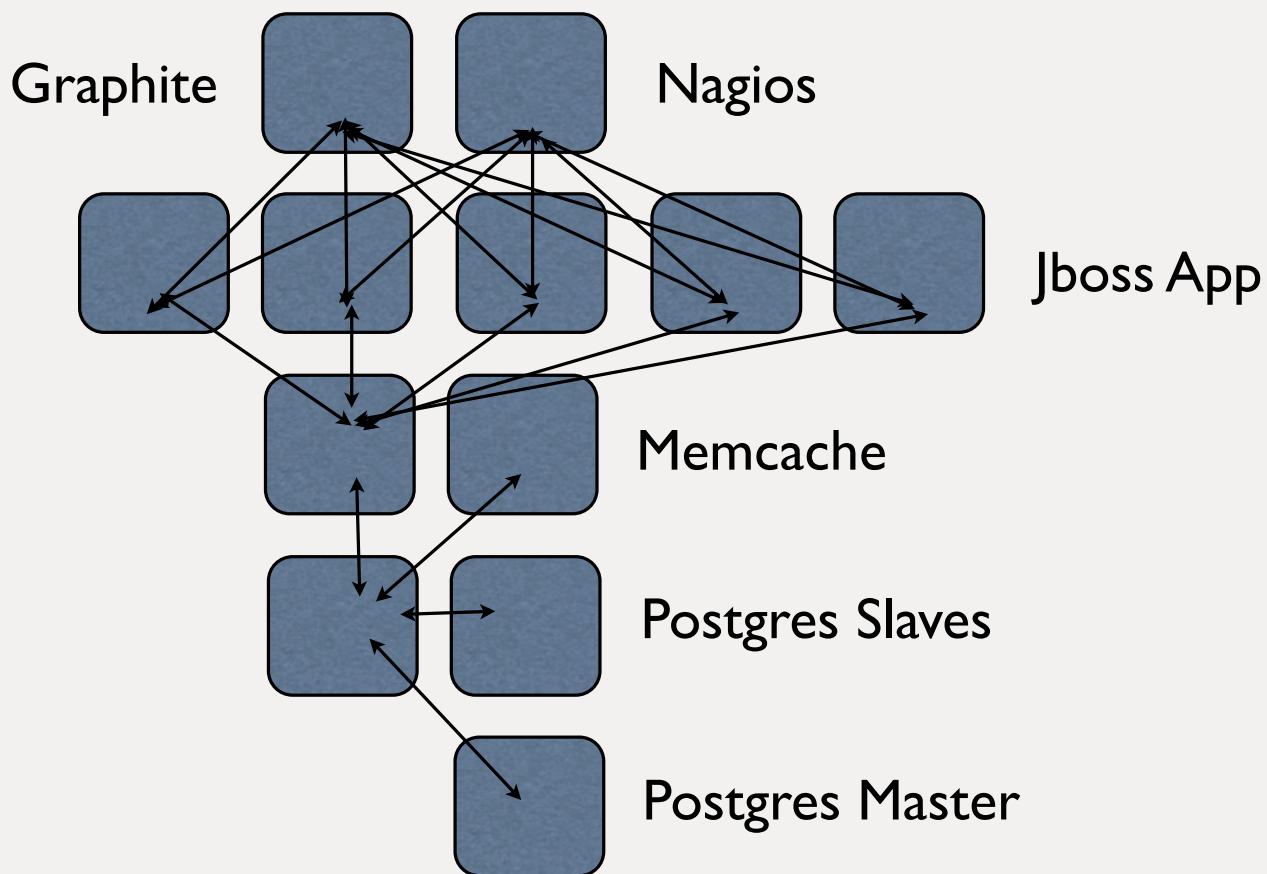
```
node.set[:munin][:client] = true
munin_servers = search(:node, "munin_server:true")

unless munin_servers.empty?
  package "munin-node" do
    action :install
  end

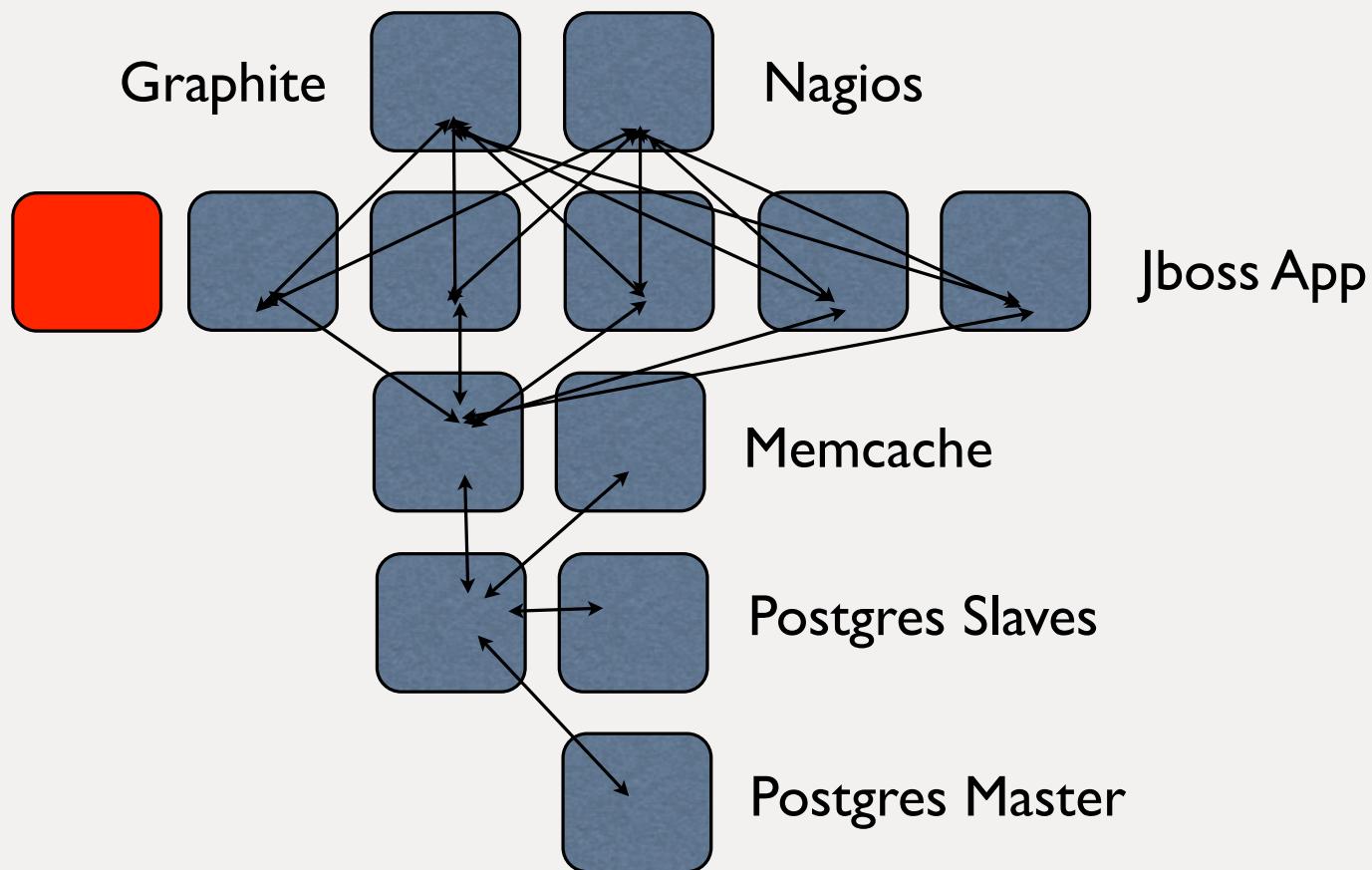
  template "/etc/munin/munin-node.conf" do
    source "munin-node.conf.erb"
    mode 0644
    variables :munin_servers => munin_servers
    notifies :restart, "service[munin-node]"
  end

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

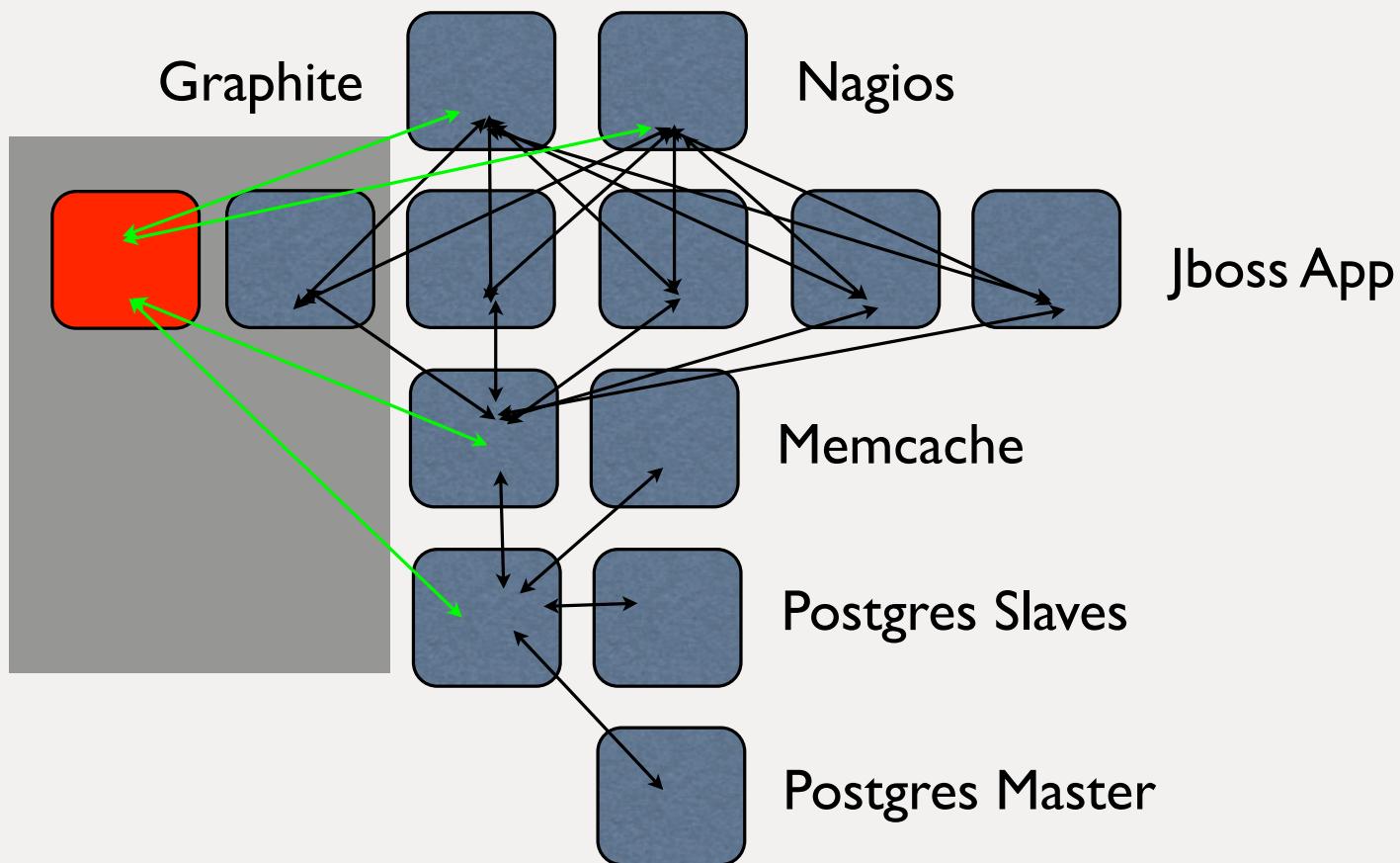
So when this



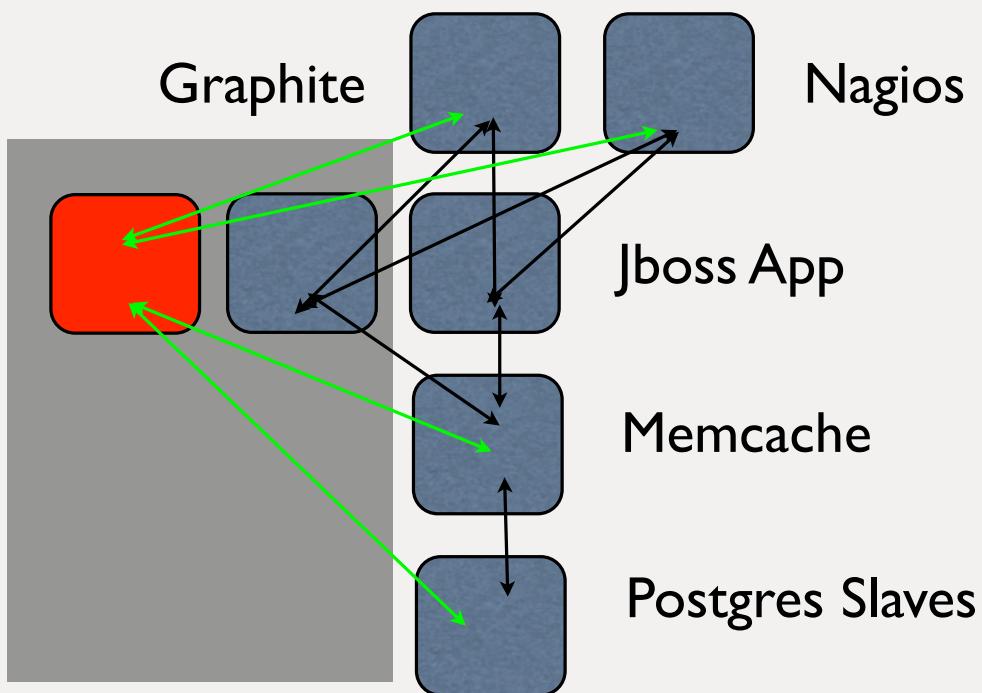
Becomes this



This can happen automatically



Count the resources



- 12+ resource changes for 1 node addition

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



Questions!

Workstation Setup



- Have Chef installed on your Workstation
- Have an empty clone of the Opscode standard chef-repo
- Have a unique User created
- Have an Organization created for use during training
- Understand what Knife is
- Have a working Knife configuration
- Ensure a properly configured text editor is available



Exercise: Installing Chef on Ubuntu or Enterprise Linux

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



Exercise: Installing Chef on Ubuntu, Enterprise Linux, or Mac OSX

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

```
% Total    % Received % Xferd  Average Speed   Time     Time
Time   Current                                         Dload  Upload  Total  Spent
Left   Speed
100  6396  100  6396      0       0  61346      0  --:--:--
--:--:-- --:--:-- 98400
Downloading Chef for ubuntu...
Installing Chef
Unpacking chef (from .../tmp.CP0FBY/chef_amd64.deb) ...
Setting up chef (10.16.0-1.ubuntu.11.04) ...
Thank you for installing Chef!
```

Exercise: Installing Chef on Windows

- Download the Installer to your Desktop
 - <http://www.opscode.com/chef/install.msi>
- Double click to begin the installation
- Once it is complete, log all the way out and back in to ensure the Chef commands are in your command path.

Exercise: Create your Chef Repository

- Every Chef installation has (at least one) Chef Repository.
- Download the following URL
 - <https://github.com/opscode/chef-repo/zipball/master>
- Uncompress it
- Move resulting directory to ~chef-repo
- Create a ~chef-repo/.chef directory

Exercise: Create a User

- Ensure you download your private key
 - Every request made to Chef must be digitally signed with a private key
 - The Chef Server never stores copies of these keys - so keep them safe
 - Copy your private key to `~/chef-repo/.chef`

Exercise: Create an Organization

- Organizations are like sandboxed Chef Servers.
 - A user can be a part of many organizations
 - Download your validation key
 - Download your knife configuration
 - Place them both in `~/chef-repo/.chef`



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



Exercise: Run 'knife help list'

```
$ knife help list
```

Available help topics are:

- bootstrap
- client
- configure
- cookbook
- cookbook-site
- data-bag
- environment
- exec
- index
- knife
- node
- role
- search
- shef
- ssh
- status
- tag



Exercise: Run ‘knife client list’

```
$ knife client list
```

```
opscode-training-validator
```



OPSCODE

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++, Eclipse, etc.

RULE THE CLOUD



Best Practice: Set your EDITOR

Linux/OSX/Unix, in your ~/.bash_profile

```
export EDITOR=/path/to/emacs  
source ~/.bash_profile
```

Windows in cmd.exe

```
setx EDITOR "\"%ProgramFiles%\Windows NT\Accessories\wordpad.exe\""  
or  
setx EDITOR "\"%programfiles%\Sublime Text 2\sublime_text.exe\" --wait"  
close cmd.exe and start a new cmd.exe session
```



Exercise: Verify a working EDITOR is configured

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

A screenshot of a terminal window titled "knife-edit-040246275713260737350.js". The window shows the following JSON configuration:

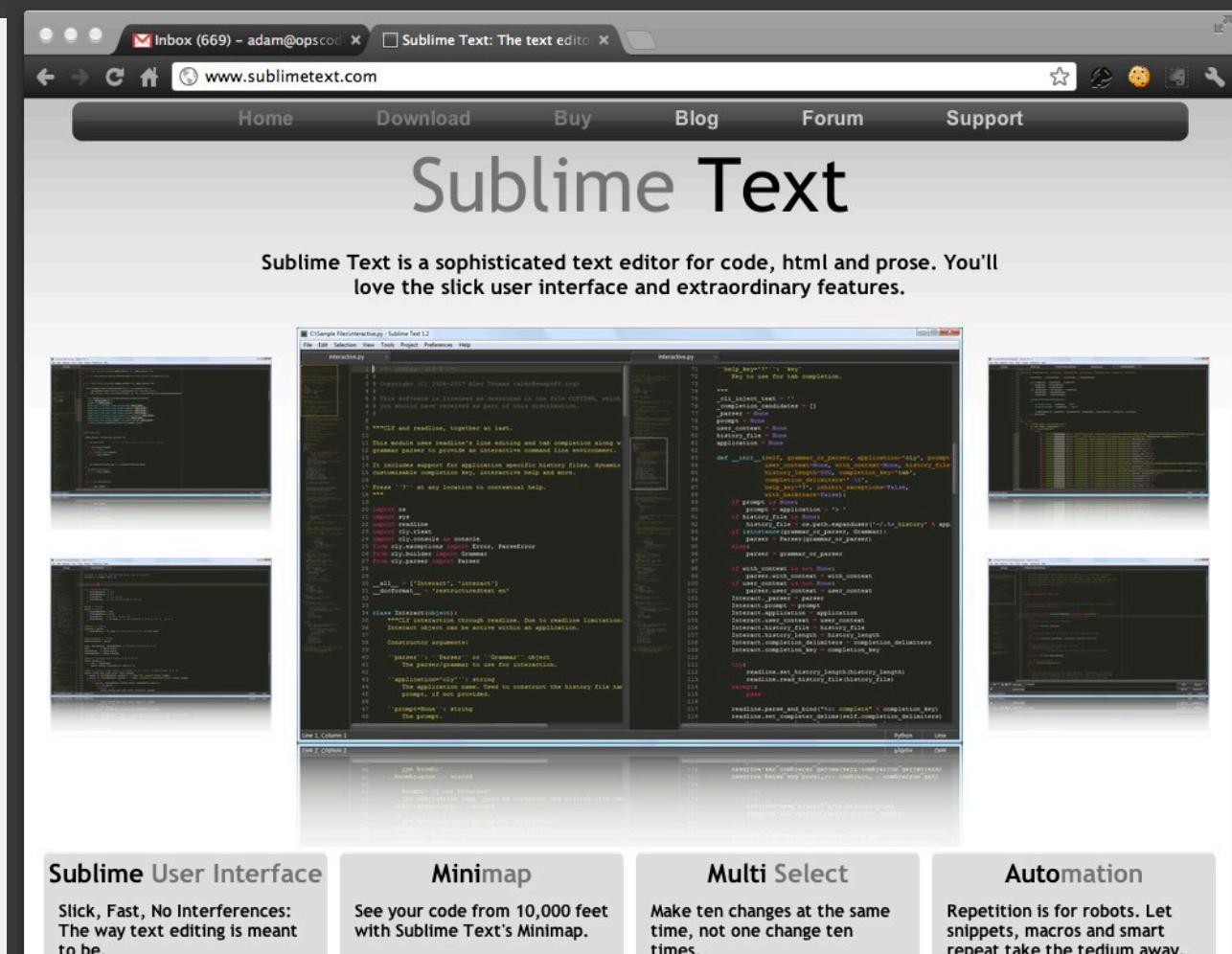
```
1
2   "name": "editor-test",
3   "public_key": null,
4   "admin": false,
5   "json_class": "Chef::ApiClient",
6   "chef_type": "client"
7
```

The terminal prompt shows the file path: "/private/var/folders/0c/j_mymdt9063_195pf830x70r0000gn/T/knife-edit-040246275713260737350.js".

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

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
- <http://sublimetext.com>



The screenshot shows the Sublime Text website (www.sublimetext.com) with the following content:

- Sublime Text**: The main heading.
- Sublime Text is a sophisticated text editor for code, html and prose. You'll love the slick user interface and extraordinary features.**
- Interactive.py**: A screenshot of the Sublime Text editor showing a Python file named `Interactive.py` with code related to readline and tab completion.
- Sublime User Interface**: A section describing the editor's sleek and fast interface.
- Slick, Fast, No Interferences: The way text editing is meant to be.**
- Minimap**: A section showing the minimap feature.
- See your code from 10,000 feet with Sublime Text's Minimap.**
- Multi Select**: A section about multi-select editing.
- Make ten changes at the same time, not one change ten times.**
- Automation**: A section about automation features.
- Repetition is for robots. Let snippets, macros and smart repeat take the tedium away.**



Using Sublime on Mac OS X

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



Using Sublime on Windows

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



Exercise: Verify a working EDITOR is configured

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

A screenshot of a terminal window titled "knife-edit-040246275713260737350.js". The window shows the following JSON configuration:

```
1
2   "name": "editor-test",
3   "public_key": null,
4   "admin": false,
5   "json_class": "Chef::ApiClient",
6   "chef_type": "client"
7
```

The terminal prompt shows the file path: "/private/var/folders/0c/j_mymdt9063_195pf830x70r0000gn/T/knife-edit-040246275713260737350.js".

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



Exercise: Verify a working EDITOR is configured

```
Created client[editor-test]
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAv313jX0crYxdNsqXTf lh2N6e3kGSUTCJRVwCXIFh6YmdoY3c
bcIWESebcL5y5Vta3HjTsmohkSIW3JF0BLK39fQ8V0dM0wSBmZm9NvJwKucuwRes
u8EszMeLY/9tGFD9TK0iFlPN1jtRE1Ym61QRz iRyAvkbkPEZacIVdayNM5TS8LuN
BSdYZM8vnqRr6cf7WJn0m+lIKdcR1plHWtQT9EgYyPBVlwW9R8fR1i/QHe0yGz2
RTlYYAq+qvUimv30QoZXca/nkxroFWj80s9RJCgPdI44n51ws/PPmPQYeWykZft
9+pg5dud+Q1WbxR5yH01FDGVGaoFBB6mny+JaQIDAQABoIBAQCeKdC8gfFU+1Hm
trEAi5IhIcfQxhJHkzJeop+kro0j6zHSxayiz60Qacl+9x9JiAplfj rvBPhSuvht
GIrecot10FSitlajGN6+8vgRUXAKT8cTYC2hKu6I+eyNH0WMJyEA2yQDkxa582aE
9SzSRS8ruHifMght0GZwLwmwl2RNjBpbqWK6YI24yv6ljgqkcUoC7gGECMKWFoE
4DLYKnmza+0WeVe+xeC9R9gCz4qqq56U06RptcJ1jLQNzjLshv/1x1tRgyiy1k7Q
INmMPYcproh2Frmx5sipBml2nxGUNfJncMMefV9AGddIdvPYzY2Wp5vZukTUVTxB
p003T2sBAoGBAoFRX6cEAvh4P8qZ3UqzDbBuuREJ6XGWNoMdjk4CgJn7owGXBig
XIiB4du30lCfD2G7p2FFipzaxylEVHTd9djT5NA6FBvUL6ATo0cS6lteTwixwbth
1RhM2im8DYx+RR2PjpsaerQw3DBVx19EUaU7LS5W7CSMuHuEjhpru1QhAoGBANmQ
2c3NGshBy06d0fKSXIU5h2nyM7N33tsj08upgerA1QXxcRLjWJ2Xr12W+npeFtTV
LtSsBV9HirayhfVqiSiLj1ulS+GbFHMZy8i9vu009JM+ysI0NY1XPhxcNjpqRZLn
/ppAbBaZCunx9S7zLRN/YQZ0dsScRoYkH//UcwxJAoGBAMiuVHaKESdC2vZVco6s
yZ2CEJ1Hab98XGRCgGCKw+vh/z5UR+yFlj8TB5pNMhT0zxmCd+0M7Ye+dIch0i0
JQDZQWvgvrZGl6CIvR15kHi0l/ATeoYWhnwqEsK8JSjv+3wpEKWyl+fHxrrvy0p/0
Vo/HFCe4zZyEJXAGD6SdvXxhAoGBAkfUWF+sV3uhXW7QVFIt21cd8LqmjoFj97K+
KXRS1xg7FljWTi60194Bk9KzU5cm5ckuFJwPFiqfHPAtRuCyjlPpw/ALA/lrfm5
zXyV+nnZ3gr0Bj7XPXRBl3UTIPXg4riXY8yj431vI38iGcvU5LHEshjGUfIMZ060
8UZNIU6ZAoGAGXIrvnC/vCSmt8wEPl3F5xfp9P3w+7gZExl8lo104bSxtTIF3+BP
M/0QbtBm294EBzyyRZzTEXhaAByGI07vXeRuv9H8cpFnSlrMaAXB+WV+drh0lL3J
AWlIT8ILi3VhbmG+1Kx7CsK8JinBey3dPvfJ8L9l+gbsEU+3Cv8+U8M=
-----END RSA PRIVATE KEY-----
```



Exercise: Clean up after our EDITOR test

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

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

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

- What is the chef-repo?
- What is an organization?
- What is knife?
- What is your favorite text editor? :)

Test Node Setup



- Have Chef installed on your test node
- Have the Chef Client configured to use the organization created in the previous section
- Successfully execute your first chef-client run.



Log in to your test node now



Exercise: Installing Chef on Ubuntu

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

```
% Total    % Received % Xferd  Average Speed   Time     Time     Time
Current                                         Dload  Upload   Total   Spent   Left   Speed
100  6396  100  6396      0      0  61346       0  --:--:--  --:--:--  --:--:--  98400
Downloading Chef for ubuntu...
Installing Chef
Selecting previously unselected package chef.
(Reading database ... 25367 files and directories currently installed.)
Unpacking chef (from .../tmp.CP0FByIY/chef_amd64.deb) ...
Setting up chef (10.16.0-1.ubuntu.11.04) ...
Thank you for installing Chef!
Processing triggers for initramfs-tools ...
update-initramfs: Generating /boot/initrd.img-3.2.0-23-virtual
```

- The chef-client is the program that runs on all of your nodes, ensuring they are in the desired state.
- It is configured through `/etc/chef/client.rb`
- The configuration file itself is written in ruby, a trend you are about to see a lot of with Chef.



Exercise: Populate /etc/chef/client.rb

```
$ sudo mkdir -p /etc/chef  
$ sudo nano /etc/chef/client.rb
```

```
log_level :info  
log_location STDOUT  
chef_server_url "https://api.opscode.com/organizations/training"  
validation_client_name 'training-validator'  
validation_key '/etc/chef/training-  
validator.pem'
```

Exercise: Populate the validation key

- The **validation key** (or **validator** for short) is a key unique to each organization that can be used only to create new clients.
- Your validation key can be found in `~/chef-repo/.chef/ORGANIZATION-validator.pem`
- Place the key in `/etc/chef` on the test node.
 - This is typically handled by your companies bootstrapping process



Exercise: Run chef-client for the first time

```
$ sudo chef-client
```

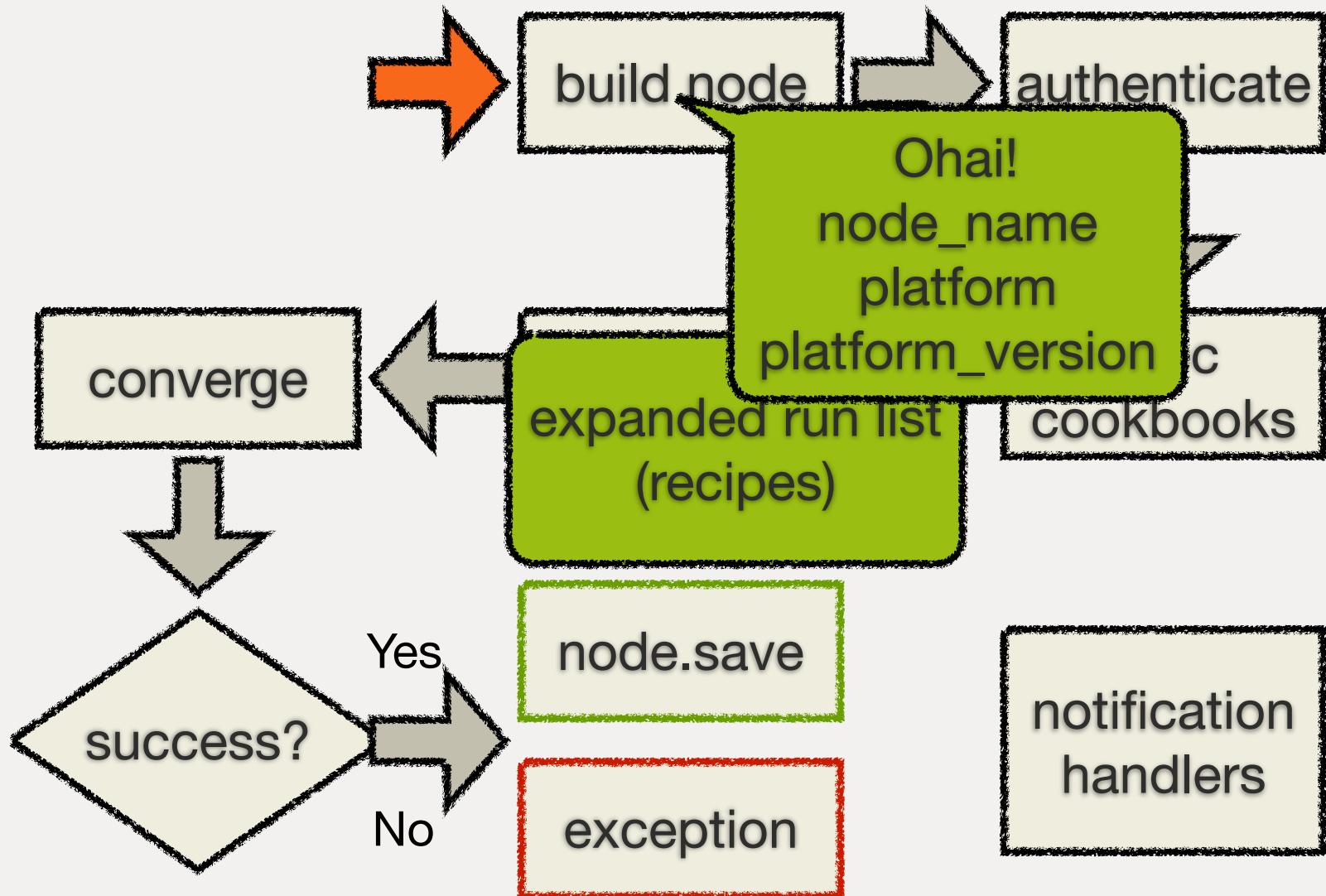
```
INFO: *** Chef 10.16.0 ***
INFO: Client key /etc/chef/client.pem is not present - registering
INFO: Run List is []
INFO: Run List expands to []
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks []
WARN: Node target1.local has an empty run list.
INFO: Chef Run complete in 0.807242752 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

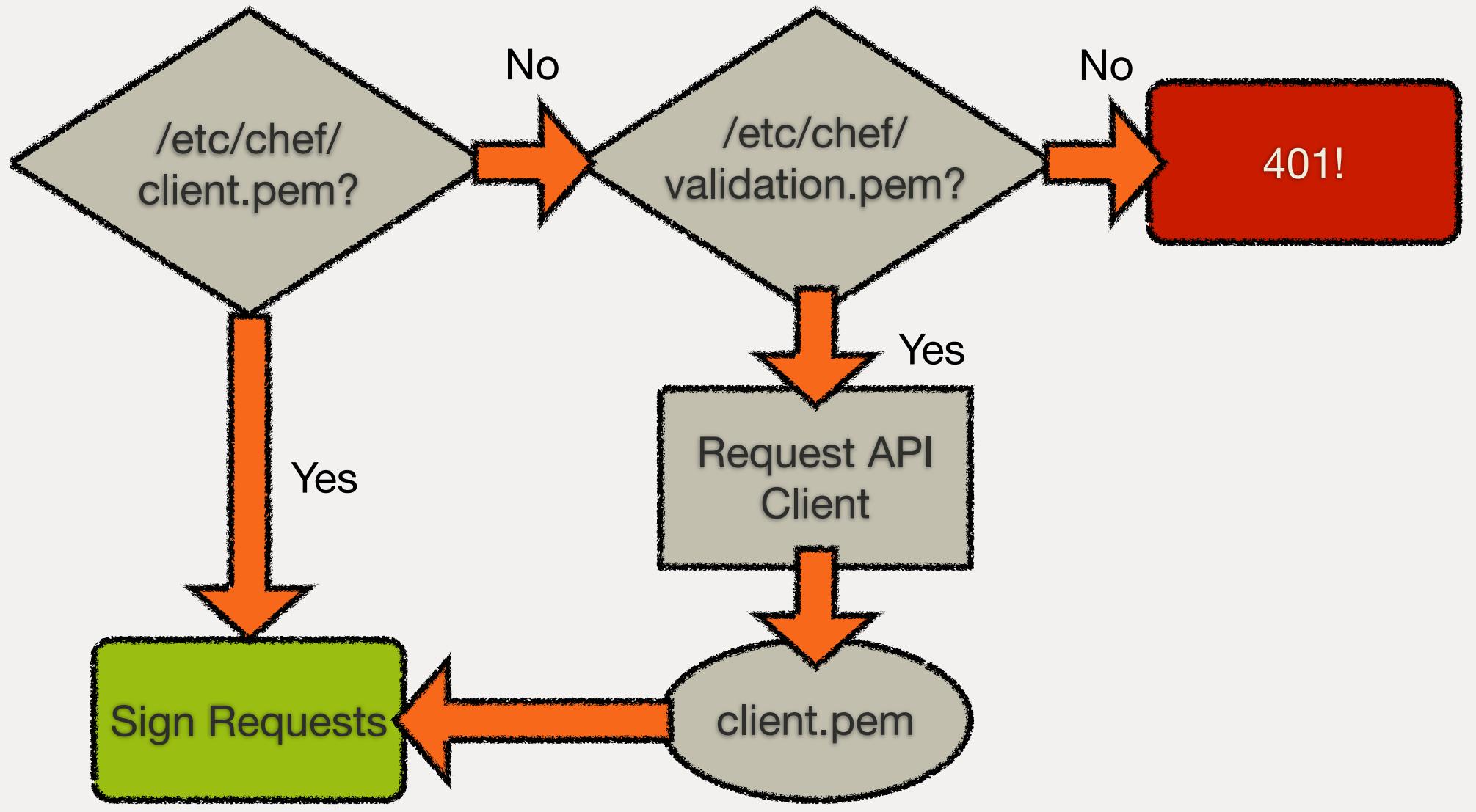
- Where is the chef-client configuration file?
- What is the command to run chef?
- What is the validation key used for?

Dissecting your first Chef run



- Understand the steps taken by a chef-client during a run
- Understand the basic security model of Chef





- What are the steps in a chef client run?
- How does a new server get a private key with which to authenticate requests?

Introducing the Node object

Attributes & Search



- Understand what the Node object represents in Chef
- Learn how to list the Nodes in an organization
- Know how to show details about a Node
- Understand what Node Attributes are
- Know how to add an attribute to a Node
- Know how to retrieve a node attribute directly, and via search

- A Node object in Chef represents anything that needs to be configured.
- For now, just think of all the servers you need to configure as Node
- When you are writing Cookbooks, the Node object is almost always available to you.



Exercise: Listing nodes

```
$ knife node list
```

target1.local

Each node must have a unique name

- Every node must have a unique name
- Chef defaults to the *Fully Qualified Domain Name* of the server.



Exercise: Showing details about a node

```
$ knife node show target1.local
```

```
Node Name: target1.local
Environment: _default
FQDN: target1.local
IP: 50.19.162.97
Run List:
Roles:
Recipes:
Platform: ubuntu 12.04
Tags:
```

- Nodes are made up of Attributes
 - Many are discovered **automatically** (platform, ip address, number of CPUs)
 - Attributes can be added directly to a node
 - Many other objects in Chef can also add Node attributes (Cookbooks, Roles and Environments)



Exercise: Showing all the attributes of a node

```
$ knife node show target1.local -l
```

```
Node Name: target1.local
Environment: _default
FQDN: target1.local
IP: 50.19.162.97
Run List:
Roles:
Recipes:
Platform: ubuntu 12.04
Attributes:
tags: []

Default Attributes:

Override Attributes:

Automatic Attributes (Ohai Data):
block_device:
loop0:
removable: 0
size: 0
loop1:
removable: 0
size: 0
```



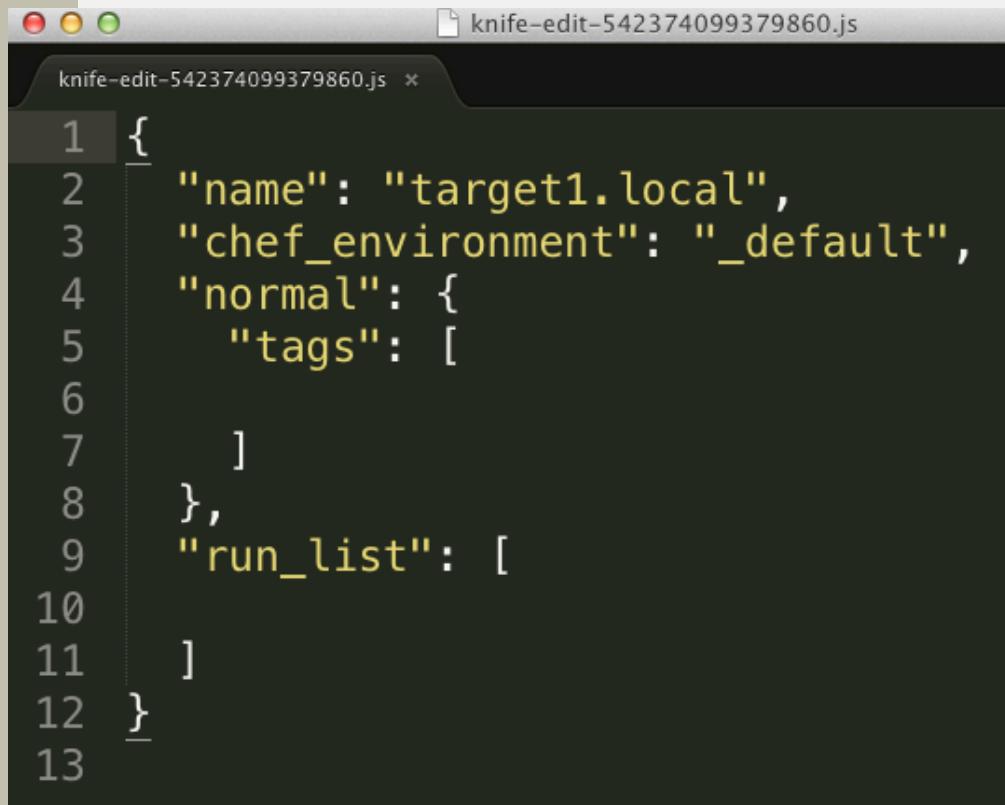
Exercise: Showing the raw format of the node

```
$ knife node show target1.local -Fj
```

```
{
  "name": "target1.local",
  "chef_environment": "_default",
  "run_list": [
    ],
  "normal": {
    "tags": [
      ]
  }
}
```

Exercise: Add an attribute to a node

```
$ knife node edit target1.local
```

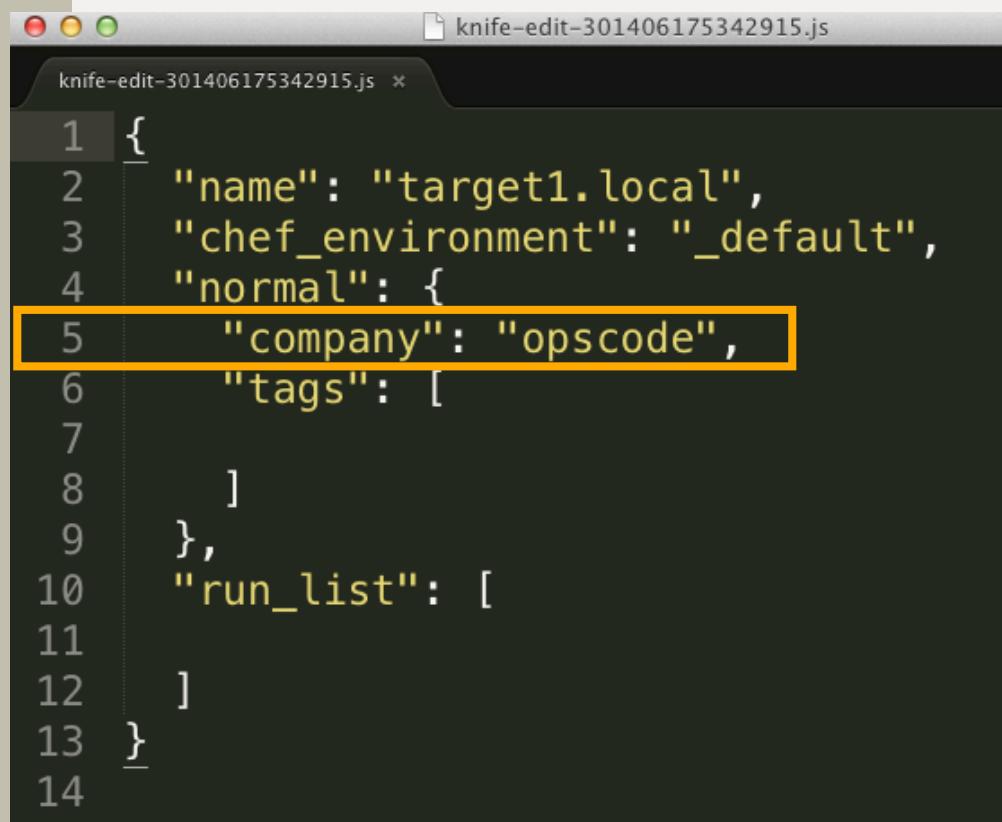


```
knife-edit-542374099379860.js ×
```

```
1 {  
2   "name": "target1.local",  
3   "chef_environment": "_default",  
4   "normal": {  
5     "tags": [  
6       ]  
7     },  
8     "run_list": [  
9       ]  
10    ]  
11  }  
12 }  
13 }
```

Exercise: Add an attribute to a node

Add a “company” attribute, **save** and **close** your editor



```
knife-edit-301406175342915.js *
1  {
2    "name": "target1.local",
3    "chef_environment": "_default",
4    "normal": {
5      "company": "opscode",
6      "tags": [
7        ],
8      },
9    },
10   "run_list": [
11     ],
12   }
13 }
```

- The changed Node data is sent to the Chef Server
- It is stored in the database
- It is indexed for search



Exercise: Show only the new company attribute

```
$ knife node show target1.local -a company
```

```
company: opscode
```



Exercise: Use search to find the same data

```
$ knife search node “*:*” -a company
```

```
1 items found
```

```
company: opscode
id: target1.local
```

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

Writing an Apache cookbook

Packages, Cookbook Files, and Services



- Understand what a **cookbook** is
- Know how to create a new cookbook
- Understand what a **recipe** is
- Understand how to use the **package**, **service**, and **cookbook_file** resources
- Know how to **upload a cookbook** to the Chef Server
- Understand what a **run list** is, and how to set it for a node via knife
- How to read the output of a chef-client run

- 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.

The Problem and the Success Criteria

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

Required steps

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



Exercise: Create a new cookbook

```
$ knife cookbook create apache
```

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



Exercise: Check out what just got created

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

```
total 23
drwxr-xr-x 12 opscode    pandas   442 Oct 22 18:57 .
drwxr-xr-x  4 opscode    pandas   136 Oct 22 18:57 ..
-rw-r--r--  1 opscode    pandas   413 Oct 22 18:57 CHANGELOG.md
-rw-r--r--  1 opscode    pandas   88  Oct 22 18:57 README.md
drwxr-xr-x  2 opscode    pandas   68  Oct 22 18:57 attributes
drwxr-xr-x  2 opscode    pandas   68  Oct 22 18:57 definitions
drwxr-xr-x  3 opscode    pandas  102  Oct 22 18:57 files
drwxr-xr-x  2 opscode    pandas   68  Oct 22 18:57 libraries
-rw-r--r--  1 opscode    pandas  250  Oct 22 18:57 metadata.rb
drwxr-xr-x  2 opscode    pandas   68  Oct 22 18:57 providers
drwxr-xr-x  3 opscode    pandas  102  Oct 22 18:57 recipes
drwxr-xr-x  2 opscode    pandas   68  Oct 22 18:57 resources
drwxr-xr-x  3 opscode    pandas  102  Oct 22 18:57 templates
```



Exercise: Open the default apache recipe in your editor

```
$ vi cookbooks/apache/recipes/default.rb
```

```
1 #
2 # Cookbook Name:: apache
3 # Recipe:: default
4 #
5 # Copyright 2012, YOUR_COMPANY_NAME
6 #
7 # All rights reserved - Do Not Redistribute
8 #
9
```

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



Exercise: Add a package resource to install Apache to the default recipe

Add the following to cookbooks/apache/recipes/default.rb

```
7 # All rights reserved - Do Not Redistribute
8 #
9
10 package "apache2" do
11   action :install
12 end
13
```

- Have a type.
- Have a name.
- Have parameters.
- Take action to put the resource in the declared state.
- Can send notifications to other resources.

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

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

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

So the resource we just wrote...

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

```
10 package "apache2" do
11   action :install
12 end
```

Notice we didn't say how to install the package

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

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

Add the following to cookbooks/apache/recipes/default.rb

```
10 package "apache2" do
11   action :install
12 end
13
14 service "apache2" do
15   action [ :start, :enable ]
16 end
17
```

So the resource we just wrote...

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

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

- The order you write resources in a recipe is the order they will be executed in



Exercise: Add a cookbook_file resource to copy the home page in place

Add the following to cookbooks/apache/recipes/default.rb

```
14 service "apache2" do
15   action [ :start, :enable ]
16 end
17
18 cookbook_file "/var/www/index.html" do
19   source "index.html"
20   mode "0644"
21 end
22
```

So the resource we just wrote...

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

```
18 cookbook_file "/var/www/index.html" do
19   source "index.html"
20   mode "0644"
21 end
```

Best Practice: Omit the action if it is the default

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

```
18 cookbook_file "/var/www/index.html" do
19   source "index.html"
20   mode "0644"
21 end
```

Full contents of the apache recipe

```
default.rb (~/src/sandbox/...okbooks/apache/recipes) – VIM
1 #
2 # Cookbook Name:: apache
3 # Recipe:: default
4 #
5 # Copyright 2012, YOUR_COMPANY_NAME
6 #
7 # All rights reserved - Do Not Redistribute
8 #
9
10 package "apache2" do
11   action :install
12 end
13
14 service "apache2" do
15   action [ :start, :enable ]
16 end
17
18 cookbook_file "/var/www/index.html" do
19   source "index.html"
20   mode "0644"
21 end
~
```



Question!

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



Best Practice: Omit the default action

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



Exercise: Add index.html to your cookbooks files/default directory

Add the following to cookbooks/apache/files/default/index.html

```
1 <html>
2   <body>
3     <h1>Hello world</h1>
4   </body>
5 </html>
```

What's with the 'default' subdirectory?

- Chef allows you to select the most appropriate file (or template) within a cookbook according to the platform of the node it is being executed on
 - node name (foo.bar.com)
 - platform-version (redhat-6.2)
 - platform-major (redhat-6)
 - platform
 - default
- 98% of the time, you will just use **default**



Exercise: Upload the cookbook

```
$ knife cookbook upload apache
```

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

Exercise: Add the apache recipe to your test nodes run list

```
$ knife node edit target1.local
```

```
1▼ {  
2   "name": "target1.local",  
3   "chef_environment": "_default",  
4   "normal": {  
5     "company": "opscode",  
6     "tags": [  
7       ]  
8     ]  
9   },  
10  "run_list": [  
11    ]  
12  ]  
13}  
14
```

Exercise: Add the apache recipe to your test nodes run list

Add `recipe[apache]` to the `run_list`, **save** and **close**.

```
1  {
2    "name": "target1.local",
3    "chef_environment": "_default",
4    "normal": {
5      "company": "opscode",
6      "tags": [
7
8        ]
9      },
10   "run_list": [
11     "recipe[apache]"
12   ]
13 }
14
```

- Run lists specify what recipes or roles the node should run, along with the order they should be run in
- Run lists are represented by an array
- Recipes are specified by “**recipe[name]**”
- Roles are specified by “**role[name]**”

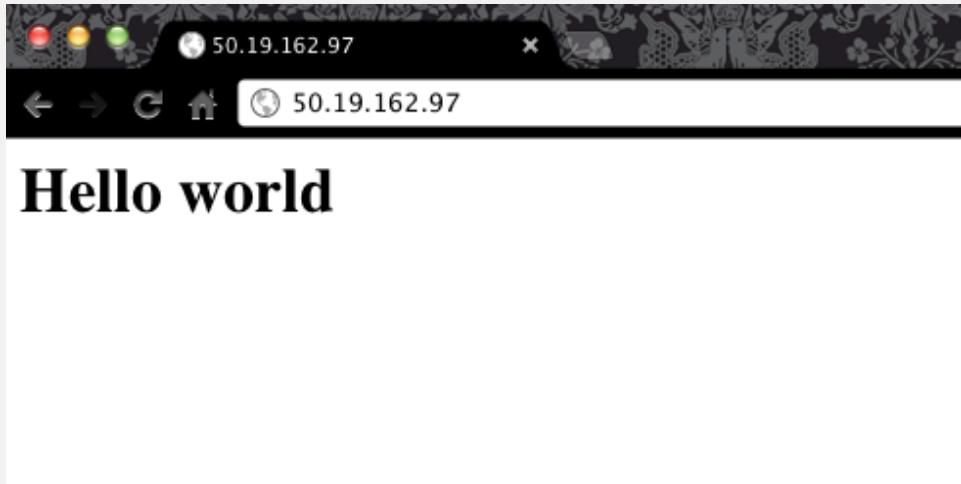


Exercise: Run the chef-client on your test node

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache]]
INFO: Run List expands to [apache]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache]
INFO: Storing updated cookbooks/apache/recipes/default.rb in the cache.
INFO: Storing updated cookbooks/apache/CHANGELOG.md in the cache.
INFO: Storing updated cookbooks/apache/metadata.rb in the cache.
INFO: Storing updated cookbooks/apache/README.md in the cache.
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: package[apache2] installed version 2.2.22-1ubuntu1
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 18)
INFO: cookbook_file[/var/www/index.html] backed up to /var/chef/backup/var/www/index.html.chef-20121023020345
INFO: cookbook_file[/var/www/index.html] owner changed to 0
INFO: cookbook_file[/var/www/index.html] group changed to 0
INFO: cookbook_file[/var/www/index.html] mode changed to 644
INFO: cookbook_file[/var/www/index.html] created file /var/www/index.html
INFO: Chef Run complete in 13.761588783 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

Exercise: Verify that the home page works



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



Congratulate yourself!

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



Reading the output of a chef-client run

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache]]
INFO: Run List expands to [apache]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
```

- We tell you the node's Run List
- The expanded Run List is the complete list,
after nested roles are expanded



Reading the output of a chef-client run

```
INFO: Loading cookbooks [apache]
INFO: Storing updated cookbooks/apache/recipes/default.rb in the
cache.
INFO: Storing updated cookbooks/apache/CHANGELOG.md in the cache.
INFO: Storing updated cookbooks/apache/metadata.rb in the cache.
INFO: Storing updated cookbooks/apache/README.md in the cache.
```

- We start loading the cookbooks in the order specified by the run list
- We download any files we are missing from the server



Reading the output of a chef-client run

```
INFO: Processing package[apache2] action install  
(apache::default line 10)  
INFO: package[apache2] installed version  
2.2.22-1ubuntu1
```

- We are checking to see if the package apache2 is installed
- It was not, and so we installed version 2.2.22-1ubuntu1 (yours may be different)



Reading the output of a chef-client run

```
INFO: Processing service[apache2] action start  
(apache::default line 14)  
INFO: Processing service[apache2] action enable  
(apache::default line 14)
```

- We check to see if apache2 is already started - and it is, so we do nothing
- We check to see if apache2 is already enabled to run at boot - and it is, so we do nothing

- 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



Reading the output of a chef-client run

```
INFO: Processing cookbook_file[/var/www/index.html] action
      create (apache::default line 18)
INFO: cookbook_file[/var/www/index.html] backed up to /var/
      chef/backup/var/www/index.html.chef-20121023020345
INFO: cookbook_file[/var/www/index.html] mode changed to 644
INFO: cookbook_file[/var/www/index.html] created file /var/
      www/index.html
```

- We check to see if we need to create the index.html file
- There is already one in place, whose contents are different than ours, so we back it up
- We also set the permissions appropriately



Reading the output of a chef-client run

```
INFO: Chef Run complete in 13.761588783 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

- We see Chef Run complete, with the time it took for the run to finish
- Report and exception handlers are now run



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache]]
INFO: Run List expands to [apache]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache]
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 18)
INFO: Chef Run complete in 1.216262701 seconds
INFO: Removing cookbooks/apache/files/default/index.html from the cache; it is no longer
needed by chef-client.
INFO: Running report handlers
INFO: Report handlers complete
```

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

Writing an MOTD Cookbook

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



RULE THE CLOUD



- Understand Cookbook Attribute files
- Learn how to use ERB Templates in Chef
- Understand Attribute Precedence
- Learn about Cookbook Metadata
- Know how to specify cookbook dependencies
- Practice the cookbook creation, upload, and test loop

The Problem and the Success Criteria

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

We have a small problem...

- We added a 'company' attribute earlier
- But we don't have an attribute that reflects whether we are in or out of scope for PCI Compliance

Well factored Cookbooks only contain the information relevant to their domain

- We could add a node attribute for PCI Compliance, but that will become very tiresome at scale
- We know we will likely have other things related to PCI (security settings, for example)
- The best thing to do is create a PCI cookbook, and add our attribute there



Exercise: Create a cookbook named 'pci'

```
$ knife cookbook create pci
```

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

Exercise: Create a default.rb attribute file in the PCI cookbook

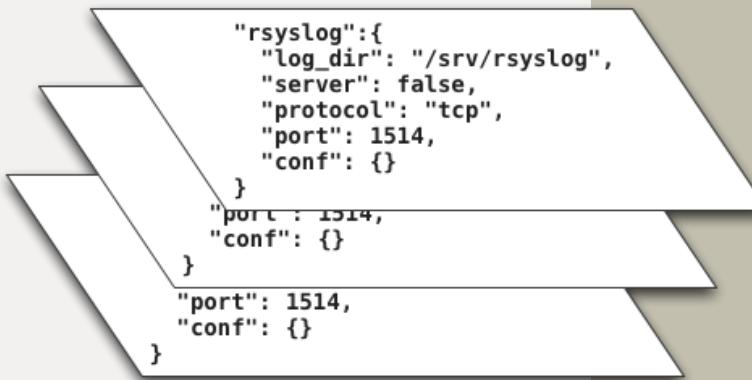
```
$ vi cookbooks/pci/attributes/default.rb
```

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

- Creates a new Node attribute: node['pci']['in_scope']
- Sets the value to the Ruby *false* literal

Node Attributes have four levels of precedence

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



Overrides
Normal
Default

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. :)



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**



Exercise: Upload the PCI cookbook

```
$ knife cookbook upload pci
```

```
Uploading pci  
Uploaded 1 cookbook.
```

```
[0.1.0]
```



Exercise: Create a cookbook named 'motd'

```
$ knife cookbook create motd
```

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



Exercise: Open the default recipe in your editor

```
$ vi cookbooks/motd/recipes/default.rb
```

```
1 #
2 # Cookbook Name:: motd
3 # Recipe:: default
4 #
5 # Copyright 2012, YOUR_COMPANY_NAME
6 #
7 # All rights reserved - Do Not Redistribute
8 #
```

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 /etc/motd.tail

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



The template[/etc/motd.tail] resource

```
[○○○] default.rb (~/src/sandbox/...cookbooks/motd/recipes) - VIM  
1 #  
2 # Cookbook Name:: motd  
3 # Recipe:: default  
4 #  
5 # Copyright 2012, YOUR_COMPANY_NAME  
6 #  
7 # All rights reserved - Do Not Redistribute  
8 #  
9  
10 template "/etc/motd.tail" do  
11   source "motd.tail.erb"  
12   mode "0644"  
13 end
```



Exercise: Open motd.tail.erb in your Editor

```
$ vi cookbooks/motd/templates/default/motd.tail.erb
```

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

~

- “erb” stands for “Embedded Ruby”



Exercise: Open motd.tail.erb in your Editor

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

~

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



Exercise: Open motd.tail.erb in your Editor

```
1 This server is property of <%= node['company'] %>
2 <% if node['pci']['in_scope'] -%>
3 This server is in-scope for PCI compliance
4 <% 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

Templates are what you will use for almost all your 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



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

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



Exercise: Upload the motd cookbook

```
$ knife cookbook upload motd
```

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

```
[0.1.0]
```

Add `recipe[motd]` to the `run_list`, **save** and **close**.

```
1 {
2   "name": "target1.local",
3   "chef_environment": "_default",
4   "normal": {
5     "company": "opscode",
6     "tags": [
7       ]
8   },
9   "run_list": [
10     "recipe[apache]",
11     "recipe[motd]"
12   ]
13 }
14 }
```

- **Do** add a comma after `recipe[apache]`
- **Don't** add a comma after `recipe[motd]`



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd]]
INFO: Run List expands to [apache, motd]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache, motd]
INFO: Storing updated cookbooks/recipes/.rb in the cache.
INFO: Storing updated cookbooks/CHANGELOG in the cache.
INFO: Storing updated cookbooks/motd/metadata.rb in the cache.
INFO: Storing updated cookbooks/motd/README.md in the cache.
INFO: Processing package[apache2] action install (apache2::default line 0)
INFO: Processing service[apache2] action start (apache2::default line 0)
INFO: Processing service[apache2] action stop (apache2::default line 0)
INFO: Processing cookbook_file[/var/www/html] action create (apache2::default line 18)
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)

=====
Error executing action `create` on resource: 'template[/etc/motd.tail']
```

FAIL!

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

Resource Declaration:
```



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

Resource Declaration:

```
# In /var/chef/cache/cookbooks/motd/recipes/default.rb

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

Compiled Resource:

```
# Declared in /var/chef/cache/cookbooks/motd/recipes/default.rb:10:in `from_file'

template("/etc/motd.tail") do
  provider Chef::Provider::Template
  action "create"
  retries 0
  retry_delay 2
  path "/etc/motd.tail"
  backup 5
  source "motd.tail.erb"
  cookbook_name "motd"
  recipe_name "default"
  mode "0644"
end

ERROR: Running exception handlers
FATAL: Saving node information to /var/chef/cache/failed-run-data.json
ERROR: Exception handlers complete
FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
FATAL: Chef::Mixin::Template::TemplateError: undefined method `[]' for nil:NilClass
```

- 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

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

```
=====
Error executing action `create` on resource 'template[/etc/motd.tail]'
=====
```

```
Chef::Mixin::Template::TemplateError
```

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

```
Resource Declaration:
```

```
# In /var/chef/cache/cookbooks/motd/recipes/default.rb  
  
10: template "/etc/motd.tail" do  
11:   source "motd.tail.erb"  
12:   mode "0644"  
13: end
```



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

```
INFO: Run List is [recipe[apache], recipe[motd]]  
INFO: Run List expands to [apache, motd]  
INFO: Starting Chef Run for target1.local  
INFO: Running start handlers  
INFO: Start handlers complete.  
INFO: Loading cookbooks [apache, motd]  
INFO: Storing updated cookbooks/motd/recipes/default.rb in the cache.  
INFO: Storing updated cookbooks/motd/metadata.rb in the cache.  
INFO: Storing updated cookbooks/motd/README.md
```

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



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

```
$ vi cookbooks/motd/metadata.rb
```

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

```
1 maintainer          "YOUR_COMPANY_NAME"
2 maintainer_email    "YOUR_EMAIL"
3 license             "All rights reserved"
4 description         "Installs/Configures motd"
5 long_description   IO.read(File.join(File.dirname(_
6 version             "0.1.0"
7 depends             "pci"
```

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

```
1 maintainer          "YOUR_COMPANY_NAME"
2 maintainer_email    "YOUR_EMAIL"
3 license             "All rights reserved"
4 description         "Installs/Configures motd"
5 long_description   IO.read(File.join(File.dirname(_
6 version            "0.1.0"
7 depends  "pci"
```

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



Cookbook Attributes are applied for all
downloaded cookbooks!

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



Exercise: Upload the motd cookbook

```
$ knife cookbook upload motd
```

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

```
[0.1.0]
```

Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd]]
INFO: Run List expands to [apache, motd]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbook apache, motd
INFO: Storing updated cookbook apache [pci/] /default in the cache.
INFO: Storing updated cookbook apache [pci/] /sites/default /erb in the apache cache.
INFO: Storing updated cookbook apache [pci/] /sites/default /erb /G.md in the apache cache.
INFO: Storing updated cookbook apache [pci/] /sites/default /erb /index.html in the apache cache.
INFO: Storing updated cookbook apache [pci/] /sites/default /erb /index.html /D in the apache cache.
INFO: Storing updated cookbook apache [pci/] /sites/default /erb /index.html /D /index in the apache cache.
INFO: Processing package[apache] action install (apache::default line 1)
INFO: Processing service[apache] action start (apache::default line 14)
INFO: Processing service[apache] action enable (apache::default line 14)
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 1)
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)
INFO: template[/etc/motd.tail] updated content
INFO: template[/etc/motd.tail] mode changed to 644
INFO: Chef Run complete in 1.760498119 seconds
INFO: Running report handlers
INFO: Report handlers complete
```





Exercise: Check your work

```
$ cat /etc/motd.tail
```

This server is property of opscode



Exercise: Show your test nodes pci attribute

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

```
1 items found
```

```
id: target1.local
pci:
  in_scope: false
```

Exercise: Set your test node to be in scope for PCI compliance

```
$ knife node edit target1.local
```

```
1  {
2    "name": "target1.local",
3    "chef_environment": "_default",
4    "normal": {
5      "company": "opscode",
6      "pci": {
7        "in_scope": true
8      },
9      "tags": [
10        ]
11    },
12    "run_list": [
13      "recipe[apache]",
14      "recipe[motd]"
15    ]
16  }
17}
18
```

- The node['pci']['in_scope'] attribute is represented as a JSON Hash
- Don't forget the **trailing comma**
- We are setting a **normal** attribute - so it will take precedence

Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd]]
INFO: Run List expands to [apache, motd]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache, motd, pci]
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 18)
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)
INFO: template[/etc/motd.tail] backed up to /var/chef/backup/etc/motd.tail.chef-20121023023657
INFO: template[/etc/motd.tail] updated content
INFO: template[/etc/motd.tail] mode changed to 644
INFO: Chef Run complete in 1.169403524 seconds
INFO: Running report handlers
INFO: Report handlers complete
```



Exercise: Check your work

```
$ cat /etc/motd.tail
```

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



Exercise: Show your test nodes pci attribute

```
$ knife node show target1.local -a pci
```

```
1 items found
```

```
id: target1.local
pci:
  in_scope: true
```

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

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

Refactoring the Apache Cookbook

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



RULE THE CLOUD



- Know how to use the Execute resource
- Understand how to control idempotence manually with not_if and only_if
- Learn about the Resources page on wiki.opscode.com
- Familiarize yourself with the Directory resource
- Understand how to use resource notifications
- Understand what Template Variables are, and how to use them
- Use Ruby variables, loops, and string expansion

The Problem and the Success Criteria

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



Exercise: Create a default.rb attribute file in the apache cookbook

```
$ vi cookbooks/apache/attributes/default.rb
```

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

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



Exercise: Open the default apache recipe in your editor

```
$ vi cookbooks/apache/recipes/default.rb
```

```
1 #  
2 # Cookbook Name:: apache  
3 # Recipe:: default  
4 #  
5 # Copyright 2012, YOUR_COMPANY_NAME  
6 #  
7 # All rights reserved - Do Not Redistribute  
8 #  
9  
10 package "apache2" do  
11   action :install  
12 end  
13  
14 service "apache2" do  
15   action [ :start, :enable ]  
16 end  
17  
18 cookbook_file "/var/www/index.html" do  
19   source "index.html"  
20   mode "0644"  
21 end
```



Exercise: Disable the default Apache virtual host with an Execute resource

```
14 service "apache2" do
15   action [ :start, :enable ]
16 end
17
18 execute "a2dissite default" do
19   only_if do
20     File.symlink?("/etc/apache2/sites-enabled/000-default")
21   end
22   notifies :restart, "service[apache2]"
23 end
24
25 cookbook_file "/var/www/index.html" do
```

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



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

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

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

- The Chef Wiki is the home for all of the documentation about Chef.
 - It is very comprehensive
 - It has a page on every topic
- <http://wiki.opscode.com>
- Let's use the wiki to learn more about **not_if** and **only_if**



Exercise: Search for more information about Resources

The screenshot shows the homepage of the Chef Wiki at wiki.opscode.com/display/chef/Home. The page title is "Home". On the left, there's a sidebar with links like "About", "Chef Basics", "Architecture", etc. The main content area features the Chef logo and the text: "Welcome to the Chef Wiki! Chef is a systems integration framework, built to bring the benefits of configuration management to your entire infrastructure." To the right, there's a "Quick Links" box with a list of links including "Installation", "Walkthrough Guides and Tutorials", "Chef Wiki As A PDF", etc. At the top right, there's a navigation bar with "Browse", "Log In", "Sign Up", and a search bar labeled "Search this space". An orange circle highlights the search bar.

- Search for “Resources”

The Resources Page

[Dashboard](#) > [Chef](#) > [Home](#) > [Chef Essentials](#) > [Resources and Providers](#) > [Resources](#)

[Browse](#) ▾ [Log In](#) [Sign Up](#)

 [Search this space](#)

- [About](#)
- [Chef Basics](#)
- [Architecture](#)
- [Chef Essentials](#)
- [Attributes](#)
- [Cookbooks](#)
- [Data Bags](#)
- [Environments](#)
- [Exception and Report Handlers](#)
- [Nodes](#)
- [Ohai](#)
- [Resources and Providers](#)
- [**Resources**](#)
- [Providers](#)
- [Lightweight Resources and Providers \(LWRP\)](#)
- [Opscode LWRP Resources](#)
- [Roles](#)
- [Search](#)
- [Installation](#)
- [Managing Chef](#)
- [Documentation](#)
- [Guides](#)
- [Developers](#)
- [Community Events](#)
- [Support](#)

Resources

 3 Added by [Adam Jacob](#), last edited by [Paul Mooring](#) on May 25, 2012 ([view change](#))



Overview

Resources are the basic units of work in Chef - discrete chunks of a system's configuration that are declared in [Recipes and Definitions](#), and applied to your nodes.

They are usually a cross platform abstraction of the thing you're configuring on the host.

For example, packages may be installed via apt, yum, or the BSD ports and packages systems, but the [package resource](#) abstracts these differences away so you can specify that a package should be installed in a cross-platform way.

This page provides documentation of the Resources included in the Chef library.

Each description is broken in to several categories:

Category	Description
Actions	The list of actions for this resource
Attributes	The list of attributes for this resource
Providers	The list of providers for this resource, along with any shortcut resource name. (You can use the shortcut resource name instead of specifying the provider attribute - <code>apt_package</code> rather than <code>package</code> with <code>provider Chef::Provider::Apt</code> .)
Examples	The first section, shows examples of using each action.

New custom resources can also be created using Chef's [Lightweight Resources and Providers \(LWRP\)](#) DSL. Opscode provides several cookbooks that define custom Lightweight Resources and Providers (LWRP). [Opscode LWRP Resources](#) details those customer LWRPs, grouped with the cookbook that provides them.

- Overview
- Meta
 - Common Actions
 - Common Attributes
 - Conditional Execution
 - Notifications
 - Relative Paths
- Cookbook File
- Cron
- Deploy
- Directory
- Env
- Erlang Call
- Execute
- File
- Git
- Group
- HTTP Request
- Ifconfig
- Link
- Log
- Mkdir
- Mount
- Ohai
- Package
- PowerShell Script
- Remote Directory
- Remote File
- Route
- Ruby Block
- SCM
- Script
- Service
- Subversion
- Template
- User
- Opscode Cookbook LWRPs

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

- Resource Notifications in Chef are used to trigger an action on a resource when the current resources actions are successful.
- “If we delete the site, restart apache”
- The first argument is an action, and the second argument is the string representation of a given resource
- Like `not_if` and `only_if`, **notifies** is a resource metaparameter - any resource can notify any other

Exercise: Iterate over each apache site

```
18 execute "a2dissite default" do
19   only_if do
20     File.symlink?("/etc/apache2/sites-enabled/000-default")
21   end
22   notifies :restart, "service[apache2]"
23 end
24
25 node['apache']['sites'].each do |site_name, site_data|
26   document_root = "/srv/apache/#{site_name}"
```

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

Exercise: Iterate over each apache site

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

- Calling .each loops over each site

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

~

- | | |
|---|---|
| <ul style="list-style-type: none">• First pass• site_name = 'clowns'• site_data = { "port" => 80 } | <ul style="list-style-type: none">• Second pass• site_name = 'bears'• site_data = { "port" => 81 } |
|---|---|



Exercise: Iterate over each apache site

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

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



Exercise: Add a template for the apache sites virtual host configuration

```
25 node['apache']['sites'].each do |site_name, site_data|
26   document_root = "/srv/apache/#{site_name}"
27
28   template "/etc/apache2/sites-available/#{site_name}" do
29     source "custom.erb"
30     mode "0644"
31     variables(
32       :document_root => document_root,
33       :port => site_data['port']
34     )
35     notifies :restart, "service[apache2]"
36   end
```

- 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



OPSCODE



Chef

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

```
28   template "/etc/apache2/sites-available/#{site_name}" do
29     source "custom.erb"
30     mode "0644"
31     variables(
32       :document_root => document_root,
33       :port => site_data['port']
34     )
35     notifies :restart, "service[apache2]"
36   end
37
38   execute "a2ensite #{site_name}" do
39     not_if do
40       File.symlink?("/etc/apache2/sites-enabled/#{site_name}")
41     end
42     notifies :restart, "service[apache2]"
43   end
```

Exercise: Add a directory resource that creates the document_root

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

```
38   execute "a2ensite #{site_name}" do
39     not_if do
40       File.symlink?("/etc/apache2/sites-enabled/#{site_name}")
41     end
42     notifies :restart, "service[apache2]"
43   end
44
45   directory document_root do
46     mode "0755"
47     recursive true
48   end
```



Exercise: Add a template resource to supply the index.html for the virtual host

```
45   directory document_root do
46     mode "0755"
47     recursive true
48   end
49
50   template "#{document_root}/index.html" do
51     source "index.html.erb"
52     variables(
53       :site_name => site_name,
54       :port => site_data['port']
55     )
56     mode "0644"
57   end
58 end
```



Don't forget the last "end"

```
50   template "#{document_root}/index.html" do
51     source "index.html.erb"
52     variables(
53       :site_name => site_name,
54       :port => site_data['port']
55     )
56     mode "0644"
57   end
58 end
```

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

Exercise: Add custom.erb to your templates directory

```
$ vi cookbooks/apache/templates/default/custom.erb
```

```
1 <% if @port != 80 -%>
2   Listen <%= @port %>
3 <% end -%>
4
5 <VirtualHost *:<%= @port %>>
6   ServerAdmin webmaster@localhost
7
8   DocumentRoot <%= @document_root %>
9   <Directory />
10    Options FollowSymLinks
11    AllowOverride None
12  </Directory>
13  <Directory <%= @document_root %>>
14    Options Indexes FollowSymLinks MultiViews
15    AllowOverride None
16    Order allow,deny
17    allow from all
18  </Directory>
19 </VirtualHost>
```

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

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

```
$ vi cookbooks/apache/templates/default/index.html.erb
```

```
1 <html>
2   <body>
3     <h1>Welcome to <%= node['company'] %></h1>
4     <h2>We love <%= @site_name %></h2>
5     <%= node['ipaddress'] %>:<%= @port %>
6   </body>
7 </html>
```

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

```
$ vi cookbooks/apache/metadata.rb
```

```
4 description      "Installs/
5 long_description IO.read(File
6 version          "0.2.0"
```

- Major, Minor, Patch



Exercise: Upload the apache cookbook

```
$ knife cookbook upload apache
```

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



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd]]
INFO: Run List expands to [apache, motd]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache, motd, pci]
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing execute[a2dissite default] action run (apache::default line 18)
Site default disabled.
To activate the new configuration, you need to run:
  service apache2 reload
INFO: execute[a2dissite default] ran successfully
```



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing template[/etc/motd.tail] action  
create (motd::default line 10)
```

```
INFO: template[/etc/apache2/sites-available/clowns]  
sending restart action to service[apache2] (delayed)
```

```
INFO: Processing service[apache2] action restart  
(apache::default line 14)
```

```
INFO: service[apache2] restarted
```

```
INFO: Chef Run complete in 5.352761142 seconds
```

```
INFO: Running report handlers
```

```
INFO: Report handlers complete
```



Exercise: Verify our two sites are working!

← → C ⌂ 50.19.162.97

Welcome to opscode

We love clowns

10.4.25.155:80

← → C ⌂ 50.19.162.97:81

Welcome to opscode

We love bears

10.4.25.155:81



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

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

- How do you control the idempotence of an Execute resource?
- Where can you learn the details about all the core resources in Chef?
- What is a notification?
- What is a template variable?
- What does #{foo} do in a Ruby string?

Writing a Users cookbook

Users, Groups, Data Bags, Recipe Inclusion and Search



- Know what Data Bags are, and how they are used
- Know how to use the User and Group resources
- Learn how to use `include_recipe`
- Understand the role Search plays in recipes
- Learn how to use Encrypted Data Bags
- Use nested hashes to iterate through data
- Use `knife ssh` to issue remote commands

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

- 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

- Via files in the `data_bag` directory of your `chef-repo` - **do this for anything you are managing by hand**
 - We are using this model in this section
- Created and managed directly via the Chef API by a custom script - **do this for anything you are integrating with**



Exercise: Create a data bag named users

```
$ mkdir data_bags/users  
$ knife data_bag create users
```

Created data_bag[users]

Exercise: Create a user item in the users data bag

```
$ vi ./data_bags/users/bobo.json
```

```
1 {  
2   "id": "bobo",  
3   "comment": "Bobo T. Clown",  
4   "uid": 2000,  
5   "gid": 0,  
6   "home": "/home/bobo",  
7   "shell": "/bin/bash"  
8 }
```



Exercise: Create the data bag item

```
$ knife data_bag from file users bobo.json
```

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

Exercise: Create a user item in the users data bag

```
$ vi ./data_bags/users/frank.json
```

```
1 {
2   "id": "frank",
3   "comment": "Frank Belson",
4   "uid": 2001,
5   "gid": 0,
6   "home": "/home/frank",
7   "shell": "/bin/bash"
8 }
```



Exercise: Create the data bag item

```
$ knife data_bag from file users frank.json
```

```
Updated data_bag_item[users::frank]
```



Exercise: Show all the items in users data bag

```
$ knife search users '*:*'
```

```
2 items found
```

```
chef_type: data_bag_item
comment: Frank Belson
data_bag: users
gid: 0
home: /home/frank
id: frank
shell: /bin/bash
uid: 2001
```

```
chef_type: data_bag_item
comment: Bobo T. Clown
data_bag: users
gid: 0
home: /home/bobo
id: bobo
shell: /bin/bash
uid: 2000
```



Exercise: Find Bobo's shell in Chef

```
$ knife search users 'comment:"Bobo T. Clown"' -a shell
```

```
1 items found
```

```
id:      bobo
shell:   /bin/bash
```



Exercise: Create a data bag named groups

```
$ mkdir data_bags/groups  
$ knife data_bag create groups
```

Created data_bag[groups]



Exercise: Create a group item in the group data bag

```
$ vi ./data_bags/groups/clowns.json
```

```
1  {
2    "id": "clowns",
3    "gid": 3000,
4    "members": [
5      "bobo",
6      "frank"
7    ]
8 }
```



Exercise: Create the data bag item

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

```
Updated data_bag_item[groups::clowns]
```



Exercise: Show all the groups in Chef

```
$ knife search groups '*:*'
```

```
1 items found
```

```
chef_type:    data_bag_item
data_bag:      groups
gid:          3000
id:           clowns
members:
  bobo
  frank
```



Exercise: Create a cookbook named 'users'

```
$ 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

```
$ vi cookbooks/users/recipes/default.rb
```

```
1 #
2 # Cookbook Name:: users
3 # Recipe:: default
4 #
5 # Copyright 2012, YOUR_COMPANY_NAME
6 #
7 # All rights reserved - Do Not Redistribute
8 #
```

Exercise: Open the default recipe in your editor

```
7 # All rights reserved - Do Not Redistribute
8 #
9
10 search(:users, "*:*").each do |user_data|
11   user user_data['id'] do
12     comment user_data['comment']
13     uid user_data['uid']
14     gid user_data['gid']
15     home user_data['home']
16     shell user_data['shell']
17   end
18 end
19
20 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 Wiki for information about the `user` resource

Exercise: Open the default recipe in your editor

```
7 # All rights reserved - Do Not Redistribute
8 #
9
10 search(:users, "*:*").each do |user_data|
11   user user_data['id'] do
12     comment user_data['comment']
13     uid user_data['uid']
14     gid user_data['gid']
15     home user_data['home']
16     shell user_data['shell']
17   end
18 end
19
20 include_recipe "users::groups"
```

- **include_recipe** ensures another recipes resources are complete before we continue
- Chef will only include each recipe once
- **include_attribute** does the same, but for attribute files



Best Practice: Use `include_recipe` and `include_attribute` liberally

- If there is a pre-requisite for your recipe that resides in another recipe (the JVM existing for your Java application, for example)
 - Always use `include_recipe` to include it specifically, even if you put it in a run list
- The same goes for `include_attribute`

```
$ vi cookbooks/users/recipes/groups.rb
```

```
1 search(:groups, "*:*").each do |group_data|
2   group group_data['id'] do
3     gid group_data['gid']
4     members group_data['members']
5   end
6 end
```

- This file follows the same pattern as the default users recipe
- Use the Chef Wiki for information about the **group** resource



Exercise: Upload the users cookbook

```
$ knife cookbook upload users
```

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

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

```
1 {  
2   "name": "target1.local",  
3   "chef_environment": "_default",  
4   "normal": {  
5     "company": "opscode",  
6     "pci": {  
7       "in_scope": true  
8     },  
9     "tags": [  
10       ],  
11     "apache": {  
12       "sites": {  
13         }  
14     }  
15   },  
16 },  
17   "run_list": [  
18     "recipe[apache]",  
19     "recipe[motd]",  
20     "recipe[users]"  
21   ]  
22 }
```

Add `recipe[users]` to the `run_list`, **save** and **close**.

- **Do** add a comma after `recipe[motd]`
- **Don't** add a comma after `recipe[users]`



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd], recipe[users]]
INFO: Run List expands to [apache, motd, users]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache, motd, pci, users]
INFO: Storing updated cookbooks/users/recipes/default.rb in the cache.
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing execute[a2dissite default] action run (apache::default line 18)
INFO: Processing template[/etc/apache2/sites-available/clowns] action create (apache::default line 28)
INFO: Processing execute[a2ensite clowns] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/clowns] action create (apache::default line 45)
INFO: Processing template[/srv/apache/clowns/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/apache2/sites-available/bears] action create (apache::default line 28)
INFO: Processing execute[a2ensite bears] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/bears] action create (apache::default line 45)
INFO: Processing template[/srv/apache/bears/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)
INFO: Processing user[bobo] action create (users::default line 11)
INFO: user[bobo] created
INFO: Processing user[frank] action create (users::default line 11)
INFO: user[frank] created
INFO: Processing group[clowns] action create (users::groups line 2)
INFO: group[clowns] created
INFO: Chef Run complete in 4.204686792 seconds
INFO: Running report handlers
INFO: Report handlers complete
```



Exercise: Verify the users and groups exist

```
$ sudo cat /etc/passwd
```

```
frank:x:2001:0:Frank Belson:/home/frank:/bin/bash
bobo:x:2000:0:Bobo T. Clown:/home/bobo:/bin/bash
```

```
$ sudo cat /etc/group
```

```
clowns:x:3000:bobo,frank
```

- We just created a centralized user and group repository, from scratch
 - (That's kind of like what LDAP and Active Directory do, only they are, well, fancier.)
- Between Data Bags and Node Attribute precedence, Chef provides a plethora of ways to inform the patterns you use to configure your infrastructure

But wait... what about passwords?

- Sometimes you may need to store sensitive information inside a data bag
- Encrypted Data Bag items allow sensitive information, such as user passwords, to be stored in an encrypted form
- This protects sensitive data from exposure if your Chef Server is compromised



Exercise: Create a data bag named passwords

```
$ mkdir data_bags/passwords  
$ knife data_bag create passwords
```

Created data_bag[users]



Exercise: Create a new data bag secret key

```
$ openssl rand -base64 512 | tr -d '\r\n' > secret_key
```

(Has no output)



Exercise: Create password hashes for your users

```
$ openssl passwd -1 "bobosplaintextpassword"  
$ openssl passwd -1 "franksplaintextpassword"
```

```
$1$yEYRpAp8$CgiRRoaX1Y/1eKB5/emss0  
$1$bEfGICWD$yvqNbdLnEj6A36ZTCTvlN0
```



Exercise: Create a userlist item in the password data bag

```
$ vi ./data_bags/passwords/userlist.json
```

```
1 {
2   "id": "userlist",
3   "bobo": "$1$yEYRpAp8$CgiRRoaX1Y/1eKB5/emss0",
4   "frank": "$1$bEfG1CWD$yvqNbdLnEj6A36ZTCTv1N0"
5 }
```



Exercise: Create the data bag item

```
$ knife data bag from file password userlist.json \
--secret-file secret_key
```

Updated data_bag_item[passwords::userlist]



Exercise: Show all entries in the userlist item of the passwords data bag

```
$ knife data_bag show passwords userlist
```

```
id:      userlist
bobo:    ppYT0lWdCzXnZ0G5tuyIwNwTTeJR3FrsMB9X4jPPq{...}
frank:   eiFzA0QdxI7ADkdBD+11SMvYmfwhnDQPSXiBBvwee{...}
```



Exercise: Show all decrypted entries in the userlist item

```
$ knife data_bag show passwords userlist --secret-file secret_key
```

```
id:      userlist
bobo:    $1$yEYRpAp8$CgiRRoaX1Y/1eKB5/emss0
frank:   $1$bEfGlcWD$yvqNbdLnEj6A36ZTCTvlN0
```



Exercise: Open the default recipe in your editor

```
$ vi cookbooks/users/recipes/default.rb
```

```
7 # All rights reserved - Do Not Redistribute
8 #
9
10 decrypted = Chef::EncryptedDataBagItem.load("passwords", "userlist")
11
12 search(:users, "*:*").each do |user_data|
13   user user_data['id'] do
```

- We use the same show we just tried with Knife to load the encrypted data_bag item
- All hashes in the item are bound to local variable decrypted



Exercise: Open the default recipe in your editor

```
10 decrypted = Chef::EncryptedDataBagItem.load("user_data")
11
12 search(:users, "*:*").each do |user_data|
13   user user_data['id'] do
14     comment user_data['comment']
15     uid user_data['uid']
16     gid user_data['gid']
17     home user_data['home']
18     shell user_data['shell']
19     password decrypted[user_data['id']]
20   end
21 end
22
23 include_recipe "users::groups"
```

- The “password” attribute supports adding shadow hash passwords

Iterating over data_bag items

```
7 # All rights reserved - Do Not Redistribute
8 #
9
10 decrypted = Chef::EncryptedDataBagItem.load("passwords", "userlist")
11
12 search(:users, "*:*").each do |user_data|
13   user user_data['id'] do
```

```
"id": "userlist",
"bobo": "$1$yEYRpAp8$CgiRRoaX1Y/1eKB5/emss0",
"frank": "$1$bEfG1CWD$yvqNbdLnEj6A36ZTCTv1N0"
```

- Bobo's password is stored as decrypted['bobo']
- Frank's password is stored as decrypted['frank']

Iterating over data_bag items

```
12 search(:users, "*:*").each do |user_data|
13   user user_data['id'] do
14     comment user_data['comment']
15     uid user_data['uid']
16     gid user_data['gid']
17     home user_data['home']
18     shell user_data['shell']
19     password decrypted[user_data['id']]
20   end
21 end
```

2 items found

chef_type:	data_bag_item
comment:	Frank Belson
data_bag:	users
gid:	0
home:	/home/frank
id:	frank
shell:	/bin/bash
uid:	2001

chef_type:	data_bag_item
comment:	Bobo T. Clown
data_bag:	users
gid:	0
home:	/home/bobo
id:	bobo
shell:	/bin/bash
uid:	2000

- First pass
- user_data['id'] = 'bobo'
- password = decrypted['bobo']
- Second pass
- user_data['id'] = 'frank'
- password = decrypted['frank']



Exercise: Upload the users cookbook

```
$ knife cookbook upload users
```

```
Uploading users  
Uploaded 1 cookbook.
```

```
[0.1.0]
```



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd], recipe[users]]
INFO: Run List expands to [apache, motd, users]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete
INFO: Loading cookbooks [apache, motd, users]
INFO: Storing updated cookbooks/users/default/default.rb
=====
Recipe Compile Error --- /var/chef/cache/cookbooks/users/recipes/default.rb:10:in `from_file'
=====
Errno::ENOENT
No such file or directory - file not found '/etc/chef/encrypted_data_bag_secret'

Cookbook Trace:
-----
/var/chef/cache/cookbooks/users/recipes/default.rb:10:in `from_file'
```



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

```
=====
Recipe Compile Error in /var/chef/cache/cookbooks/users/
recipes/default.rb
=====
```

```
Errno::ENOENT
=====
```

```
No such file or directory - file not found '/etc/chef/
encrypted_data_bag_secret'
```



Exercise: Copy the data bag secret key to the target node

```
$ cat secret_key  
$ knife ssh "name:target1.local" -a ipaddress -x ubuntu "echo  
<secret_key> | sudo tee /etc/chef/encrypted_data_bag_secret"
```

50.19.162.97

Quq7b95KjP0CDoVlcNpkKpu9oH00zcdkqE3Kvlq1FuUSFYBJFnjnfHgt9t0Ze5NSo0XSL
f7c4T1qgAptFsNcowAdGhaFa2mLWVJZkmeQj9cad sq7ToiJdC5XGQ3r4dl9bdmvuNcCnd
5ltb2NXZiUDwPwVn1EhXZBJ1nIjbewBCk05j1hPxMxFhWBXi uNUJxiPwlFdiUjMqr3XSu
FSAUlaqm3fb9wo0z2WkhtFR8azLBcmGIGdgQPGkL0XwvRX14nBMVZNHdVcrRAguo z3Ilw
MMHiG0Eozd0xeiadXk8cVcQ8M2d96nNGAU3bVZR2CpYxcSjocj5RyXGFxM2dJ0XzE4TJu
NXExXEzRNmKrvahHepKnh4dhLCtcUhZYmyNr8UqB987yaAFv61JyZnNS8RNWVb7ZNCZdQ
Bpd15WI AotPmreEPkwbksjmP5RvLI7MHA0f7JMKWYvAVVbSb4JjvK8mNSPtKIwmPvYaNx
33qnNBRJmA7uwgoJ1gY6h0HnnBP1k00t8kmUTtcfCpBCvbg47sTQmvk9oHxZEDh0kpxp4
qMPIPd6bd fMqWr178CinRBDNpTv19Z5XBngWA5oFn1Jcu2vXz8YLaMcDechj39yB4kbwU
rDZD9HpZrS0hd mBdHLGqi03wE8Df5oTCi57eGXHoHLHV1Xl1pLrZNYHRdJhIV4=

A few ways to specify your decryption key

- If no key is specified, Chef will look in `Chef::Config[:encrypted_data_bag_secret]`
- By default, that is set to `/etc/chef/encrypted_data_bag_secret`

```
10 decrypted = Chef::EncryptedDataBagItem.load("passwords", "userlist", contents_of_secret_key)
11
```

- Specify the key in your recipe

```
10 my_secret = Chef::EncryptedDataBagItem.load_secret("/path/to/other/secret_key")
11 decrypted = Chef::EncryptedDataBagItem.load("passwords", "userlist", my_secret)
```

- Load a different secret key and use its contents



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [recipe[apache], recipe[motd], recipe[users]]
INFO: Run List expands to [apache, motd, users]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache, motd, pci, users]
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing execute[a2dissite default] action run (apache::default line 18)
INFO: Processing template[/etc/apache2/sites-available/clowns] action create (apache::default line 28)
INFO: Processing execute[a2ensite clowns] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/clowns] action create (apache::default line 45)
INFO: Processing template[/srv/apache/clowns/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/apache2/sites-available/bears] action create (apache::default line 28)
INFO: Processing execute[a2ensite bears] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/bears] action create (apache::default line 45)
INFO: Processing template[/srv/apache/bears/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)

INFO: Processing user[bobo] action create (users::default line 13)
INFO: user[bobo] altered
INFO: Processing user[frank] action create (users::default line 13)
INFO: user[frank] altered
INFO: Processing group[clowns] action create (users::groups line 2)
INFO: Chef Run complete in 1.915067412 seconds
INFO: Running report handlers
INFO: Report handlers complete
```



Exercise: Verify the user passwords are correct

```
$ sudo cat /etc/shadow
```

```
frank:$1$buWwEiTZ$Fmk1E2/utBCRj9l5uSufx1:15634:0:99999:7:::  
bobo:$1$B1wv0KDX$t3WXdeILrP7W7Zkz5QeIG.:15634:0:99999:7:::
```

```
$ su - bobo
```

Password:

No directory, logging in with HOME=/
bobo@target1:/\$

- What are Data Bags?
- How are they used?
- What does the User resource do?
- What is include_recipe, and why is it useful?
- How does search work inside a recipe?
- What other applications do you see for search?
- How could we have used Data Bags in the refactored Apache recipe?
- Where would you go to find out more?

Roles

Role-based Attributes and Merge Order
Precedence



- Know what Roles are, and how they are used to provide clarity
- Be familiar with the Role Ruby DSL
- Know how to show a Role with Knife
- Understand how merge order effects the precedence hierarchy
- Understand nested Roles

- 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



Exercise: Create the webserver role

```
$ vi ./roles/webserver.rb
```

```
1 name "webserver"
2 description "Web Server"
3 run_list "recipe[apache]"
4 default_attributes({
5   "apache" => {
6     "sites" => {
7       "admin" => {
8         "port" => 82
9       }
10    }
11  }
12 })
```

- Roles have **names**
- Roles have **descriptions**
- Roles have a **run_list**, just like a node



Exercise: Create the webserver role

```
$ vi ./roles/webserver.rb
```

```
1 name "webserver"
2 description "Web Server"
3 run_list "recipe[apache]"
4 default_attributes({
5   "apache" => {
6     "sites" => {
7       "admin" => {
8         "port" => 82
9       }
10      }
11    }
12 })
```

- You can set **default** node attributes within roles!
- Save and Quit your editor



Exercise: Create the role

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

Updated Role webserver!



Exercise: Show the role with knife

```
$ knife role show webserver
```

```
chef_type:          role
default_attributes:
  apache:
    sites:
      admin:
        port:  82
description:        Web Server
env_run_lists:      {}
json_class:         Chef::Role
name:               webserver
override_attributes: {}
run_list:           ["recipe[apache]"]
```



Exercise: Search for the roles that have recipe[apache] in their run list

```
$ knife search role 'run_list:recipe\[apache\]'
```

```
1 items found
```

```
chef_type:          role
default_attributes:
  apache:
    sites:
      admin:
        port:  82
  description:      Web Server
  env_run_lists:   {}
  json_class:      Chef::Role
  name:            webserver
  override_attributes: {}
  run_list:         ["recipe[apache]"]
```



Exercise: Replace `recipe[apache]` with `role[webserver]` in your test nodes run list

```
1 {  
2     "name": "target1.local",  
3     "chef_environment": "_default",  
4     "normal": {  
5         "company": "opscode",  
6         "pci": {  
7             "in_scope": true  
8         },  
9         "tags": [  
10            ],  
11            "apache": {  
12                "sites": {  
13                    }  
14                }  
15            }  
16        },  
17        "run_list": [  
18            "role[webserver]",  
19            "recipe[motd]",  
20            "recipe[users]"  
21        ]  
22    }
```

Replace `recipe[apache]` with `role[webserver]`, **save** and **close**.



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [role[webserver], recipe[motd],
recipe[users]]
INFO: Run List expands to [apache, motd, users]
```



Exercise: Re-run the Chef Client

```
INFO: Processing template[/etc/apache2/sites-available/clowns] action create (apache::default line 28)
INFO: Processing execute[a2ensite clowns] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/clowns] action create (apache::default line 45)
INFO: Processing template[/srv/apache/clowns/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/apache2/sites-available/bears] action create (apache::default line 28)
INFO: Processing execute[a2ensite bears] action run (apache::default line 38)
INFO: Processing directory[/srv/apache/bears] action create (apache::default line 45)
INFO: Processing template[/srv/apache/bears/index.html] action create (apache::default line 50)
INFO: Processing template[/etc/apache2/sites-available/admin] action create (apache::default line 28)
INFO: template[/etc/apache2/sites-available/admin] updated content
INFO: template[/etc/apache2/sites-available/admin] mode changed to 644
INFO: Processing execute[a2ensite admin] action run (apache::default line 38)
Enabling site admin.
To activate the new configuration, you need to run:
  service apache2 reload
INFO: execute[a2ensite admin] ran successfully
INFO: execute[a2ensite admin] not queuing delayed action restart on service[apache2] (delayed), as it's already
been queued
INFO: Processing directory[/srv/apache/admin] action create (apache::default line 45)
INFO: directory[/srv/apache/admin] created directory /srv/apache/admin
INFO: directory[/srv/apache/admin] mode changed to 755
INFO: Processing template[/srv/apache/admin/index.html] action create (apache::default line 50)
INFO: template[/srv/apache/admin/index.html] updated content
INFO: template[/srv/apache/admin/index.html] mode changed to 644
```

Node Attributes that are hashes are merged

- The apache cookbooks attribute file contains:

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

- While our role has...

```
4 default_attributes({
5   "apache" => {
6     "sites" => {
7       "admin" => {
8         "port" => 82
9       },
10      }
11    }
12 })
```



Exercise: Search for the apache.sites attribute on all nodes with the role webserver

```
$ knife search node 'role:webserver' -a apache.sites
```

```
1 items found
```

```
apache.sites:
```

```
  admin:
```

```
    port: 82
```

```
  bears:
```

```
    port: 81
```

```
  clowns:
```

```
    port: 80
```

```
  id:          target1.local
```

Exercise: Edit the webserver role

```
$ vi ./roles/webserver.rb
```

```
4 default_attributes({  
5   "apache" => {  
6     "sites" => {  
7       "admin" => {  
8         "port" => 82  
9       },  
10      "bears" => {  
11        "port" => 8081  
12      }  
13    }  
14  }  
15 })
```

- Do not forget the **comma** after the admin site
- Change the value of the bears site to be 8081



Exercise: Create the role

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

Updated Role webserver!



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
[2012-10-23T03:28:34+00:00] INFO: Processing template[/etc/apache2/sites-available/bears] action create (apache::default line 28)
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] backed up to /var/chef/backup/etc/apache2/sites-available/bears.chef-20121023032834
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] updated content
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] owner changed to 0
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] group changed to 0
[2012-10-23T03:28:34+00:00] INFO: template[/etc/apache2/sites-available/bears] mode changed to 644
[2012-10-23T03:28:34+00:00] INFO: Processing execute[a2ensite bears] action run (apache::default line 38)
[2012-10-23T03:28:34+00:00] INFO: Processing directory[/srv/apache/bears] action create (apache::default line 45)
[2012-10-23T03:28:34+00:00] INFO: Processing template[/srv/apache/bears/index.html] action create (apache::default line 50)
[2012-10-23T03:28:34+00:00] INFO: template[/srv/apache/bears/index.html] backed up to /var/chef/backup/srv/apache/bears/index.html.chef-20121023032834
[2012-10-23T03:28:34+00:00] INFO: template[/srv/apache/bears/index.html] updated content
[2012-10-23T03:28:34+00:00] INFO: template[/srv/apache/bears/index.html] owner changed to 0
[2012-10-23T03:28:34+00:00] INFO: template[/srv/apache/bears/index.html] group changed to 0
[2012-10-23T03:28:34+00:00] INFO: template[/srv/apache/bears/index.html] mode changed to 644
```



Exercise: Search for the apache.sites attribute on all nodes with the role webserver

```
$ knife search node 'role:webserver' -a apache.sites
```

```
1 items found
```

```
apache.sites:
```

```
  admin:
```

```
    port: 82
```

```
  bears:
```

```
    port: 8081
```

```
  clowns:
```

```
    port: 80
```

```
  id:          target1.local
```

When you combine precedence and merge order, you get the complete picture of node attribute setting

	Attribute Files	Environment	Role	Node/Recipe
Default	1	2	3	4
Normal	5			6
Override	7	9	8	10
Automatic			11	

Best Practice: Roles get default attributes

- While it is awesome that you can use overrides, in practice there is little need
- If you always set **default** node attributes in your cookbook attribute files
- You can almost **always** set default node attributes in your role, and let merge order do the rest

Best Practice: Be careful when computing a value in an attribute file based on non-automatic data

- Because we merge data as we evaluate the Chef run, if you have something like the following in an attribute file:

```
1 default[:sneaky_port] = node['apache']['sites']['bears']['port'] * 2
```

- You will compute the value based on only what was available when the attribute file was parsed - so in our example, port 81 rather than the 8081 we just set it to.
- If you need to do this kind of computation, do it **late** in the merge order - usually in a recipe



Best Practice: Have “base” roles

- In addition to obvious roles, such as “webserver”, it is a common practice to group any functionality that “goes together” in a role
- The most common example here is a **base** role, where you include all the recipes that should be run on every node



Exercise: Create the base role

```
$ vi ./roles/base.rb
```

```
1 name "base"
2 description "Base Server Role"
3 run_list "recipe[motd]", "recipe[users]"
```

- **Save and Quit** your editor



Exercise: Create the role

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

Updated Role base!

```
$ vi ./roles/webserver.rb
```

```
1 name "webserver"
2 description "Web Server"
3 run_list "role[base]", "recipe[apache]"
4 default_attributes({
5   "apache" => {
```

- Put `role[base]` at the front of the `run_list`
- **Save and Quit** your editor



Exercise: Update the role

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

Updated Role webserver!



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [role[webserver], recipe[motd],
recipe[users]]
INFO: Run List expands to [motd, users, apache]
```



Best Practice: Be explicit about what you need or expect

- Chef will only execute a recipe the first time it appears in the run list
- So be explicit about your needs and expectations - either by nesting roles or using `include_recipe`



Exercise: Set the run list to just role[webserver]

Remove all the entries in the run list other than role[webserver], then **save and close**.

- What is a Role?
- What makes for a “good” role?
- How do you search for roles with a given recipe in their run list?
- How many times will Chef execute a recipe in the same run?

Environments

Cookbook Version Constraints, Override Attributes, and Per Environment Run Lists



RULE THE CLOUD



- Understand what an Environment is, and how it is different from an Organization
- Know how to set cookbook version constraints
- Understand when to set attributes in an environment
- Know how to leverage per-environment run lists in a role

- Environments allow you to:
 - Control what cookbook versions are used by a group of nodes
 - Set attributes that are relevant
- Typical environments would be
 - Dev
 - Test
 - Production



Exercise: Use knife to show the available cookbook versions

```
$ knife cookbook show apache
```

```
apache 0.2.0 0.1.0
```



Best Practice: If you need to share Cookbooks or Roles, use an Environment

- Environments allow for segregation within a single organization.
- If you need to share cookbooks or roles, you likely want an Environment rather than an organization.



Best Practice: Environments live in your chef-repo, until they don't

- Like Data Bags and Roles, you have options with how to create an Environment
- If you are managing them **manually**
 - The best practice is that all of your Environments live in the environments directory of your chef-repo
- If you are managing them **automatically**
 - Create all of your Environments with the automation, or with Knife directly



Exercise: List the current environments

```
$ knife environment list
```

```
_default
```

- The `_default` environment is read-only, and sets no policy at all



Exercise: Create a dev environment

```
$ vi ./environments/dev.rb
```

```
1 name "dev"  
2 description "For developers!"  
3 cookbook "apache", "= 0.2.0"
```

- Environments have **names**
- Environments have a **description**
- Environments *can* have one or more **cookbook** constraints

- = Equal to
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- ~> Approximately greater than
- Learn more at [http://wiki.opscode.com/
display/chef/Version+Constraints](http://wiki.opscode.com/display/chef/Version+Constraints)



Best Practice: Use only the = operator

- When you need more complex version constraints, you'll be glad they are there
- But do not be tempted to build complicated schemes tying cookbook development branches to complex version constraints
 - That way leads madness :)



Exercise: Create the dev environment

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

Updated Environment dev



Exercise: Show your Chef dev environment

```
$ knife environment show dev
```

```
chef_type:          environment
cookbook_versions:
  apache:    = 0.2.0
default_attributes:  {}
description:        For developers!
json_class:         Chef::Environment
name:               dev
override_attributes: {}
```



Exercise: Change your nodes chef_environment to “dev”

```
1 {
2   "name": "target1.local",
3   "chef_environment": "dev",
4   "normal": {
5     "company": "opscode",
6     "pci": {
7       "in_scope": true
8     },
9     "tags": [
10    ],
11    "apache": {
12      "sites": {
13        }
14      }
15    },
16  },
17  "run_list": [
18    "role[webserver]"
19  ]
20 }
```

Replace `_default` with `dev`
Delete the `pci` section (in red)



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Chef Run complete in 1.587776095 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

Exercise: Create a production environment

```
$ vi ./environments/production.rb
```

```
1 name "production"
2 description "For Prods!"
3 cookbook "apache", "= 0.1.0"
4 override_attributes({
5   "pci" => {
6     "in_scope" => true
7   }
8 })
```

- Make sure the apache cookbook is set to version 0.0.1
- Set an override attribute for being in scope - no matter what, you are in scope



Exercise: Create the production environment

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

Updated Environment production



Exercise: Change your nodes chef_environment to “production”

```
1 {
2   "name": "target1.local",
3   "chef_environment": "production",
4   "normal": {
5     "company": "opscode",
6     "tags": [
7       ],
8     "apache": {
9       "sites": {
10         }
11       }
12     },
13   },
14   "run_list": [
15     "role[webserver]"
16   ]
17 }
```

Replace dev with production



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [role[webserver], recipe[motd], recipe[users]]
INFO: Run List expands to [motd, users, apache]
INFO: Starting Chef Run for target1.local
INFO: Running start handlers
INFO: Start handlers complete.
INFO: Loading cookbooks [apache, motd, pci, users]
INFO: Storing updated cookbooks/apache/recipes/default.rb in the cache.
INFO: Storing updated cookbooks/apache/metadata.rb in the cache.
INFO: Processing template[/etc/motd.tail] action create (motd::default line 10)
INFO: Processing user[bobo] action create (users::default line 13)
INFO: Processing user[frank] action create (users::default line 13)
INFO: Processing group[circus] action create (users::groups line 2)
INFO: Processing package[apache2] action install (apache::default line 10)
INFO: Processing service[apache2] action start (apache::default line 14)
INFO: Processing service[apache2] action enable (apache::default line 14)
INFO: Processing cookbook_file[/var/www/index.html] action create (apache::default line 18)
INFO: Chef Run complete in 2.187502202 seconds
INFO: Removing cookbooks/apache/CHANGELOG.md from the cache; it is no longer needed by chef-client.
INFO: Removing cookbooks/apache/templates/default/custom.erb from the cache; it is no longer needed by chef-client.
INFO: Removing cookbooks/apache/templates/default/index.html.erb from the cache; it is no longer needed by chef-client.
INFO: Running report handlers
INFO: Report handlers complete
```

- We just rolled back to an earlier version of the apache cookbook
- While the recipe applied fine, investigating the system will reveal Apache is still configured as it was in the 0.2.0 cookbook
- Chef is not magic - it manages state for declared resources. A better way to ensure a smooth rollback: write contra-resources to clean up, and have a new version of the cookbook.



Exercise: Update production to use version 0.2.0 of the apache cookbook

```
$ vi ./environment/production.rb
```

- Make sure the apache cookbook
is set to version 0.2.0



Exercise: Upload the updated production environment

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

Updated Environment production



Security Policy Change!

- Nodes in the production environment should not have users created
- How can we do that without removing the users recipe from our base role?

Exercise: Add an environment specific run list for production to the base role

```
$ vi ./roles/base.rb
```

```
1 name "base"
2 description "Base Server Role"
3 def_run_list = [ "recipe[motd]", "recipe[users]" ]
4 env_run_lists({
5   "_default" => def_run_list,
6   "dev" => def_run_list,
7   "production" => [
8     "recipe[motd]"
9   ]
10 })
```

- Save and Quit your editor

- Delete the run_list



Exercise: Create the role

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

Updated Role base!



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: *** Chef 10.16.0 ***
INFO: Run List is [role[webserver]]
INFO: Run List expands to [motd, apache]
```

- What is an Environment?
- How is it different from an Organization?
- What is a cookbook version constraint?
- Which cookbook constraint should we probably not be using?
- What kind of node attributes do you typically set from an Environment?
- What is a per-environment run list?

Using Community Cookbooks



- Use the Opscode Chef Community site to find, preview and download cookbooks.
- Use knife to work with the Community Site API.
- Download, extract, examine and implement cookbooks from the Community site.



The easy way...

We've been writing some cookbooks so far...

Hundreds already exist for a large number of use cases and purposes. Many (but only a fraction) are maintained by Opscode.

Think of it like RubyGems.org, CPAN.org, or other focused plugin-style distribution sites.

RULE THE CLOUD

Exercise: Find and preview cookbooks on the site

<http://community.opscode.com/>



The screenshot shows the Opscode Community website at <http://community.opscode.com/>. A large orange arrow points from the text "Exercise: Find and preview cookbooks on the site" to the "Cookbooks!" button on the top navigation bar. The "Cookbooks!" button is highlighted with a dashed orange border.

The page features a dark header with the Opscode logo and Chef logo. The main navigation includes "Community Login", "Sign Up", "Recover Password", social media links for Facebook and Twitter, and a "Cookbooks" link in the top right. Below the header, there's a search bar with "Search Cookbooks" and "Advanced Search" options. The main content area includes sections for "Chef Documentation" (with a Chef logo graphic), "Blog Posts" (listing "Here at Opscode we have customers using Chef to implement some amazing use cases. One ... Read more"), and "Virtualization makes it easier and more cost-effective to scale physical environments. But it doesn't do ... Read more". Navigation tabs at the bottom include "Download & Install", "How to", "Chef", and "Ways to".



Exercise: Search for a chef-client cookbook

The screenshot shows the Opscode Community website at <http://community.opscode.com/cookbooks>. A large orange arrow points from the text "Search for: chef-client" to the search bar. The search results page displays various cookbooks, including mysql, apache2, and nginx under the All Categories section, and cookbooks for databases, web servers, process management, monitoring, and trending.

Search for: chef-client

All Categories

See all Cookbooks

Databases

Process Management

Web Servers

Monitoring & Trending

Add a New Cookbook

Need Help?

REST API

Community Login | Sign Up | Recover Password | Facebook | Twitter

Cookbooks | Users | Questions | Ideas | Chat | Opscode.com

mysql
★★★★★ 5075 downloads

apache2
★★★★★ 3057 downloads

nginx
★★★★★ 2903 downloads

mysql
★★★★★ 5075 downloads

postgresql
★★★★★ 1332 downloads

riak
★★★★★ 777 downloads

runit
★★★★★ 865 downloads

bluepill
★★★★★ 420 downloads

god
★★★★★ 211 downloads

apache2
★★★★★ 3057 downloads

nginx
★★★★★ 2903 downloads

passenger_apache2
★★★★★ 475 downloads

nagios
★★★★★ 1268 downloads

graylog2
★★★★★ 882 downloads

revealcloud
★★★★★ 878 downloads



Search Results...

+ | <http://community.opscode.com/search?query=chef-client&scope=cookbook> Google

Community Login | Sign Up | Recover Password f t

OPSCODE RULE THE CLOUD COMMUNITY Cookbooks | Users | Questions | Ideas | Chat Opscode.com >

Search chef-client Advanced Search

Search results for 'che'

chef-client We're probably looking for this one

Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac_os_x (>= 0.0.0), mac_os_x_server (>= 0.0.0), openbsd (>= 0.0.0), redhat (>= 0.0.0), ubuntu (>= 0.0.0), windows (>= 0.0.0)

★★★★★ 2 ratings Category: Utilities

Manages aspects of only chef-client

432 downloads 36 followers

chef-client-cron

Platforms: centos (>= 0.0.0), debian (>= 0.0.0), fedora (>= 0.0.0), freebsd (>= 0.0.0), mac_os_x (>= 0.0.0), openbsd (>= 0.0.0), redhat (>= 0.0.0), ubuntu (>= 0.0.0)

★★★★★ 0 ratings Category: Utilities

Manages aspects of only chef-client

27 downloads 0 followers

4 downloads 2 followers



Viewing a cookbook

The screenshot shows a web browser displaying the Opscode Community website at <http://community.opscode.com/cookbooks/chef-client>. The page title is "chef-client". The main content area includes a "Description" section with the text "This cookbook is used to configure a system as a Chef Client.", a "Requirements" section, and a large central box containing the text "Browse source code". To the right of this box are two orange buttons: "Download" and "Follow". Below the "Follow" button is a "Maintainer" section with the Opscode logo. A red arrow points from the text "README displayed on the page (if it has one)" to the "Browse source code" box. Another red arrow points from the "Description" section to the explanatory text.

+ | <http://community.opscode.com/cookbooks/chef-client>

Community Login | Sign Up | Recover Password | f | t

Cookbooks | Users | Questions | Ideas | Chat | [Opscode.com >](#)

Search Cookbooks | Advanced Search

chef-client

Manages aspects of only chef-client

★★★★★ 2 ratings | Downloaded 432 times | Last updated

Category: Utilities | Homepage: <http://github.com/opscode-cookbooks/chef-client> | License: Apache 2.0 | Platforms: centos (>= 6.0.0), debian (>= 6.0.0), fedora (>= 17.0.0), freebsd (>= 8.0.0), macosx (>= 10.6.0), redhat (>= 6.0.0), suse (>= 11.0.0)

mac_os_x_server (>= 0.0.0)

Created: Dec 16, 2011 | Last updated: Jun 10, 2012 | Versions: 1.1.4, 1.1

Description

This cookbook is used to configure a system as a Chef Client.

Requirements

Browse source code

Download

Follow

Maintainer

opscode

Collaborators

README displayed on the page
(if it has one)

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!

RULE THE CLOUD

Download and use chef-client cookbook





Exercise: knife cookbook site search

```
$ knife cookbook site search chef-client
```

```
chef:
  cookbook: http://cookbooks.opscode.com/api/v1/cookbooks/chef
  cookbook_description: Installs and configures Chef for chef-client and chef-server
  cookbook_maintainer: opscode
  cookbook_name: chef

chef-client:
  cookbook: http://cookbooks.opscode.com/api/v1/cookbooks/chef-client
  cookbook_description: Manages aspects of only chef-client
  cookbook_maintainer: opscode
  cookbook_name: chef-client

chef-client-cron:
  cookbook: http://cookbooks.opscode.com/api/v1/cookbooks/chef-client-cron
  cookbook_description: Manages aspects of only chef-client
  cookbook_maintainer: bryanwb
  cookbook_name: chef-client-cron

forkable_client:
  cookbook: http://cookbooks.opscode.com/api/v1/cookbooks/forkable_client
  cookbook_description: Provides forked chef-client runs
  cookbook_maintainer: chrisroberts
  cookbook_name: forkable_client
```



Exercise: knife cookbook site show

```
$ knife cookbook site show chef-client
```

```
average_rating: 4.66667
category: Utilities
created_at: 2010-12-16T23:00:45Z
description: Manages aspects of only chef-client
external_url: github.com/opscode-cookbooks/chef-client
latest_version: http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_2_0
maintainer: opscode
name: chef-client
updated_at: 2012-08-30T20:55:46Z
versions:
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_2_0
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_1_4
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_1_2
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_1_0
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_0_4
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_0_2
  http://cookbooks.opscode.com/api/v1/cookbooks/chef-client/versions/1_0_0
```



Exercise: Download chef-client cookbook

```
$ knife cookbook site download chef-client
```

Downloading chef-client from the cookbooks site at
version 1.2.0 to /Users/YOU/chef-repo/chef-
client-1.2.0.tar.gz
Cookbook saved: /Users/YOU/chef-repo/chef-
client-1.2.0.tar.gz



Exercise: Extract chef-client cookbook tarball

```
$ tar -zxvf chef-client*.tar.gz -C cookbooks
```

```
x chef-client/
x chef-client/attributes/
x chef-client/CHANGELOG.md
x chef-client/CONTRIBUTING
x chef-client/LICENSE
x chef-client/metadata.json
x chef-client/metadata.rb
x chef-client/README.md
x chef-client/recipes/
x chef-client/templates/
x chef-client/templates/arch/
x chef-client/templates/default/
x chef-client/templates/windows/
x chef-client/templates/default/debian/
x chef-client/templates/default/redhat/
x chef-client/templates/default/solaris/
x chef-client/templates/arch/conf.d/
x chef-client/templates/arch/rc.d/
x chef-client/recipes/config.rb
x chef-client/recipes/cron.rb
x chef-client/recipes/default.rb
x chef-client/recipes/delete_validation.rb
x chef-client/recipes/service.rb
x chef-client/attributes/default.rb
```

- 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.

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!



Best Practice: This runs as root!

So, you just downloaded source code from the internet.

As root.

To load in the magic machine that:

- **Makes your computers run code**

Read the *entire* cookbook first!

RULE THE CLOUD



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)`

RULE THE CLOUD



Exercise: view delete_validation recipe

```
$ vi cookbooks/chef-client/recipes/delete_validation.rb
```

```
20 unless node[:recipes].include?("chef-server")
21   file Chef::Config[:validation_key] do
22     action :delete
23     backup false
24     only_if { ::File.exists?(Chef::Config[:client_key]) }
25   end
26 end
```

Exercise: Add chef-client::delete_validation to your base role

```
$ vi ./roles/base.rb
```

```
1 name "base"
2 description "Base Server Role"
3 run_list "recipe[chef-client::delete_validation]", "recipe[motd]",
           "recipe[users]"
4 #def_run_list = "recipe[motd]", "recipe[users]"
5 #env_run_lists({
6 #  "_default" => def_run_list,
7 #  "dev" => def_run_list,
8 #  "production" => [
9 #    "recipe[motd]"
10 #  ]
11 #})
```

- Add the **delete_validation** recipe
- Comment the environment specifics (*in red*)



Best Practice: Delete the validation certificate when it isn't required

- Once Chef enters the actual run, synchronizing cookbooks, it has registered its own API client with the validation certificate.
- That certificate is no longer required. We do this first because in case the run fails for another reason, we know at least the validation certificate is gone.

RULE THE CLOUD



Exercise: View the default recipe

```
$ vi ./cookbooks/chef-client/recipes/default.rb
```

```
12 #
13 # Unless required by applicable law or agreed to in writing,
14 # distributed under the License is distributed on an "AS
15 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either ex-
16 # See the License for the specific language governing per-
17 # limitations under the License.
18 #
19
20 include_recipe "chef-client::service"
```

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 service recipe

```
$ vi ./cookbooks/chef-client/recipes/service.rb
```

```
71 case node["chef_client"]["init_style"]
72 when "init"
73
74   dist_dir, conf_dir = value_for_platform(
75     ["ubuntu", "debian"] => { "default" => ["debian", "default"] },
76     ["redhat", "centos", "fedora", "scientific", "amazon"] => { "default" =>
77       ["suse"] => { "default" => ["suse", "sysconfig"] } }
78 )
79
80   template "/etc/init.d/chef-client" do
81     source "#{dist_dir}/init.d/chef-client.erb"
82     mode 0755
83     variables(
84       :client_bin => client_bin
85     )
86     notifies :restart, "service[chef-client]", :delayed
87   end
88
89   template "/etc/#{conf_dir}/chef-client" do
90     source "#{dist_dir}/#{conf_dir}/chef-client.erb"
91     mode 0644
92     notifies :restart, "service[chef-client]", :delayed
93   end
94
95   service "chef-client" do
96     supports :status => true, :restart => true
97     action :enable
98   end
```

- The recipe supports a number of **service providers** and styles.
- It works on a lot of **platforms**.
- Everything is controllable through **attributes**.



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.

RULE THE CLOUD



Exercise: Upload the chef-client cookbook

```
$ knife cookbook upload chef-client
```

Uploading chef-client [1.2.0]
Uploaded 1 cookbook.



Exercise: Add chef-client recipe to base role

```
> vi ./roles/base.rb
```

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



Exercise: upload the base role

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

Updated Role base!

Exercise: Run chef-client on your node

```
$ sudo chef-client
```

```
INFO: Processing file[/etc/chef/validation.pem] action delete (chef-client::delete_validation line 21)
INFO: file[/etc/chef/validation.pem] deleted file at /etc/chef/validation.pem
INFO: Processing directory[/var/run/chef] action create (chef-client::service line 42)
INFO: directory[/var/run/chef] created directory /var/run/chef
INFO: Processing directory[/var/cache/chef] action create (chef-client::service line 42)
INFO: directory[/var/cache/chef] created directory /var/cache/chef
INFO: Processing directory[/var/lib/chef] action create (chef-client::service line 42)
INFO: directory[/var/lib/chef] created directory /var/lib/chef
INFO: Processing directory[/var/log/chef] action create (chef-client::service line 42)
INFO: directory[/var/log/chef] created directory /var/log/chef
INFO: Processing template[/etc/init.d/chef-client] action create (chef-client::service line 61)
INFO: template[/etc/init.d/chef-client] mode changed to 755
INFO: template[/etc/init.d/chef-client] updated content
INFO: Processing template[/etc/default/chef-client] action create (chef-client::service line 70)
INFO: template[/etc/default/chef-client] mode changed to 644
INFO: template[/etc/default/chef-client] updated content
INFO: template[/etc/default/chef-client] not queuing delayed action restart on service[chef-client]
(delayed), as it's already been queued
INFO: Processing service[chef-client] action enable (chef-client::service line 76)
INFO: service[chef-client] enabled
INFO: template[/etc/init.d/chef-client] sending restart action to service[chef-client] (delayed)
INFO: Processing service[chef-client] action restart (chef-client::service line 76)
INFO: service[chef-client] restarted
```



Exercise: Verify chef-client is running

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

```
root      3227  1.1  1.7 113928 30040 ?          Sl
20:23  0:01 /opt/chef/embedded/bin/ruby /usr/bin/
chef-client -d -P /var/run/chef/client.pid -L /var/
log/chef/client.log -c /etc/chef/client.rb -i 1800 -s
20
```

- Our node is now running chef-client as a daemon, and it will converge itself over time on a (by default) 30 minute interval.
- The amount of resources converged may vary with longer intervals, depending on configuration drift on the system.
- Because Chef resources are idempotent, it will only configure what it needs to each run.

Download and use ntp cookbook



- Network time protocol - keeps system clocks in sync
- Chef Server authentication is time sensitive!



Exercise: Download the ntp cookbook

```
$ knife cookbook site download ntp
```

```
Downloading ntp from the cookbooks site at version  
1.1.8 to /Users/YOU/chef-repo/ntp-1.2.0.tar.gz  
Cookbook saved: /Users/YOU/chef-repo/ntp-1.2.0.tar.gz
```



Exercise: Extract the ntp cookbook

```
$ tar -zxvf ntp*.tar.gz -C cookbooks
```

```
x ntp/
x ntp/attributes/
x ntp/CHANGELOG.md
x ntp/CONTRIBUTING
x ntp/LICENSE
x ntp/metadata.json
x ntp/metadata.rb
x ntp/README.md
x ntp/recipes/
x ntp/templates/
x ntp/templates/default/
x ntp/templates/default/ntp.conf.erb
x ntp/recipes/default.rb
x ntp/attributes/default.rb
```

The cookbook is quite flexible, but for this exercise we're just interested in the most basic use, an NTP client.

- default recipe

Exercise: view the ntp default recipe

```
> vi ./cookbooks/ntp/recipes/default.rb
```

```
22 case node[:platform]
23 when "ubuntu", "debian"
24   package "ntpdate" do
25     action :install
26   end
27   package "ntp" do
28     action :install
29   end
30 when "redhat", "centos", "fedora"
31   package "ntp" do
32     action :install
33   end
34   package "ntpdate" do
56   service node[:ntp][:service] do
57     supports :status => true, :restart => true
58     action [ :enable, :start ]
59   end
60
61   template "/etc/ntp.conf" do
62     source "ntp.conf.erb"
63     owner "root"
64     group root_group
65     mode "0644"
66     notifies :restart, resources(:service => node[:ntp][:service])
67   end
```

- The template notifies the service



Exercise: Upload the ntp cookbook

```
$ knife cookbook upload ntp
```

Uploading ntp
Uploaded 1 cookbook.

[1.2.0]



Exercise: Add the ntp recipe to the base role

```
$ vi ./roles/base.rb
```

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



Exercise: Upload the base role

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

Updated Role base!



Exercise: Run chef-client on the node

```
$ sudo chef-client
```

```
INFO: Processing package[ntp] action install (ntp::default line 21)
INFO: Processing package[ntpdate] action install (ntp::default line 21)
INFO: Processing directory[/var/lib/ntp] action create (ntp::default
line 26)
INFO: Processing directory[/var/log/ntpstats/] action create
(ntp::default line 26)
INFO: Processing service[ntp] action enable (ntp::default line 33)
INFO: Processing service[ntp] action start (ntp::default line 33)
INFO: Processing template[/etc/ntp.conf] action create (ntp::default
line 38)
INFO: template[/etc/ntp.conf] backed up to /var/chef/backup/etc/
ntp.conf.chef-20121023040506
INFO: template[/etc/ntp.conf] updated content
INFO: template[/etc/ntp.conf] owner changed to 0
INFO: template[/etc/ntp.conf] group changed to 0
INFO: template[/etc/ntp.conf] mode changed to 644
```

- 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?

Just enough Ruby for Chef



- 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 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



Exercise: Start `irb` on your test node

```
$ /opt/chef/embedded/bin/irb
```

```
irb(main):001:0>
```



Exercise: Variable assignment

```
$ /opt/chef/embedded/bin/irb
```

```
irb(main):001:0> x = "hello"
=> "hello"
irb(main):002:0> puts x
hello
=> nil
```



Exercise: Arithmetic

```
irb(main):003:0> 1 + 2  
=> 3
```



Exercise: Arithmetic

```
irb(main):004:0> 18 - 5  
=> 13
```



Exercise: Arithmetic

```
irb(main):005:0> 2 * 7  
=> 14
```



Exercise: Arithmetic

```
irb(main):006:0> 5 / 2  
=> 2
```



Exercise: Arithmetic

```
irb(main):007:0> 5 / 2.0  
=> 2.5
```



Exercise: Arithmetic

```
irb(main):008:0> 5.class
=> Fixnum
irb(main):009:0> 5.0.class
=> Float
```



Exercise: Arithmetic

```
irb(main):010:0> 1 + (2 * 3)
=> 7
```



Exercise: Strings

```
irb(main):011:0> 'jungle'  
=> "jungle"
```



Exercise: Strings

```
irb(main):012:0> 'it\'s alive'  
=> "it's alive"
```



Exercise: Strings

```
irb(main):013:0> "animal"  
=> "animal"
```



Exercise: Strings

```
irb(main):014:0> "\"so easy\""  
=> "\"so easy\""  
irb(main):015:0> puts "\"so easy\""  
"so easy"  
=> nil
```

```
irb(main):016:0> x = "pretty"
=> "pretty"
irb(main):017:0> "#{x} nice"
=> "pretty nice"
irb(main):018:0> '#{x} nice'
=> "\#{x} nice"
```

```
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
```

```
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
```

```
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
```

```
irb(main):035:0> 5 <=> 5
=> 0
irb(main):036:0> 5 <=> 6
=> -1
irb(main):037:0> 5 <=> 4
=> 1
```

```
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"
```

```
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"]
```

```
irb(main):046:0> x << "e"
=> ["a", "b", "c", "d", "e"]
irb(main):047:0> x
=> ["a", "b", "c", "d", "e"]
```

```
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"]
```

```
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"]
```



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"}
```

```
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"]
```



Exercise: Hashes

```
irb(main):060:0> h.values  
=> ["Gary", "Gygax", 33]
```

```
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}
```



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
```



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
```



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
```



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
```



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
```

```
irb(main):095:0> class Person
irb(main):096:1>   attr_accessor :name, :is_metal
irb(main):097:1>
irb(main):098:1*>
irb(main):099:2>
irb(main):100:3>
irb(main):101:3>
irb(main):102:2>   def metal
irb(main):103:1>     if @is_metal
irb(main):104:2>       puts "!!#{@name} is metal!!"
irb(main):105:3>     end
irb(main):106:2>   end
irb(main):107:1> end
=> nil
```

```
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
```

- 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>

Shef - The Chef Console



- Learn about Shef, the interactive Chef console (REPL)
- Understand Shef's execution contexts
- Use Shef to inspect and debug recipes
- Interact with a Chef Server API
- Learn about the magic that is knife exec

- Interactive Ruby console (REPL) for Chef.
- Three startup modes (standalone, solo, client).
- Three run-time contexts (main, attribute, recipe).
- Configured with a specific or implied config file.

Three modes for startup

- Standalone: no cookbooks and empty run list (default)
- Solo: attempts to act as chef-solo, using solo config and JSON attributes. (-s or --solo)
- Client: contacts chef server for node object, retrieving cookbooks and using specified run list (-z or --client).



Exercise: Start the shef REPL

```
$ sudo shef -z
```

```
loading configuration: /etc/chef/client.rb
Session type: client
Loading...[2012-10-23T04:08:35+00:00] INFO: Run List is [role[webserver]]
[2012-10-23T04:08:35+00:00] INFO: Run List expands to [chef-client::delete_validation, chef-client,
ntp, motd, users, apache]
..[2012-10-23T04:08:36+00:00] INFO: Loading cookbooks [apache, chef-client, motd, ntp, pci, users]
.done.
```

This is shef, the Chef shell.

Chef Version: 10.16.0

<http://www.opscode.com/chef>

<http://wiki.opscode.com/display/chef/Home>

run `help` for help, `exit` or ^D to quit.

Ohai2u [ubuntu@target1.local!](#)

chef >

```
chef > help
```

Chef Help	
Command	Description
help	prints this help message
version	prints information about chef
recipe	switch to recipe mode
attributes	switch to attributes mode
run_chef	run chef using the current recipe
chef_run	returns an object to control a paused chef run
chef_run.resume	resume the chef run
chef_run.step	run only the next resource
chef_run.skip_back	move back in the run list
chef_run.skip_forward	move forward in the run list
reset	resets the current recipe
become_node	assume the identity of another node.
echo	turns printout of return values on or off
echo?	says if echo is on or off
tracing	turns on or off tracing of execution. *verbose*
tracing?	says if tracing is on or off
ls	simple ls style command
node	returns the current node (i.e., this host)
ohai	pretty print the node's attributes
edit	edit an object in your EDITOR
clients	Find and edit API clients

```
chef > help(:nodes)
```

```
Command: nodes
=====
## SUMMARY ##
+nodes+ Allows you to query your chef server for information about your nodes.

## LIST ALL NODES ##
You can list all nodes using +all+ or +list+

    nodes.all #=> [<Chef::Node...>, <Chef::Node...>, ...]

To limit the information returned for each node, pass a code block to the +all+
subcommand:

    nodes.all { |node| node.name } #=> [NODE1_NAME, NODE2_NAME, ...]

## SHOW ONE NODE ##
You can show the data for a single node using the +show+ subcommand:

    nodes.show("NODE_NAME") => <Chef::Node @name="NODE_NAME" ...>

## SEARCH FOR NODES ##
You can search for nodes using the +search+ or +find+ subcommands:

    nodes.find(:name => "app*") #=> [<Chef::Node @name="app1.example.com" ...>, ...]
```



Exercise: Look at the company attribute for the node object

```
chef > node['company']
```

```
=> "opscode"
```



Exercise: Change the context you are in to that of
a cookbook attribute file

`chef > attributes`

`chef:attributes >`



Exercise: Add a default attribute

```
chef:attributes > node.normal["interactive"] = "is cool"
```

```
=> "is cool"
```



Exercise: Add a default attribute

```
chef:attributes > node["interactive"]
```

```
=> "is cool"
```



Exercise: Search for new default attribute

```
$ knife node show target1.local -a interactive
```

interactive:



Exercise: Save node state

```
chef:attributes > node.save
```

```
=> <Chef::Node:0x11671f4 @name="target1.local">
```



Exercise: Search for new default attribute

```
$ knife node show target1.local -a interactive
```

interactive: is cool



Exercise: Display all node attributes

```
$ knife node show target1.local -Fj -l | less
```

```
"name": "target1.local",
"chef_environment": "dev",
"run_list": [
  "role[webserver]"
],
"normal": {
  "company": "opscode",
  "tags": [
  ],
  "interactive": "is cool"
},
"default": {
  "apache": {
    "sites": {
      "clowns": {
        "port": 80
      },
      "bears": {
        "port": 81
      }
    }
  }
},
"override": {},
"automatic": {
  "languages": {
    "ruby": {

```



Exercise: Switch to recipe context

chef:attributes > **recipe**

chef:recipe >

Exercise: Use a package resource

chef:recipe > package "jove"

```
=> <package[jove] @name: "jove" @noop: nil @before: nil  
@params: {} @provider: nil @allowed_actions:  
[:nothing, :install, :upgrade, :remove, :purge, :reconfig]  
@action: :install @updated: false @updated_by_last_action:  
false @supports: {} @ignore_failure: false @retries: 0  
@retry_delay: 2 @immediate_notifications: []  
@delayed_notifications: [] @source_line: "(irb#1):1:in  
'irb_binding'" @resource_name: :package @package_name: "jove"  
@version: nil @candidate_version: nil @response_file: nil  
@source: nil @options: nil @cookbook_name: nil @recipe_name:  
nil @enclosing_provider: nil>
```

chef:recipe > run_chef

```
DEBUG: Processing package[jove] on george-training.novalocal
INFO: Processing package[jove] action install ((irb#1) line 1)
DEBUG: package[jove] checking package status for jove
jove:
  Installed: (none)
  Candidate: 4.16.0.72-2.2
  Version table:
    4.16.0.72-2.2 0
      500 http://archive.ubuntu.com/ubuntu/ precise/universe i386
Packages
DEBUG: package[jove] current version is nil
DEBUG: package[jove] candidate version is 4.16.0.72-2.2
DEBUG: Executing apt-get -q -y install jove=4.16.0.72-2.2
```

chef:recipe > **reset**

```
Loading.....INFO: Run List is [role[webserver]]  
INFO: Run List expands to [motd, users, ntp, apache]  
...INFO: Loading cookbooks [apache, motd, ntp, pci,  
users]  
.done.
```

```
=> nil
```



Exercise: Debug an existing recipe

`chef:recipe > include_recipe "users"`

```
=> [#<Chef::Recipe:0x9e8cb38 @cookbook_name="users", @recipe_name="default",
@run_context=#<Chef::RunContext:0x9f23dd0 @node=<Chef::Node:0x5054a4c @name="george-training.novalocal">,
@cookbook_collection={"pci"=>#<Chef::CookbookVersion:0xa01add8 @name="pci", @frozen=false,
@attribute_filenames=["/var/chef/cache/cookbooks/pci/attributes/default.rb"], @definition_filenames=[],
@template_filenames=[], @file_filenames[], @recipe_filenames=["/var/chef/cache/cookbooks/pci/recipes/
default.rb"], @recipe_filenames_by_name={"default"=>"/var/chef/cache/cookbooks/pci/recipes/default.rb"},
@library_filenames[], @resource_filenames[], @provider_filenames[], @metadata_filenames[],
@root_dir=nil, @root_filenames=["/var/chef/cache/cookbooks/pci/metadata.rb", "/var/chef/cache/cookbooks/
pci/README.md"], @couchdb_id=nil, @couchdb=#<Chef::CouchDB:0xa01ace8 @db="chef", @rest=#<Chef::REST:
0xa01acd4 @url="http://localhost:5984", @cookies={}, @default_headers={},
@auth_credentials=#<Chef::REST::AuthCredentials:0xa01ac98 @key_file=nil, @client_name=nil>,
@sign_request=true, @sign_on_redirect=true, @redirects_followed=0, @redirect_limit=10>>,
@couchdb_rev=nil, @status=:ready, @manifest={"name"=>"pci-0.0.1", "definitions"=>[],
"json_class"=>"Chef::CookbookVersion", "files"=>[], "attributes"=>[{"name"=>"default.rb",
"url"=>"https://s3.amazonaws.com/opscode-platform-production-data/organization-f976ba971caf40918cea53a9de6076cc/checksum-0d35ace084267f0438afac50907c2163?AWSAccessKeyId=AKIAJ0ZTD2N26S7W6APA&Expires=1340264374&Signature=YVJMj7Upfjfkhrbz7WFPLLKq7kg%3D"}, "checksum"=>"0d35ace084267f0438afac50907c2163", "path"=>"attributes/default.rb",
"specificity"=>"default"}], "metadat
```



Exercise: Examine the resources it created

`chef:recipe > resources`

```
["user[frank]", "user[bobo]", "group[clowns]"]
=> ["user[frank]", "user[bobo]", "group[clowns]"]
```



Exercise: Grab a single resource

```
chef:recipe > resources("user[frank]")
```

```
<user[frank] @name: "frank" @noop: nil @before: nil @params: {} @provider: nil @allowed_actions: [:nothing, :create, :remove, :modify, :manage, :lock, :unlock] @action: :create @updated: false @updated_by_last_action: false @supports: {:manage_home=>false, :non_unique=>false} @ignore_failure: false @retries: 0 @retry_delay: 2 @immediate_notifications: [] @delayed_notifications: [] @source_line: "/var/chef/cache/cookbooks/users/recipes/default.rb:11:in `block in from_file'" @resource_name: :user @username: "frank" @comment: "Frank Belson" @uid: 2001 @gid: 0 @home: "/home/frank" @shell: "/bin/bash" @password: nil @system: false @manage_home: false @non_unique: false @cookbook_name: "users" @recipe_name: "default" @enclosing_provider: nil> => <user[frank] @name: "frank" @noop: nil @before: nil @params: {} @provider: nil @allowed_actions: [:nothing, :create, :remove, :modify, :manage, :lock, :unlock] @action: :create @updated: false @updated_by_last_action: false @supports: {:manage_home=>false, :non_unique=>false} @ignore_failure: false @retries: 0 @retry_delay: 2 @immediate_notifications: [] @delayed_notifications: [] @source_line: "/var/chef/cache/cookbooks/users/recipes/default.rb:11:in `block in from_file'" @resource_name: :user @username: "frank" @comment: "Frank Belson" @uid: 2001 @gid: 0 @home: "/home/frank" @shell: "/bin/bash" @password: nil @system: false @manage_home: false @non_unique: false @cookbook_name: "users" @recipe_name: "default" @enclosing_provider: nil>
```



Exercise: Show the resource *as you would write it if you were very, very verbose*

chef:recipe > puts resources("user[frank]").to_text

```
# Declared in /var/chef/cache/cookbooks/users/recipes/default.rb:11:in `block in
from_file'
user("frank") do
  action(:create)
  supports({:manage_home=>false, :non_unique=>false})
  retries(0)
  retry_delay(2)
  username("frank")
  comment("Frank Belson")
  uid(2001)
  gid(0)
  home("/home/frank")
  shell("/bin/bash")
  cookbook_name("users")
  recipe_name("default")
end
```



Exercise: Exit to the main context

```
chef:recipe > exit
```

```
chef >
```



Exercise: List all the nodes

```
chef > nodes.list
```

```
=> [node[target1.local]]
```



Exercise: Print all the node names and their environment

```
nodes.all { |n| "#{n.name}: #{n.chef_environment}" }
```

```
target1.local: production  
=> [nil]
```



Exercise: Change the environment on every node to dev

```
nodes.transform(:all) { |n| n.chef_environment("dev") }
```

```
=> [node[target1.local]]
```



Exercise: From your workstation, use knife to do
the same thing

```
$ knife exec -E 'nodes.transform(:all) { |n|  
  n.chef_environment("dev") }'
```

Has.. no output >:)



Exercise: Check it from search ...

```
$ knife search node '*:*' -a chef_environment
```

```
1 items found
```

```
chef_environment: dev
id: target1.local
```

- We will not save you from yourself
- You can do the same tricks on roles, data bags (and items), clients.. all kinds of stuff
- You can (and will) blow off your foot if you aren't careful

- What is Shef?
- What are the execution contexts of Shef? How do you know which one you are in?
- How do you run a recipe that is in your nodes run list from Shef?
- What does “transform” do?

Building LWRPs



- Understand the components of LWRPs
- Be familiar with the Resource DSL
- Be familiar with the Provider DSL
- Build a LWRP from scratch
- Look at examples of other LWRPs

When to use Lightweight Resources and Providers

- LWRPs are used when you want to abstract a repeated pattern of behavior with a declarative interface that doesn't already exist in Chef
- Examples
 - Managing a service with a new supervision system (e.g. bluepill or daemontools)
 - Subsystem user management (e.g samba)
 - Sending a message to a RESTful API for a monitoring service with `http_request`

Components of an LWRP

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

The Problem and the Success Criteria

- **The Problem:** We don't have anything cute and awesome on our system yet
- **Success Criteria:** We will create a mouse that tells us cute and awesome things about itself



Exercise: Create a cookbook named 'mouse'

```
$ knife cookbook create mouse
```

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



Exercise: Open the default recipe in your editor

```
$ vi cookbooks/mouse/recipes/default.rb
```

```
1 #
2 # Cookbook Name:: users
3 # Recipe:: default
4 #
5 # Copyright 2012, YOUR_COMPANY_NAME
6 #
7 # All rights reserved - Do Not Redistribute
8 #
```



Exercise: Open the default recipe in your editor

```
1 #
2 # Cookbook Name:: mou
3 # Recipe:: default
4 #
5 # Copyright 2012, YOU
6 #
7 # All rights reserved
8 #
9
10 mouse "Itchy"
```

- We call a new resource: “mouse”
- We create a new instance of that resource named “Itchy”



Exercise: Upload the mouse cookbook

```
$ knife cookbook upload mouse
```

```
Uploading mouse  
Uploaded 1 cookbook.
```

```
[0.0.1]
```



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

```
$ knife node run list add target1.local "recipe[mouse]"
```

```
run_list:  
  role[webserver]  
  recipe[mouse]
```



Example: How we would have done that before

```
$ knife node edit target1.local
```

```
1  {
2    "name": "target1.local",
3    "chef_environment": "_default",
4    "normal": {
5      "company": "opscode",
6      "tags": [
7
8        ],
9        "apache": {
10          "sites": {
11            }
12          }
13        },
14        "run_list": [
15          "role[webserver]",
16          "recipe[mouse]"
17        ]
18    }
19 }
```



Exercise: Run the Chef Client

```
$ sudo chef-client
```

```
=====
Recipe Compile Error in /var/chef/cache/cookbooks/mouse/recipes/default.rb
=====
```

```
NameError
```

```
-----  
Cannot find a resource for mouse on ubuntu version 12.04
```

```
Cookbook Trace:
```

```
-----  
/var/chef/cache/cookbooks/mouse/recipes/default.rb:10:in `from_file'
```

```
Relevant File Content:
```

```
-----  
/var/chef/cache/cookbooks/mouse/recipes/default.rb:
```

```
3: # Recipe::: default  
4: #  
5: # Copyright 2012, YOUR_COMPANY_NAME  
6: #  
7: # All rights reserved - Do Not Redistribute  
8: #  
9:  
10>> mouse "Itchy"  
11:
```

```
ERROR: Running exception handlers  
FATAL: Saving node information to /var/chef/cache/failed-run-data.json  
ERROR: Exception handlers complete  
FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
```



Exercise: Create resources for the mouse cookbook

```
$ touch ./cookbooks/mouse/resources/default.rb  
$ knife cookbook upload mouse
```

Uploading mouse
Uploaded 1 cookbook.

[0.1.0]



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing mouse[Itchy] action nothing (mouse::default line 10)
=====
Error executing action `nothing` on resource 'mouse[Itchy]'
=====

ArgumentError
-----
Cannot find a provider for mouse[Itchy] on ubuntu version 12.04

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

10: mouse "Itchy"

Compiled Resource:
-----
# Declared in /var/chef/cache/cookbooks/mouse/recipes/default.rb:10:in `from_file'

mouse("Itchy") do
  action :nothing
  retries 0
  retry_delay 2
  cookbook_name "mouse"
  recipe_name "default"
end

ERROR: Running exception handlers
FATAL: Saving node information to /var/chef/cache/failed-run-data.json
```



Exercise: Create providers for the mouse cookbook

```
$ touch ./cookbooks/mouse/providers/default.rb  
$ knife cookbook upload mouse
```

Uploading mouse
Uploaded 1 cookbook.

[0.1.0]



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing mouse[Itchy] action nothing  
(mouse::default line 10)
```

```
INFO: Chef Run complete in 2.50350876 seconds
```

```
INFO: Running report handlers
```

```
INFO: Report handlers complete
```

- Two methods used: actions and attribute
 - actions specifies allowed actions that can be used in a recipe
 - attribute specifies a new parameter attribute for the resource

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



Exercise: Set an action in our mouse default recipe

```
$ vi ./cookbooks/mouse/recipes/default.rb
```

```
10 mouse "Itchy" do
11   action :say
12 end
~
```

- We call resource: “mouse”
- With name “Itchy”
- With action “:say” in the parameter block



Exercise: Create allowed actions for the mouse resource

```
$ vi ./cookbooks/mouse/resources/default.rb
```

```
1 actions :say
```

```
~
```

- We have one allowed resource action



Exercise: Create provider for the “:say” action

```
$ vi ./cookbooks/mouse/providers/default.rb
```

```
1 action :say do
2   puts "My name is #{new_resource.name}"
3 end
```

~

- When our mouse resource calls "action :say", execute this block



Exercise: Upload the mouse cookbook

```
$ knife cookbook upload mouse
```

```
Uploading mouse  
Uploaded 1 cookbook.
```

```
[0.1.0]
```



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing mouse[Itchy] action say  
(mouse::default line 10)
```

```
My name is Itchy
```

```
INFO: Chef Run complete in 3.676864182 seconds
```

```
INFO: Running report handlers
```

```
INFO: Report handlers complete
```

Exercise: Use a Chef Resource within your provider

```
$ vi ./cookbooks/mouse/providers/default.rb
```

```
1 action :say do
2   log "My name is #{new_resource.name}"
3 end
```

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



Exercise: Upload the mouse cookbook

```
$ knife cookbook upload mouse
```

```
Uploading mouse  
Uploaded 1 cookbook.
```

```
[0.1.0]
```



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing mouse[Itchy] action say  
(mouse::default line 10)  
INFO: Processing log[My name is Itchy] action write  
(/var/chef/cache/cookbooks/mouse/providers/default.rb  
line 2)  
INFO: My name is Itchy  
INFO: Chef Run complete in 3.825978405 seconds  
INFO: Running report handlers  
INFO: Report handlers complete
```

Exercise: Create attribute parameters for the mouse resource

```
$ vi ./cookbooks/mouse/resources/default.rb
```

```
1 actions :say
2
3 attribute :given_name, :name_attribute => true
4 attribute :phrase, :default => "squeak"
5 attribute :tail, :default => true, :kind_of => [ TrueClass, FalseClass ]
```

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

```
1 actions :say
2
3 attribute :given_name, :name_attribute => true
4 attribute :phrase, :default => "squeak"
5 attribute :tail, :default => true, :kind_of => [ TrueClass, FalseClass ]
```

- **:name_attribute** sets given_name to the resource name
- **:default** sets a default value for this parameter
- **:kind_of** ensures the value is a particular class
- More validation parameters listed on the wiki!

Exercise: Extend provider actions for new resource parameters

```
$ vi ./cookbooks/mouse/providers/default.rb
```

```
1 action :say do
2   log "My name is #{new_resource.given_name}"
3   unless new_resource.phrase =~ /^squeak$/
4     log "#{new_resource.phrase}"
5   end
6   log "I #{new_resource.tail ? 'do' : 'do not'} have a tail"
7 end
```

- The unless statement checks for anything except a squeak
- A Ruby ternary operation checks for the existence of a tail



Exercise: Modify the recipe to use our new parameters

```
$ vi ./cookbooks/mouse/recipes/default.rb
```

```
10 mouse "Itchy" do
11   action :say
12   phrase "SQUEAK!"
13   tail false
14 end
```



Exercise: Upload the mouse cookbook

```
$ knife cookbook upload mouse
```

```
Uploading mouse  
Uploaded 1 cookbook.
```

```
[0.1.0]
```



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing mouse[Itchy] action say (mouse::default line 10)
INFO: Processing log[My name is Itchy] action write (/var/chef/
cache/cookbooks/mouse/providers/default.rb line 2)
INFO: My name is Itchy
INFO: Processing log[SQUEAK!] action write (/var/chef/cache/
cookbooks/mouse/providers/default.rb line 4)
INFO: SQUEAK!
INFO: Processing log[I do not have a tail] action write (/var/
chef/cache/cookbooks/mouse/providers/default.rb line 6)
INFO: I do not have a tail
INFO: Chef Run complete in 2.748432652 seconds
INFO: Running report handlers
INFO: Report handlers complete
```



Exercise: Set the default action to use for this resource

```
$ vi ./cookbooks/mouse/resources/default.rb
```

```
5 attribute :tail, :default =:  
6  
7 def initialize(*args)  
8   super  
9   @action = :say  
10 end
```

- `action` is an instance variable
- its value must be a Ruby symbol



Exercise: Add another mouse to our recipe

```
$ vi ./cookbooks/mouse/recipes/default.rb
```

```
10 mouse "Itchy" do
11   action :say
12   phrase "squeak"
13   tail false
14 end
15
16 mouse "Mighty Mouse" do
17   phrase "Here I come to save the day!"
18   tail true
19 end
```



Exercise: Upload the mouse cookbook

```
$ knife cookbook upload mouse
```

```
Uploading mouse  
Uploaded 1 cookbook.
```

```
[0.1.0]
```



Exercise: Re-run the Chef Client

```
$ sudo chef-client
```

```
INFO: Processing mouse[Itchy] action say (mouse::default line 10)
INFO: Processing log[My name is Itchy] action write (/var/chef/cache/cookbooks/mouse/
providers/default.rb line 2)
INFO: My name is Itchy
INFO: Processing log[I do not have a tail] action write (/var/chef/cache/cookbooks/mouse/
providers/default.rb line 6)
INFO: I do not have a tail
INFO: Processing mouse[Mighty Mouse] action say (mouse::default line 16)
INFO: Processing log[My name is Mighty Mouse] action write (/var/chef/cache/cookbooks/mouse/
providers/default.rb line 2)
INFO: My name is Mighty Mouse
INFO: Processing log[Here I come to save the day!] action write (/var/chef/cache/cookbooks/
mouse/providers/default.rb line 4)
INFO: Here I come to save the day!
INFO: Processing log[I do have a tail] action write (/var/chef/cache/cookbooks/mouse/
providers/default.rb line 6)
INFO: I do have a tail
INFO: Chef Run complete in 4.138446206 seconds
INFO: Running report handlers
INFO: Report handlers complete
```



LWRP examples in Opscode Cookbooks

- Opscode uses LWRPs in its open-source cookbooks
 - aws_elastic_ip
 - aws_ebs_volume
 - bluepill_service
 - mysql_database
 - php_pear
 - riak_cluster
 - samba_user
 - windows_package
- <http://wiki.opscode.com/display/chef/Opscode+LWRP+Resources>

- What are the components of an LWRP?
- What are the resource DSL's two methods?
- What method is used in providers to create actions?
- What are validation parameters? What are two examples?
- What does “:name_attribute” mean? How is it used?

Troubleshooting



Further Resources





Further Resources

- <http://opscode.com/>
- <http://community.opscode.com/>
- <http://wiki.opscode.com/>
- <http://lists.opscode.com>
- <http://youtube.com/user/Opscode>
- [irc.freenode.net #chef](#)

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





#ChefConf 2013

#ChefConf
2013

Your Road to Automation

April 24 - 26, 2013
San Francisco, California

Register Early and Save