

A Taste of Chef on AWS

Chef basics & AWS Integration

training@chef.io

Copyright (C) 2015 Chef Software, Inc.

Pre-Requisites

- Your own workstation
- Enough power to make it through the day
- Have an SSH client on your local workstation

Everything else will be provided

Introductions

AWS PopUp Loft - A Taste of Chef v2.2.0

Instructor Introduction

```
# finger $(whoami)
Login: gmiranda                               Name: George Miranda
Directory: /home/gmiranda                         Shell: /bin/bash
EDITOR: /usr/bin/vim + janus (https://github.com/carlhuda/janus)
On since Mon 14 Apr 1997 18:01 (GMT) on ttyl from :0
On AWS since Mon 05 Apr 2010 15:30 (GMT) on ec2 from :EA
```

No mail on `gmiranda@chef.io`

Plan:

twitter:	gmiranda23
github:	gmiranda23
irc:	gmiranda23 (irc.freenode.net - #chef)
.:	gmiranda23

CHEF Lab Assistant



Ricardo Lupo
rlupo@chef.io

Introduce Yourselves

- Stand up & face each other
- Name
- Current job role
- Previous job roles/background
- Experience with Chef and/or config management
- Experience with AWS EC2 and/or other AWS services

Quick Survey

- Hands up: who is a developer?

Quick Survey

- Hands up: who is a operations guru?

Quick Survey

- Hands up: who is a startup founder?

Quick Survey

- Wait, this is our first workshop at the NYC AWS Loft
(okay, let's chat a little)

Quick Survey

- Guess what: you're all developers today!
- Chef is Infrastructure as Code

Quick Survey

- Hands up: who is new to AWS?

Quick Survey

- Hands up: who is new to Chef?

Course Objectives and Style

AWS PopUp Loft - A Taste of Chef v2.2.0

Course Objectives

- Upon completion of this course you will be able to
 - Automate common infrastructure tasks with Chef
 - Describe some of Chef's tools
 - Apply Chef's primitives to solve your problems
 - Use Chef to manage EC2 instances
 - Use Chef to manage an entire application topology in AWS
 - Know where to go for more info

Learning to use Chef

- This is not a comprehensive course
- We will cover lots of topics, but won't go too deep on any of them
- An in-depth Chef Fundamentals training session can help you unlock the full power and flexibility of Chef
- Our goal is to cover enough basics around Chef and AWS integration to make getting started easier

Chef is a Language

- Learning Chef is like learning the basics of a language
- Today, we start with essential words and phrases
- 80% fluency can be gained from a full multi-day Chef Fundamentals training course
- The remaining 20% just takes practice
- The best way to **learn** Chef is to *use* Chef

Just a taste of Chef

- This course basic use of Chef with AWS
- We cover additional topics (e.g.trends in the Chef or AWS ecosystems) as time permits
- Any discussion items that are too far off topic will be captured
 - We'll return to these items at the end of class

Training is a discussion

- Lots of hands on labs
- Lots of typing
- Ask questions when they come to you
- Ask for help when you need it
- We will troubleshoot issues on the spot

Agenda

AWS PopUp Loft - A Taste of Chef v2.2.0

Topics

- Overview of Chef
- Workstation setup
- Resources
- Describing desired state
- Creating a simple webserver desired state
- Applying desired state to new EC2 instances
- Desired state for other AWS services
- Healing your infrastructure
- Wrap Up

Breaks!

- We'll take a break between each section, or every hour, whichever comes first
- We'll take a break for lunch
- If we finish early, we're happy to run some hack time

Legend

AWS PopUp Loft - A Taste of Chef v2.2.0

Legend: Example Terminal Command and Output

```
$ ifconfig
```

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    options=3<RXCSUM,TXCSUM>
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        inet 127.0.0.1 netmask 0xff000000
            inet6 ::1 prefixlen 128
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    ether 28:cf:e9:1f:79:a3
    inet6 fe80::2acf:e9ff:felf:79a3%en0 prefixlen 64 scopeid 0x4
        inet 10.100.0.84 netmask 0xffffffff broadcast 10.100.0.255
            media: autoselect
            status: active
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
    ether 0a:cf:e9:1f:79:a3
    media: autoselect
    status: inactive
utun0: flags=80d1<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1406
    inet 172.28.3.185 --> 172.28.3.185 netmask 0xffffffff
```

Legend: Example of editing a file on your workstation



OPEN IN EDITOR: ~/hello_world

Hi!

I am a friendly file.

SAVE FILE!

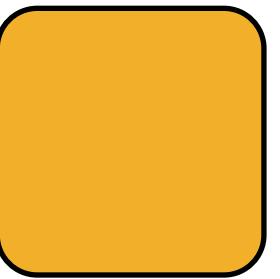
Overview of Chef

AWS PopUp Loft - A Taste of Chef v2.2.0

Lesson Objectives

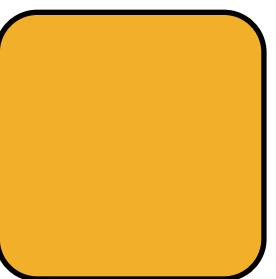
- Describe how Configuration Management applies to AWS
- Understand how Chef thinks about Infrastructure automation
- Introduce Chef terminology
 - Nodes
 - Resources
 - Recipes
 - Cookbooks
 - Run Lists
 - Roles
 - Search

A tale of Amazonian growth

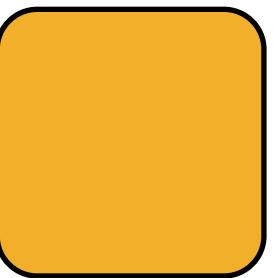


Application POC

Add a database

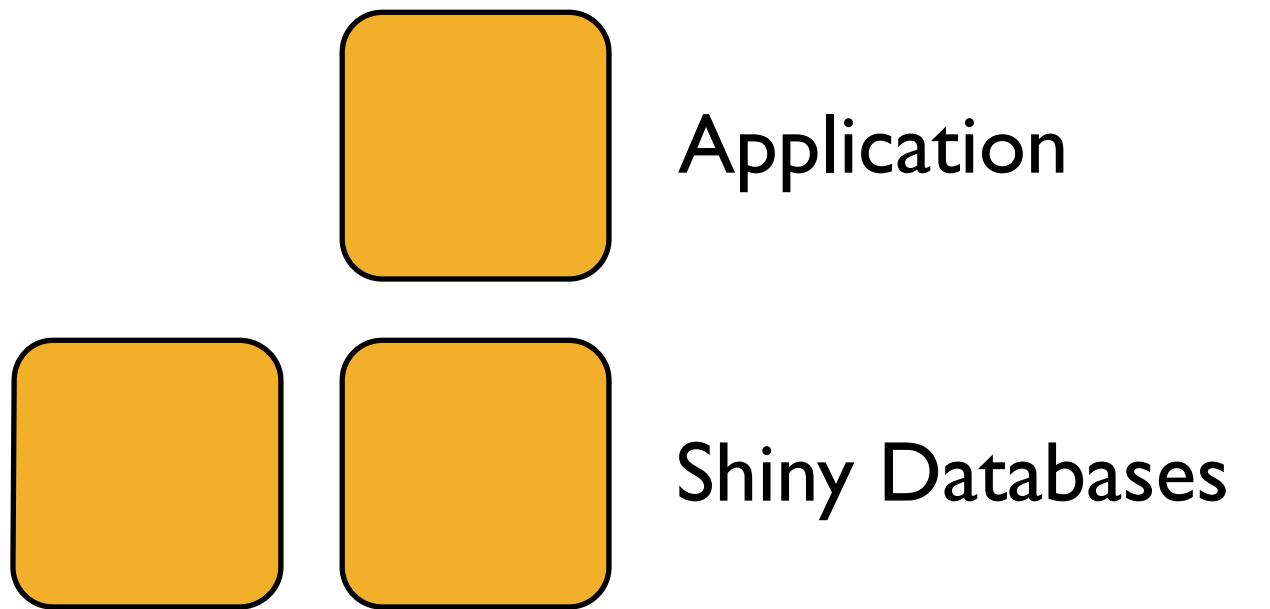


Application

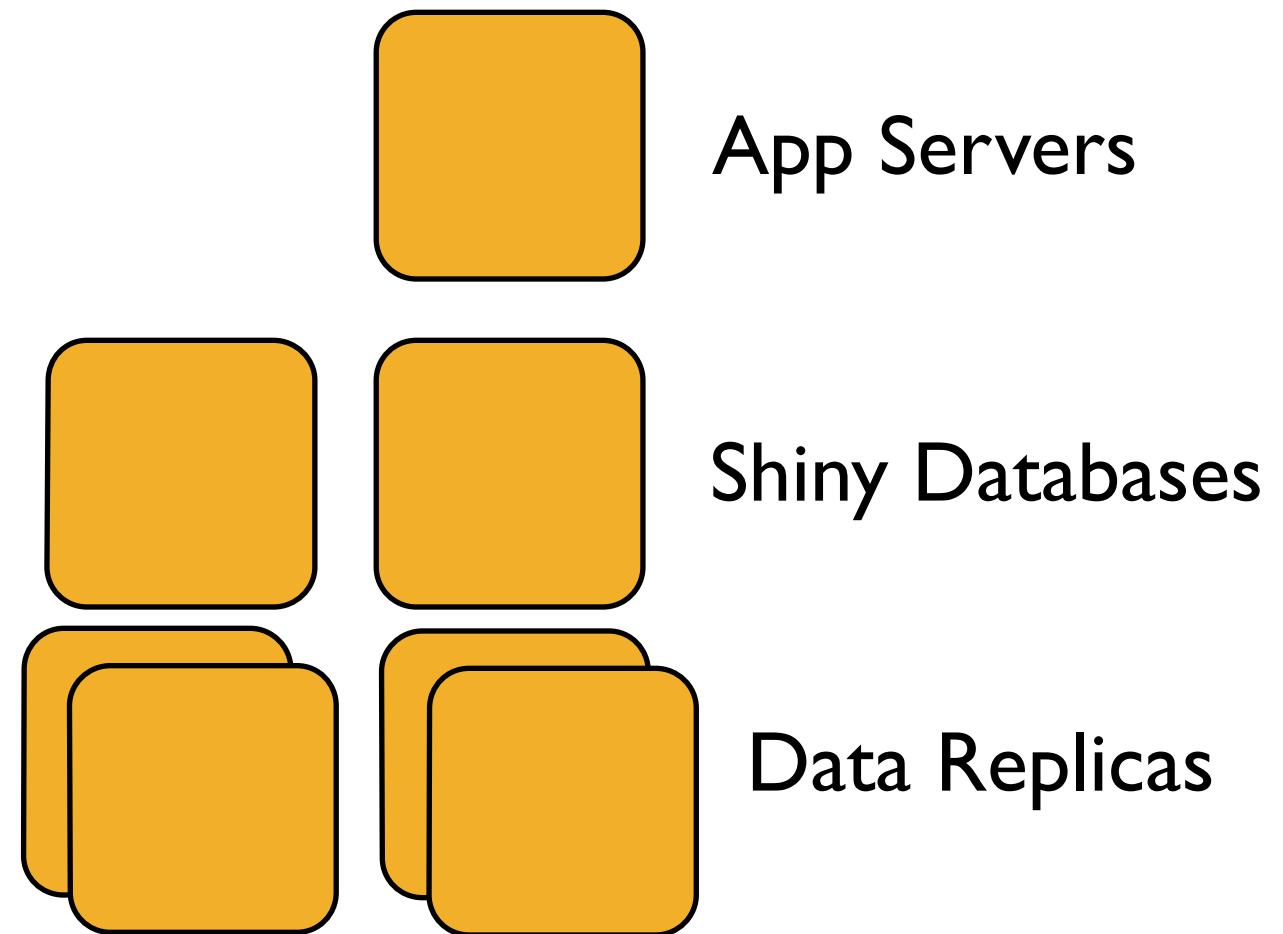


Shiny Database

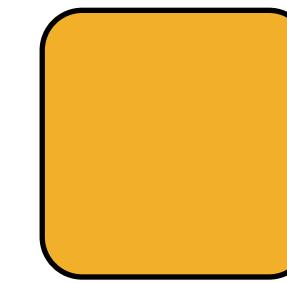
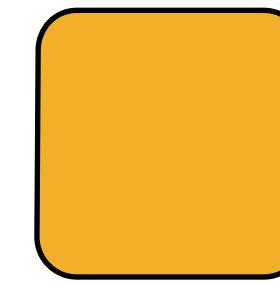
Shard your data



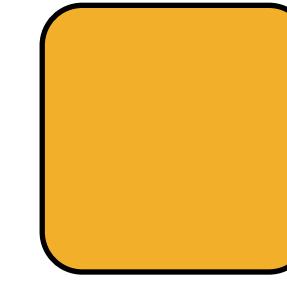
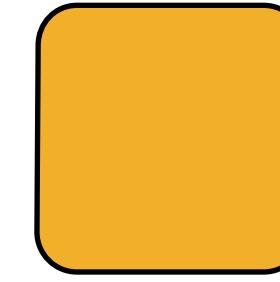
Add replica sets



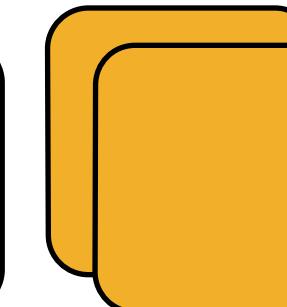
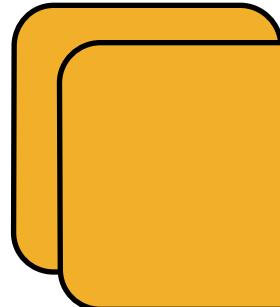
Add application servers



App Servers

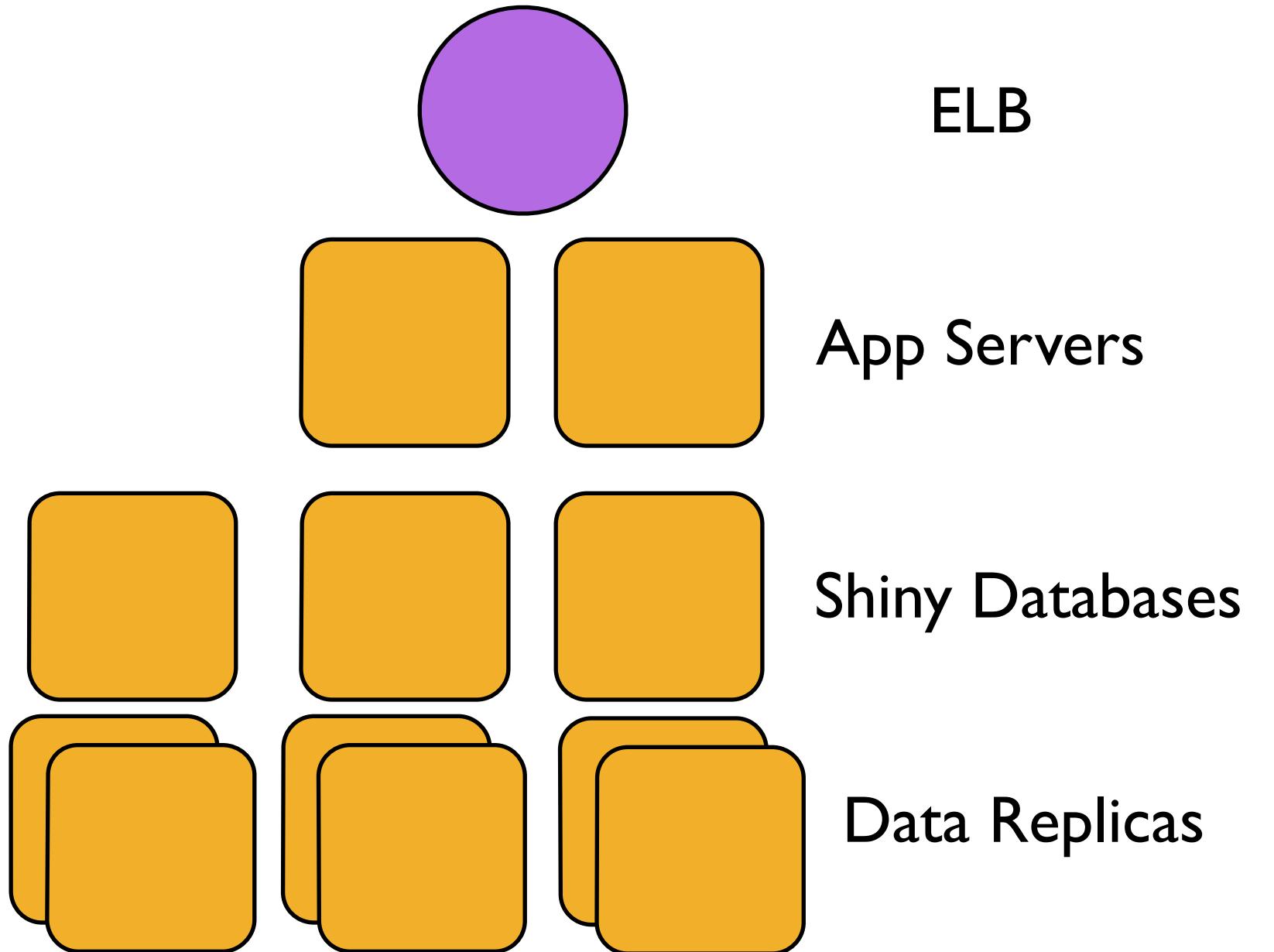


Shiny Databases

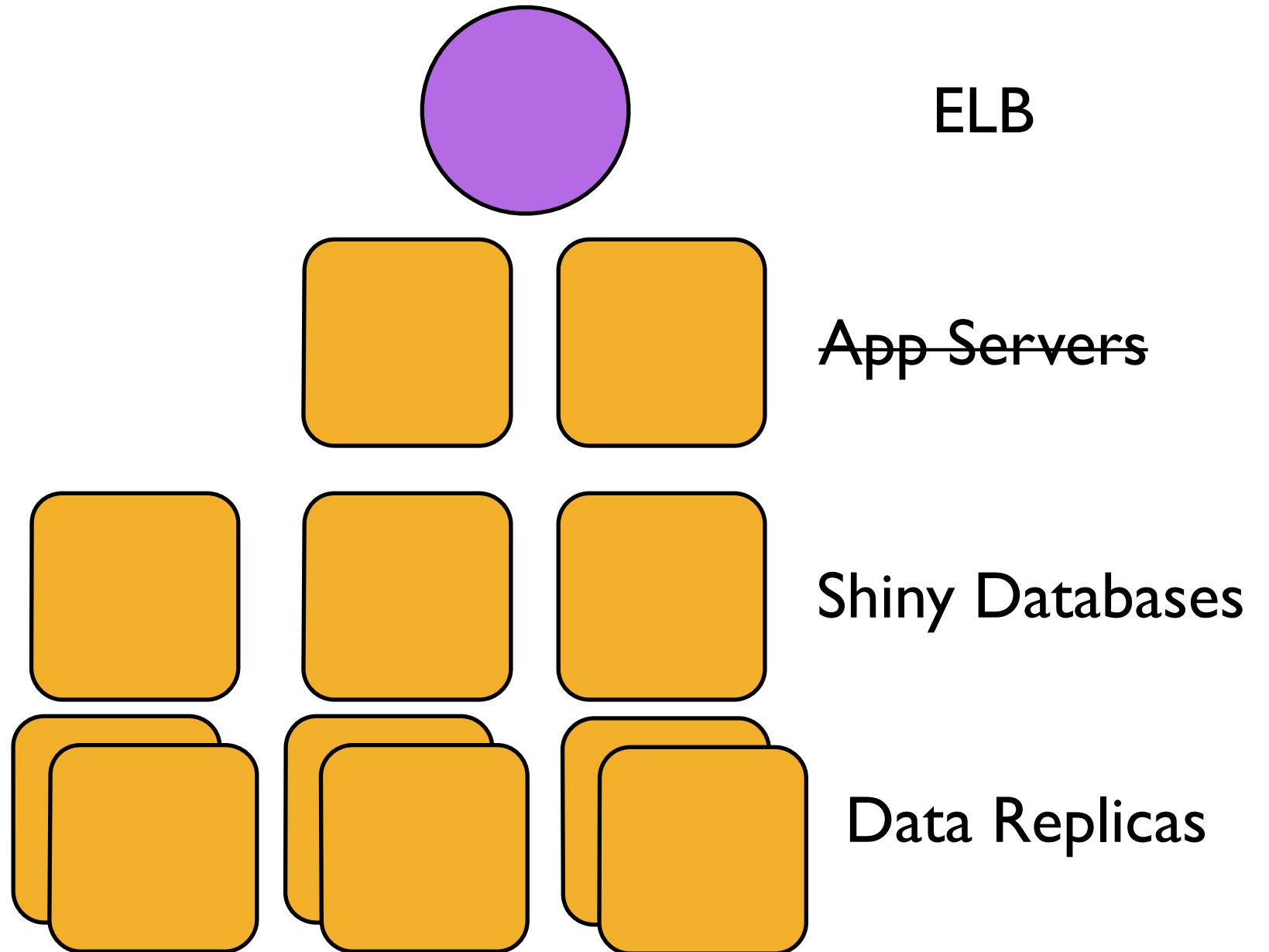


Data Replicas

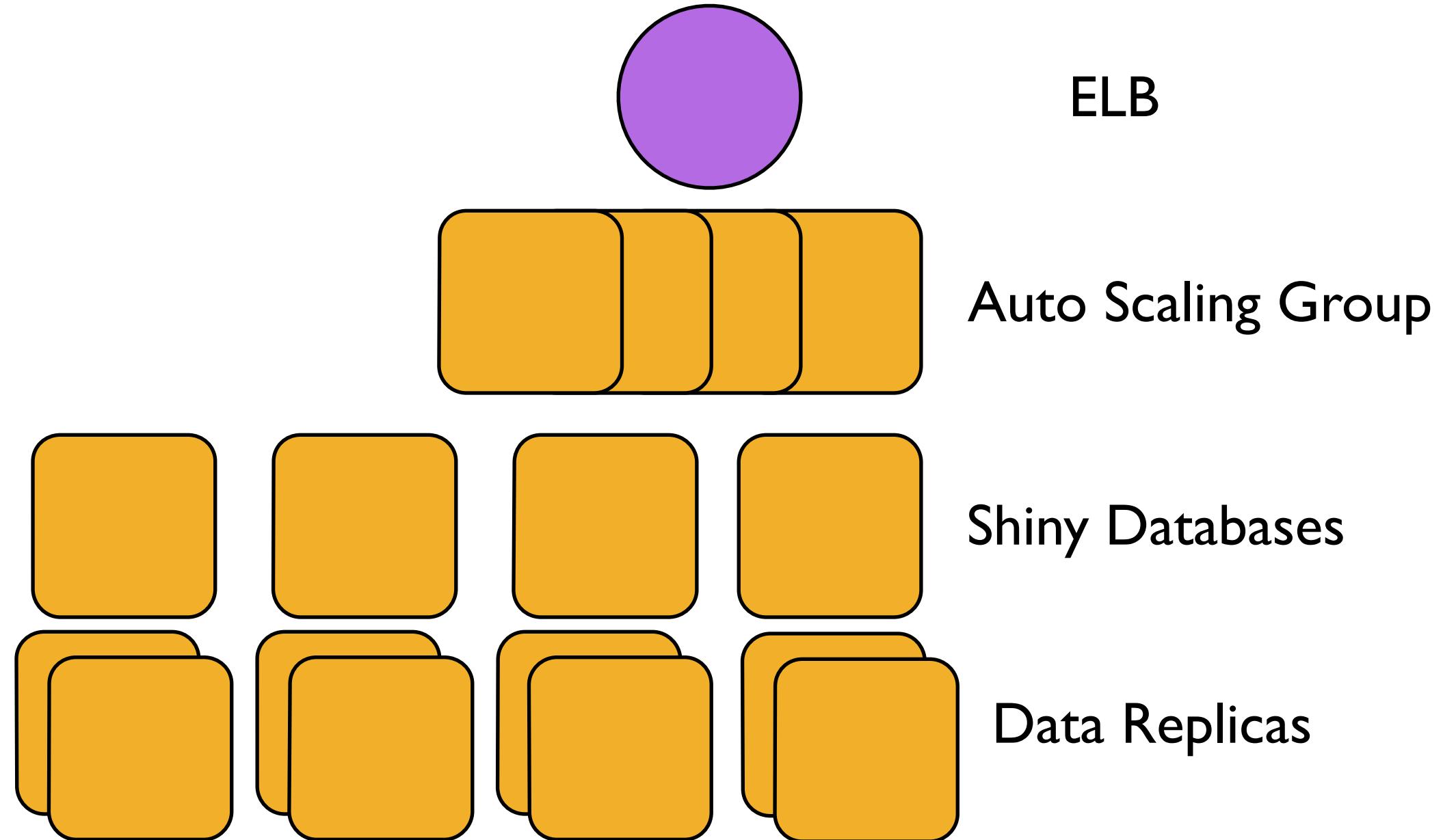
Add a load balancer



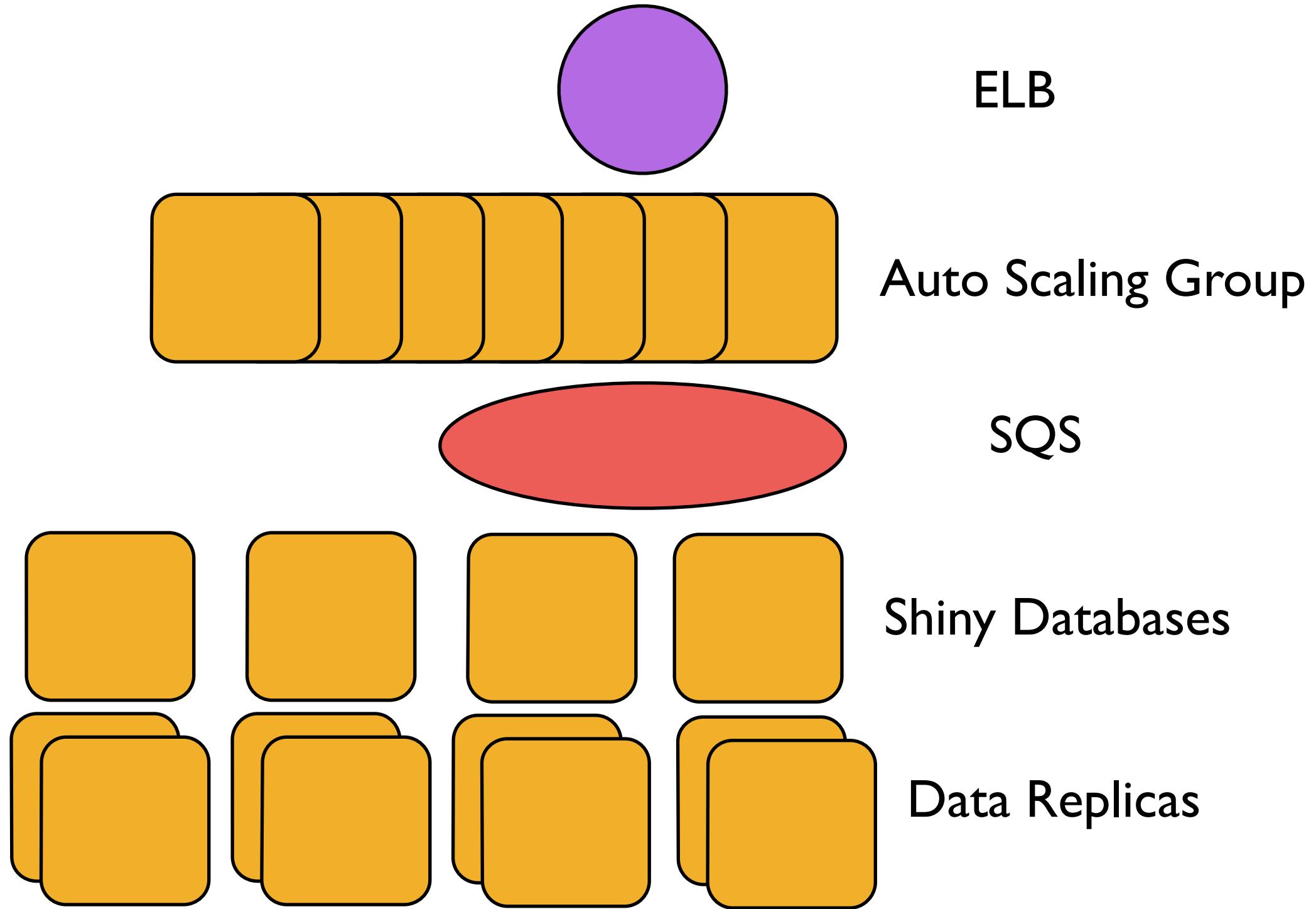
Need more horsepower



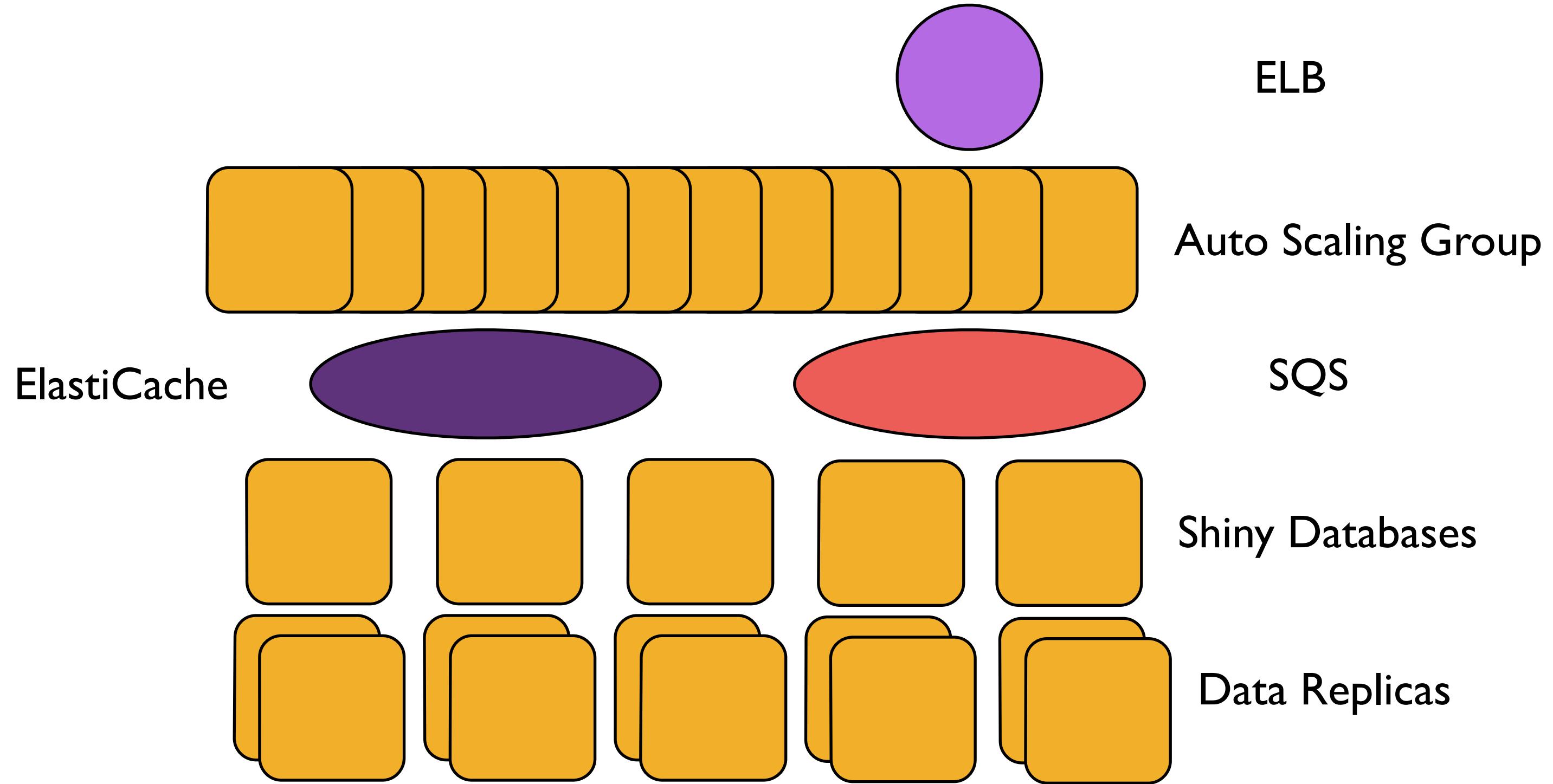
Add an autoscaling group



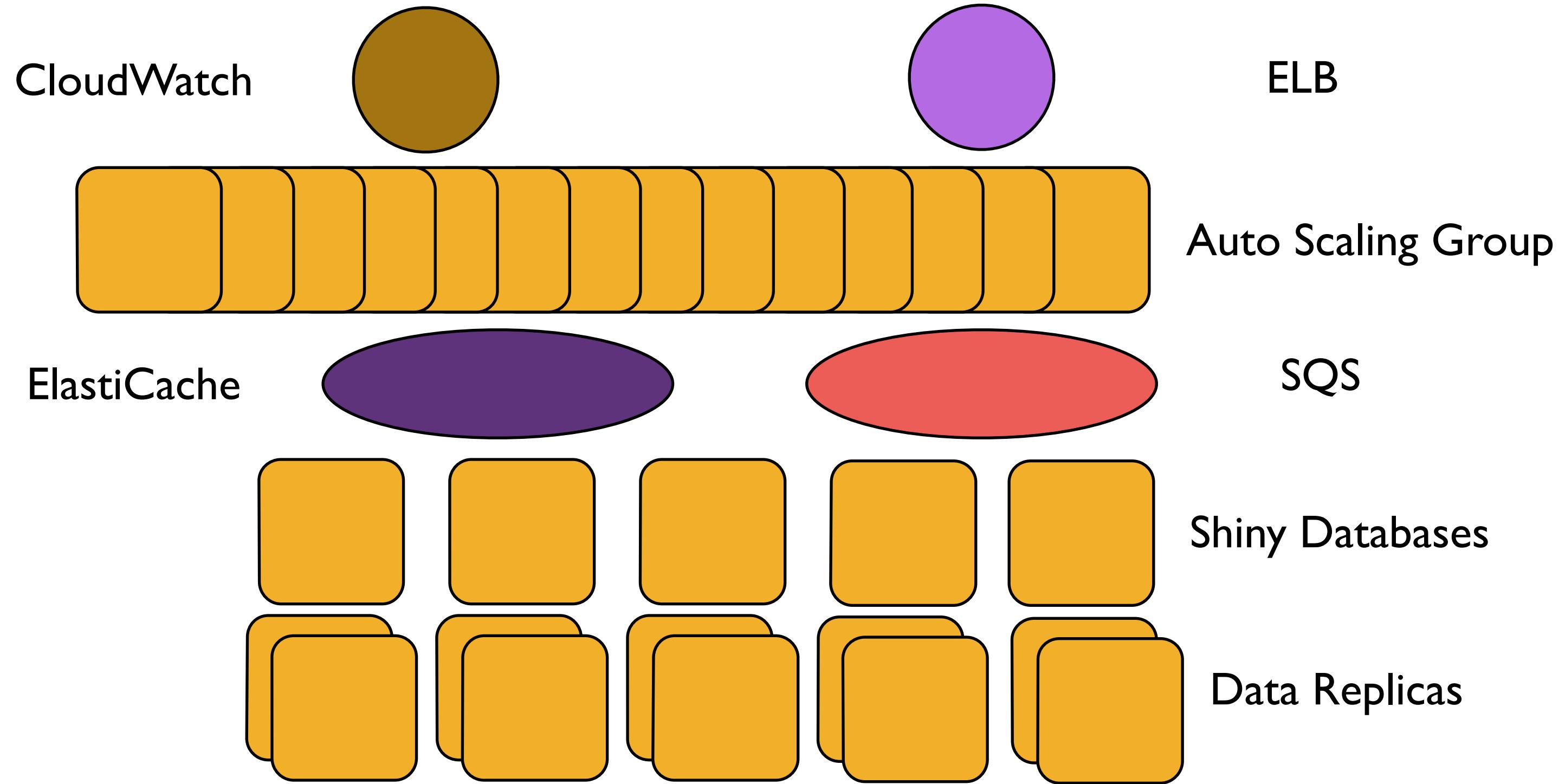
Add data resilience



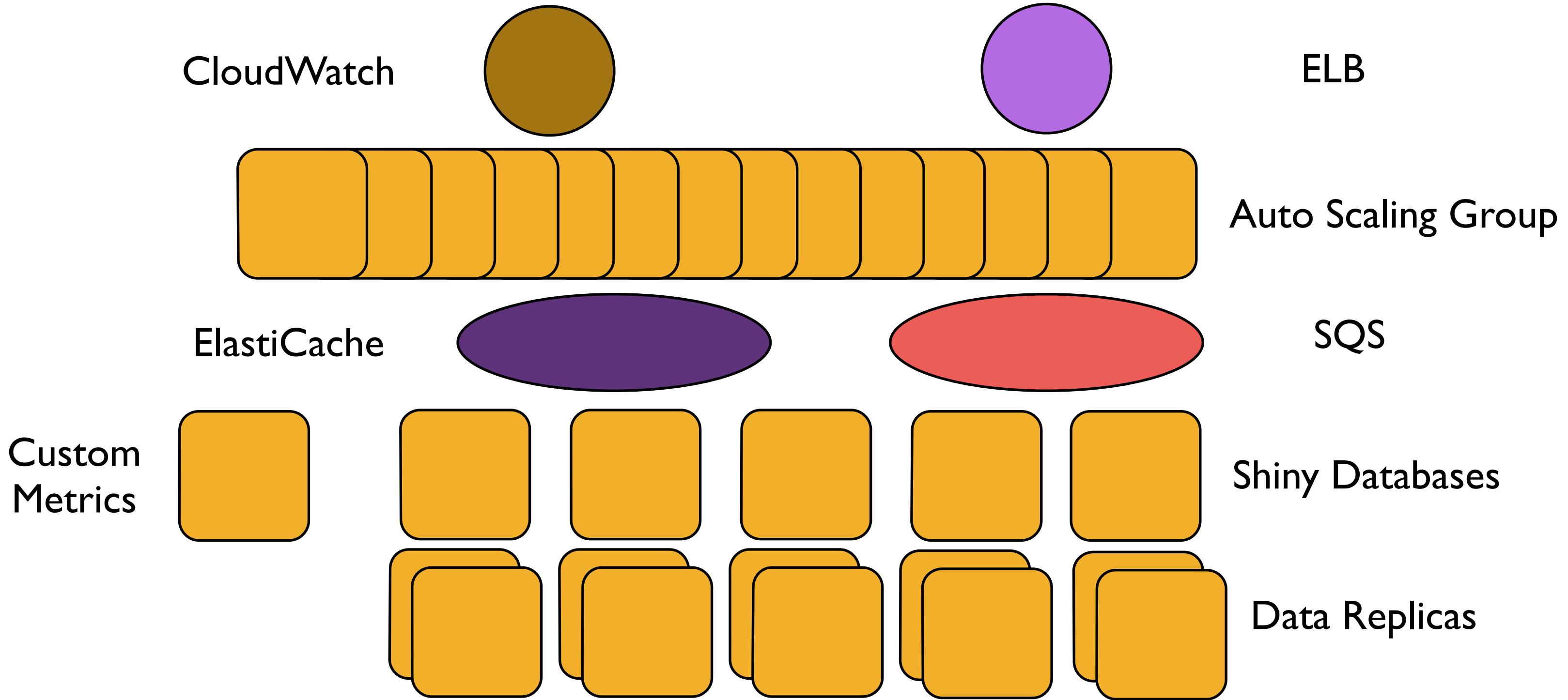
Add a caching layer



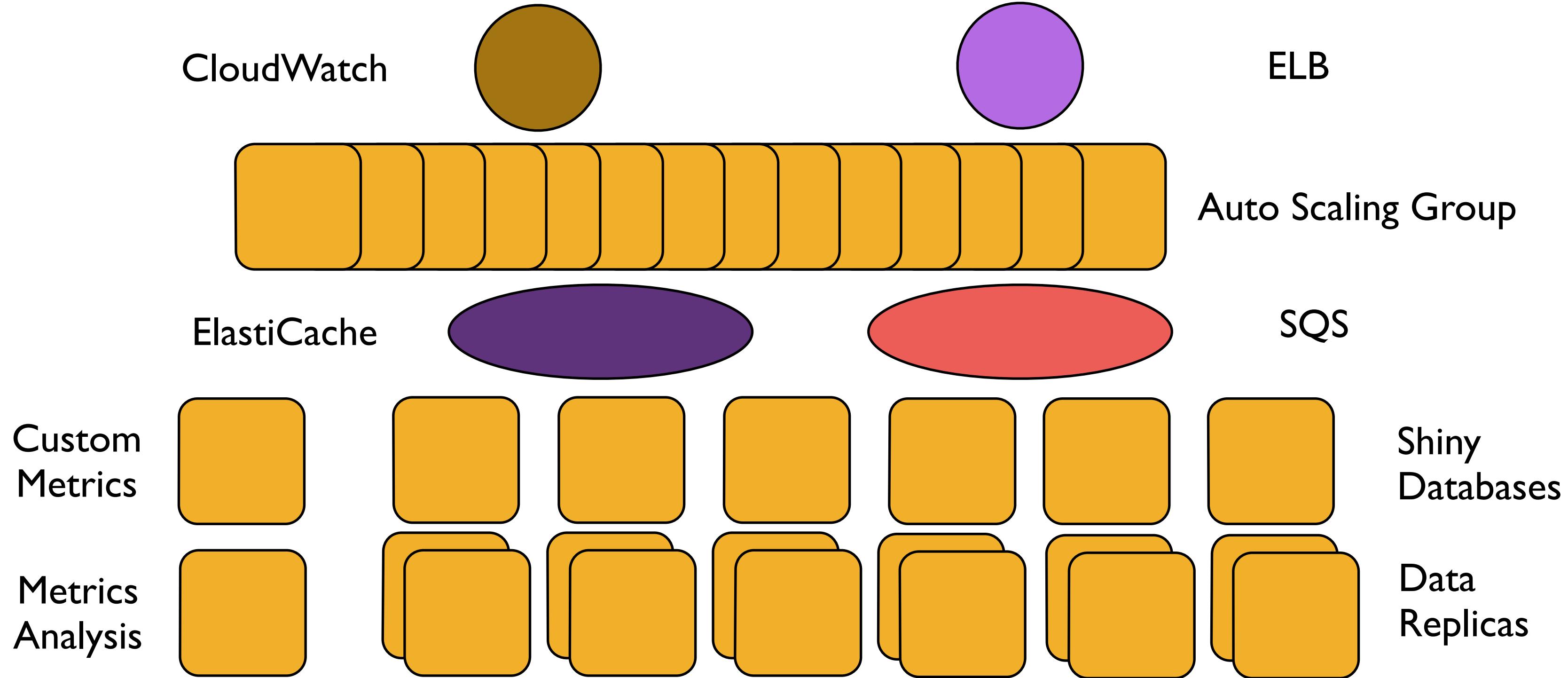
Add monitoring



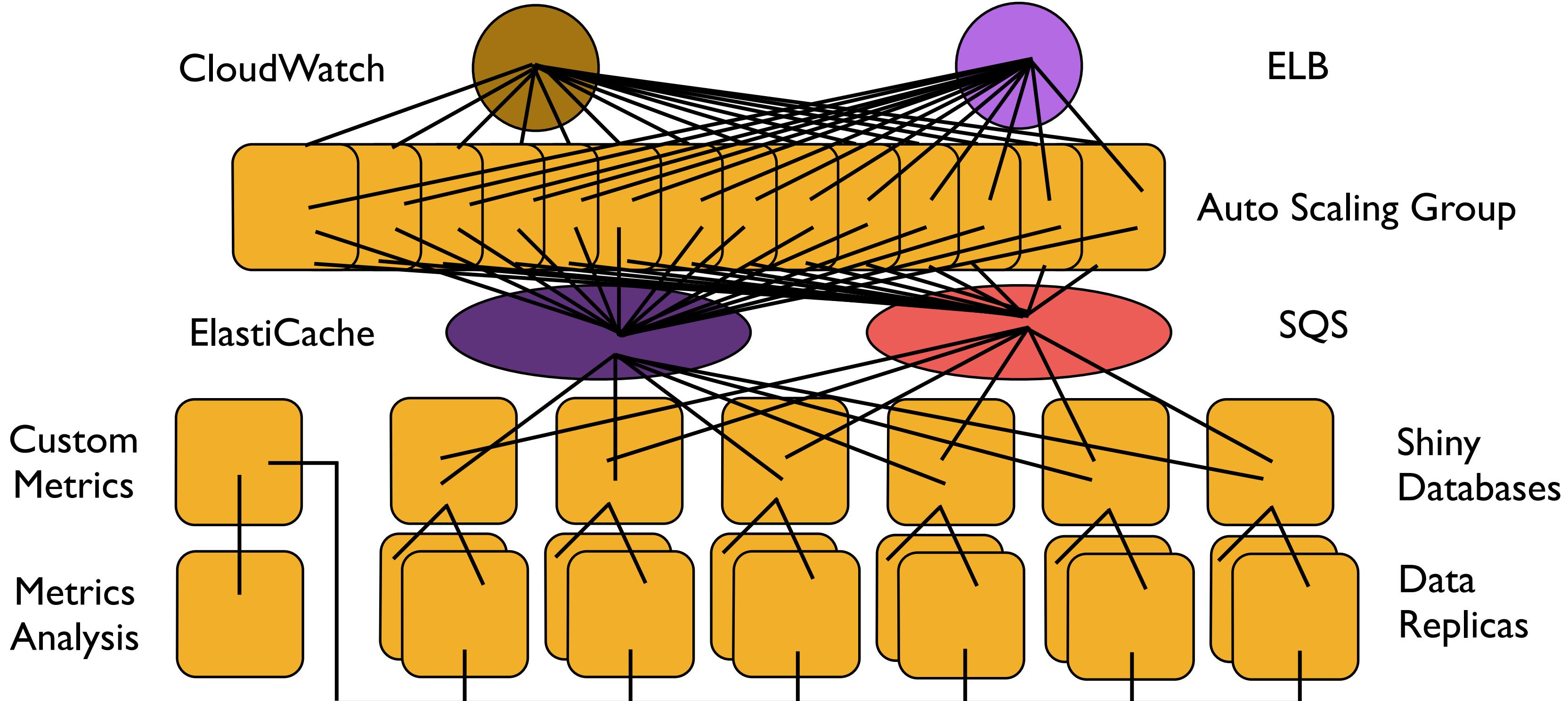
Add custom metrics



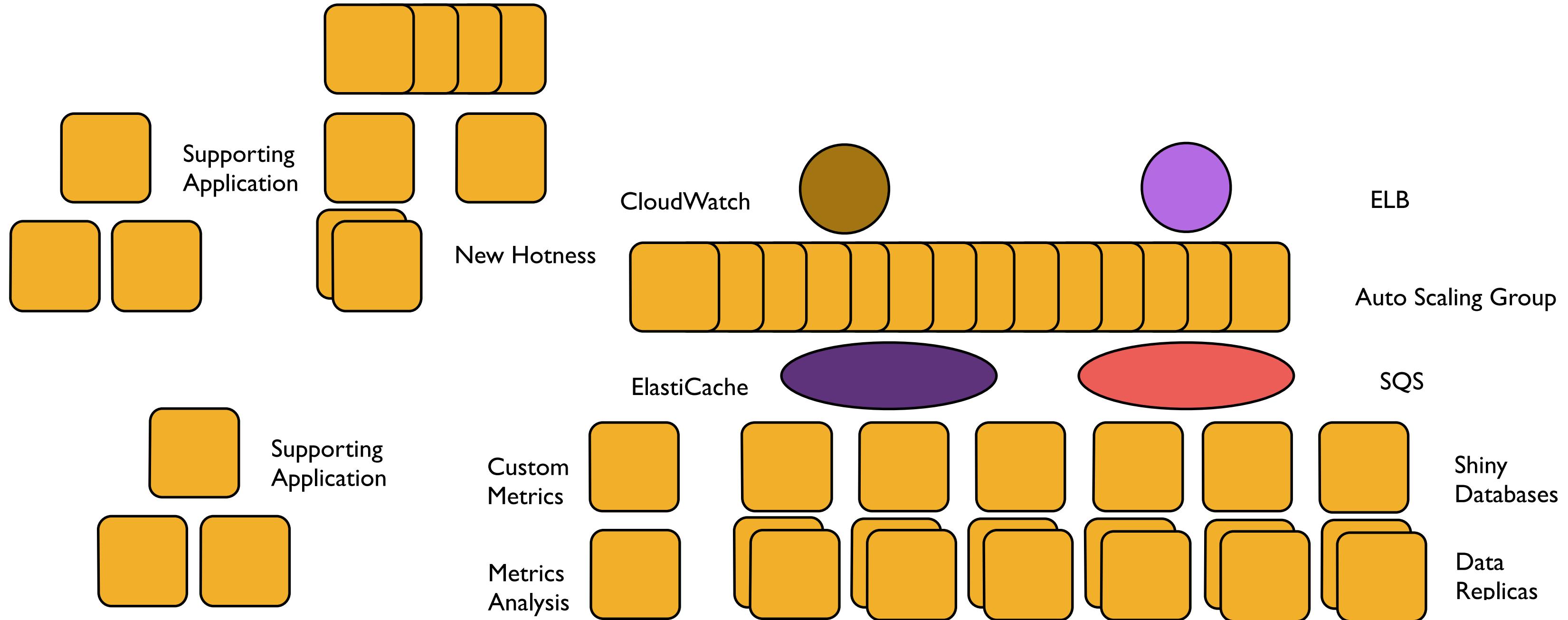
Add custom log analysis



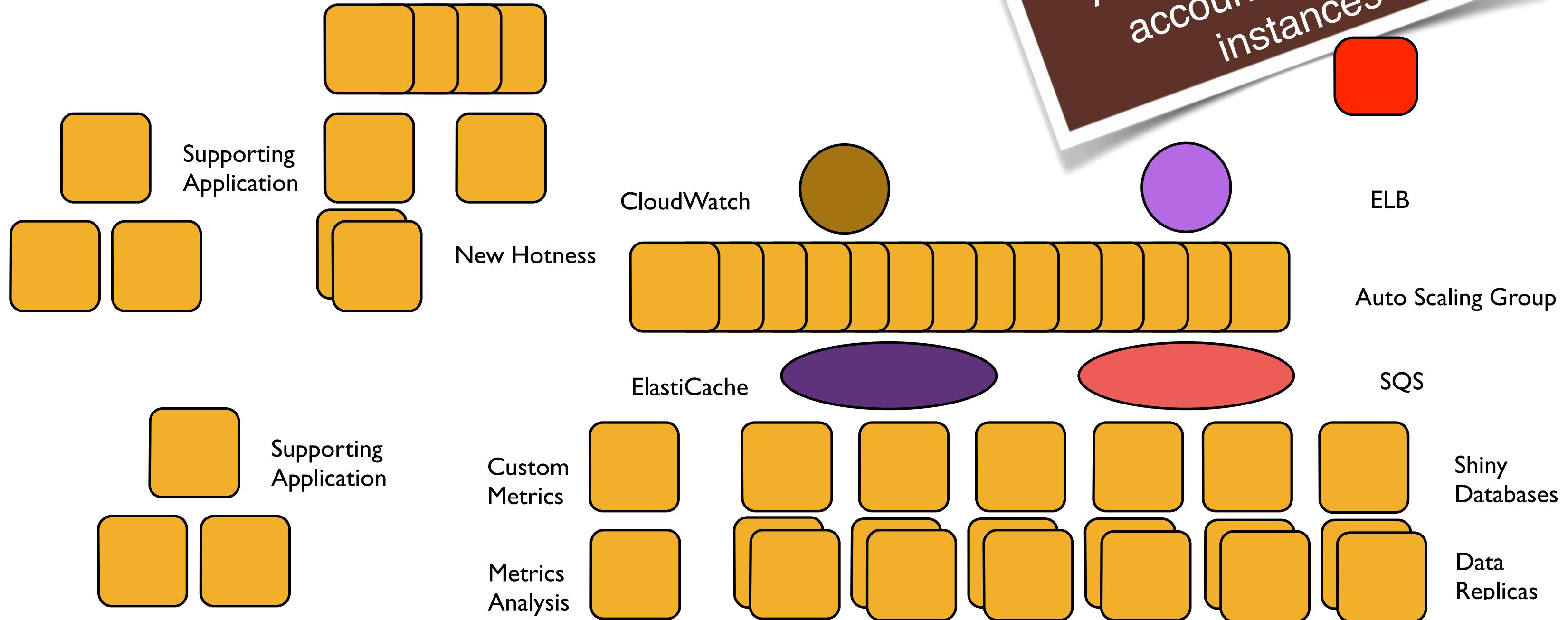
Infrastructure has a Topology



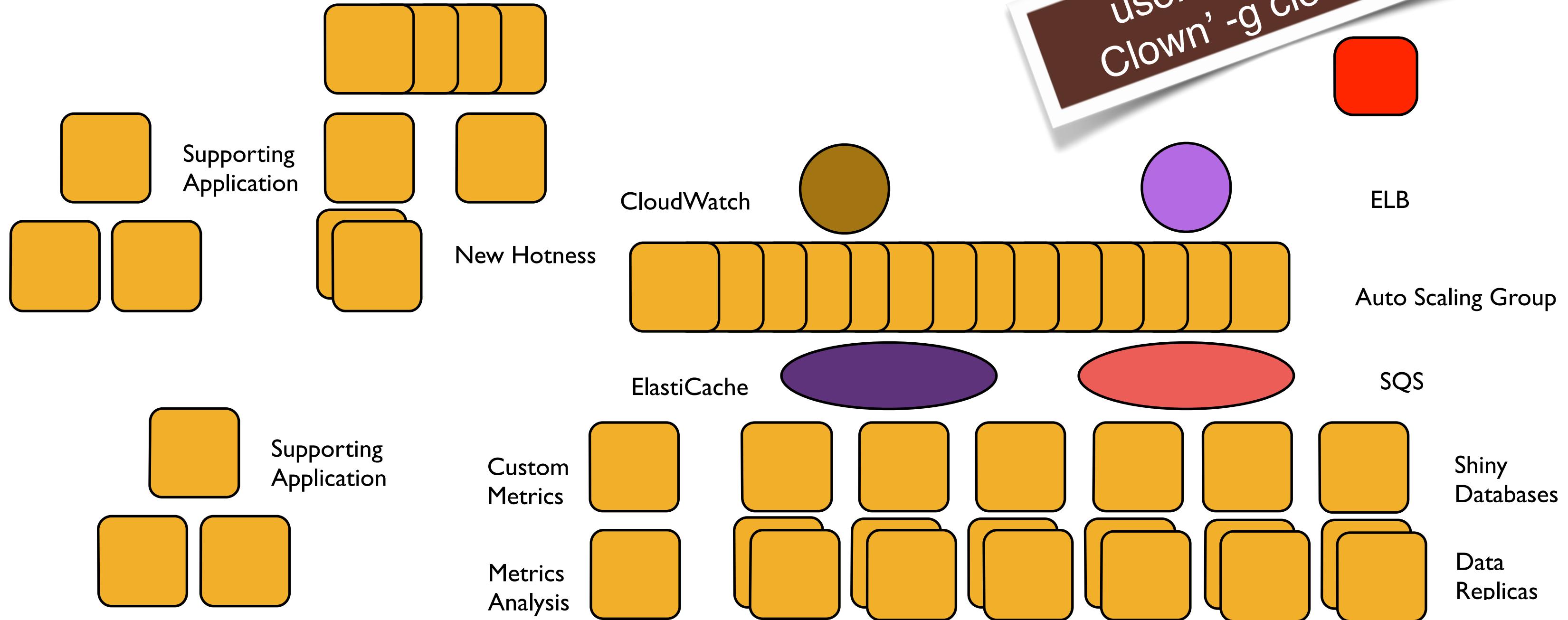
Complexity increases quickly



... and change happens!



... and change happens!



Configuration Desperation



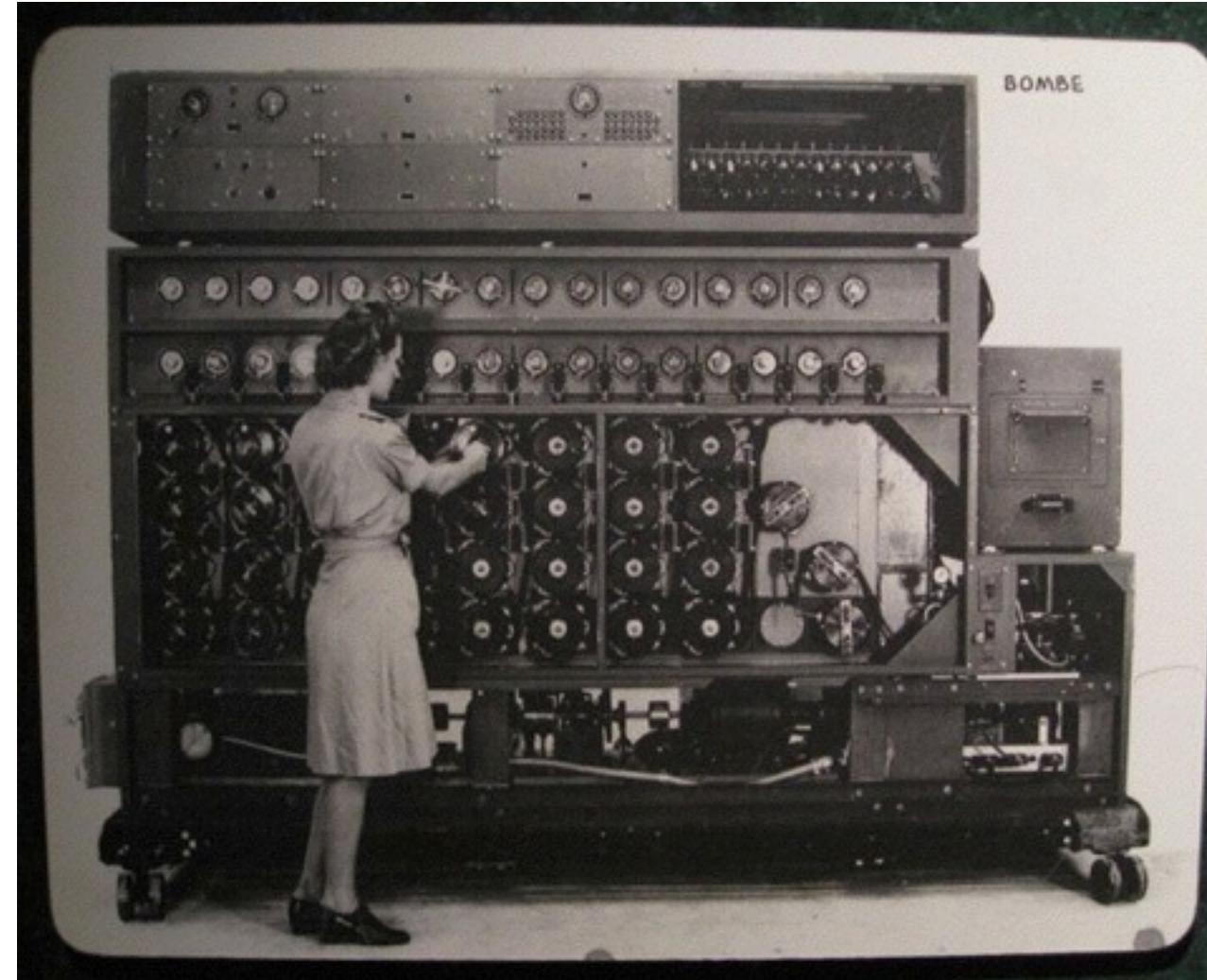
Chef Solves This Problem



CHEFTM

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

Chef is Infrastructure as Code



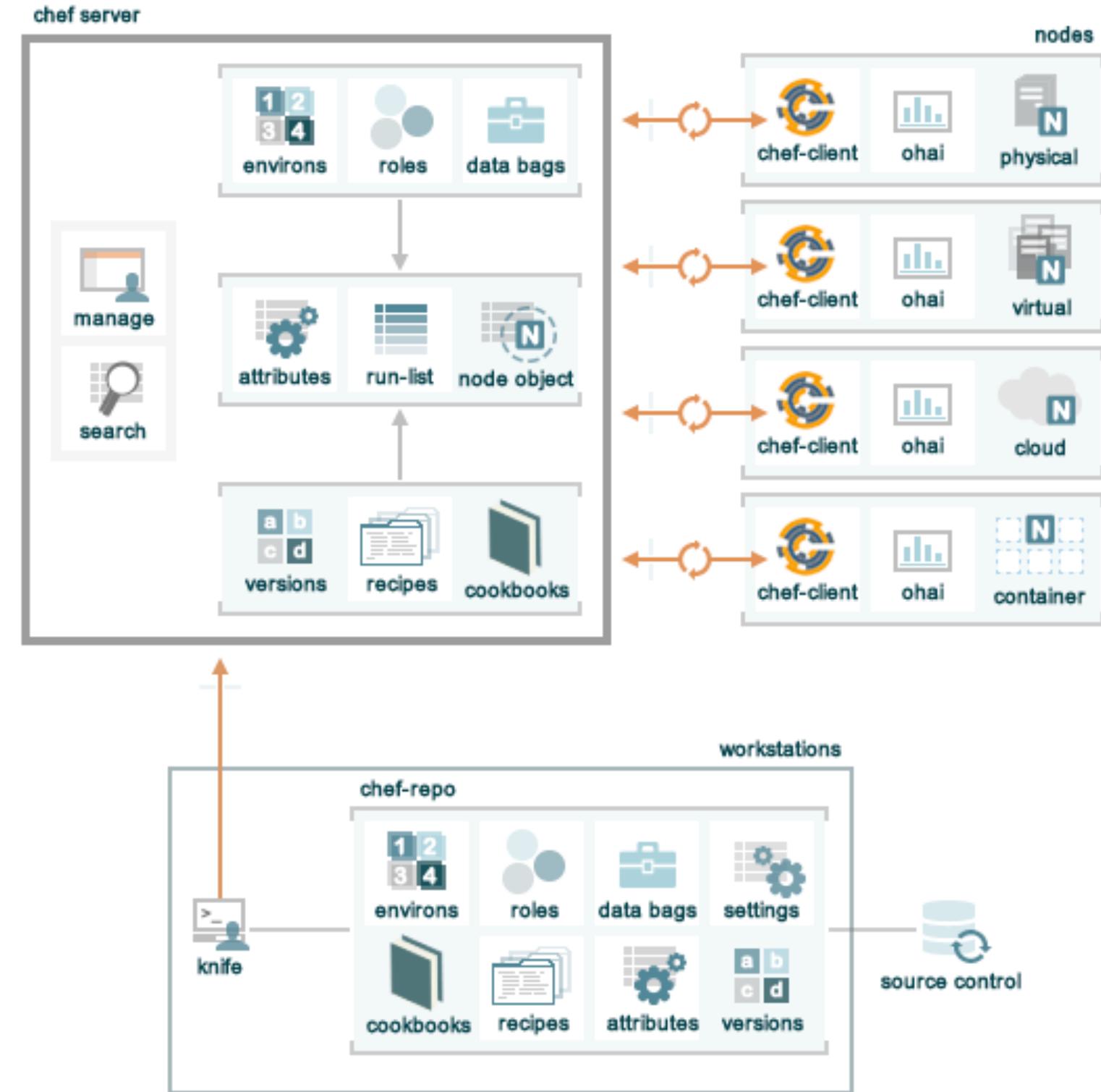
<http://www.flickr.com/photos/louisb/4555295187/>

- Programmatically provision and configure servers
- Treat like any other code base
- Reconstruct business from **code repository, data backup, and compute resources**

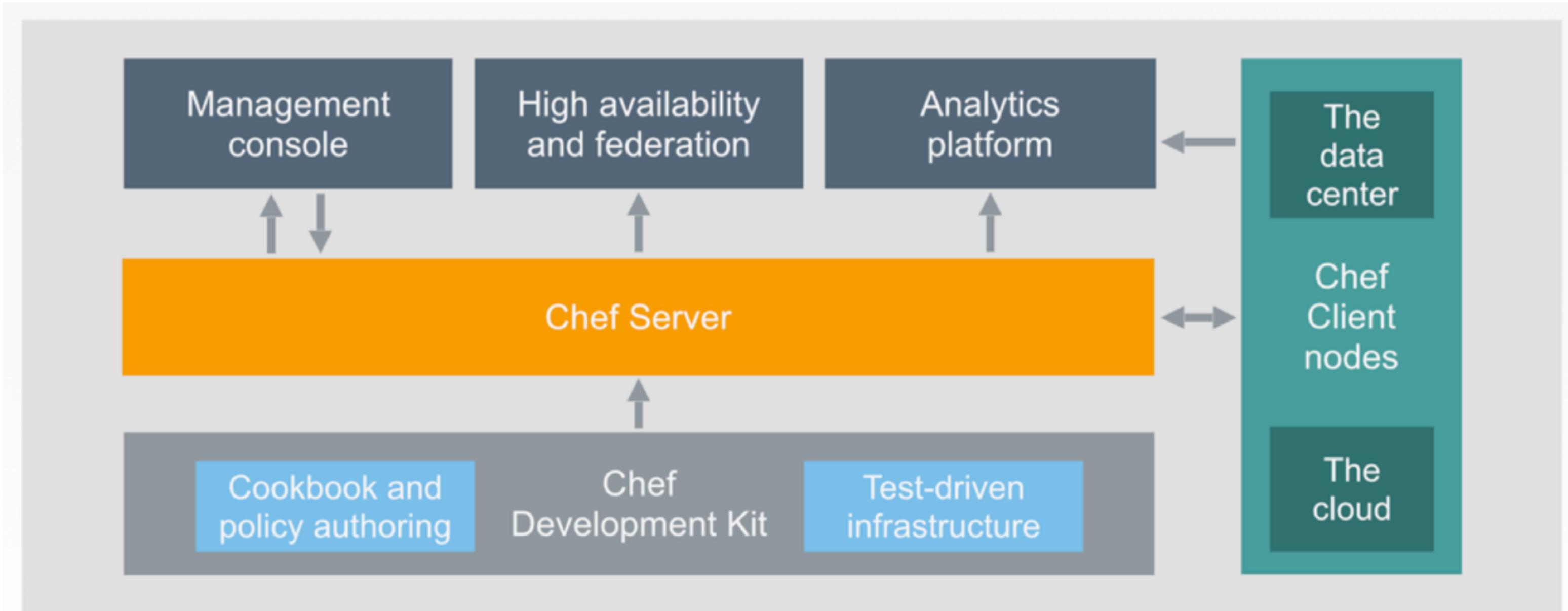
Desired-State Management

- You capture the desired state of your infrastructure in code
- Chef ensures each node in your infrastructure complies with the desired state
- Chef provides a domain-specific language (DSL) that allows you to specify the desired state of your infrastructure
- Desired states can be statically or dynamically defined

Chef Components



Chef Server Functions



Nodes

- For this class: nodes represent your ec2 instances
 - Could be non-ec2 virtual servers
 - Might be hardware that you own or compute instances in any public or private cloud
 - Support for Containers
 - Support for physical network switches

Resources

- The fundamental building blocks of Chef
- Describes a piece of the system and its desired state
 - A package that should be installed
 - A service that should be running
 - A file that should be generated
 - A cron job that should be configured
 - A user that should be managed
 - and more

Recipes

- A collection of resources
- Describe configuration state you want to apply
- Recipes can:
 - Install and configure software components
 - Manage file content
 - Deploy applications
 - Execute other recipes
 - and more

Example resources

```
package "apache2"

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

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

Recipes: a collection of resources

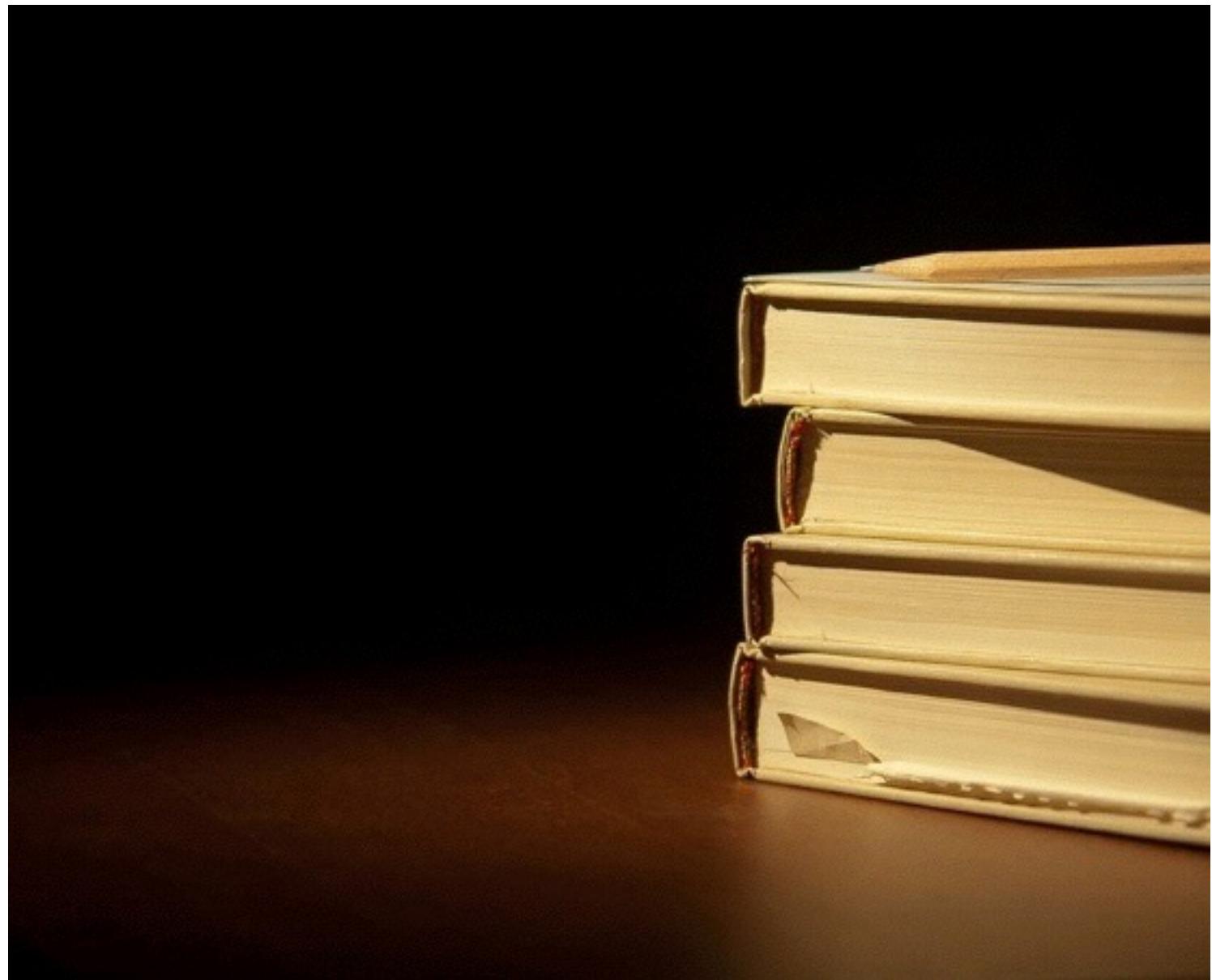
```
package "apache2"

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

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

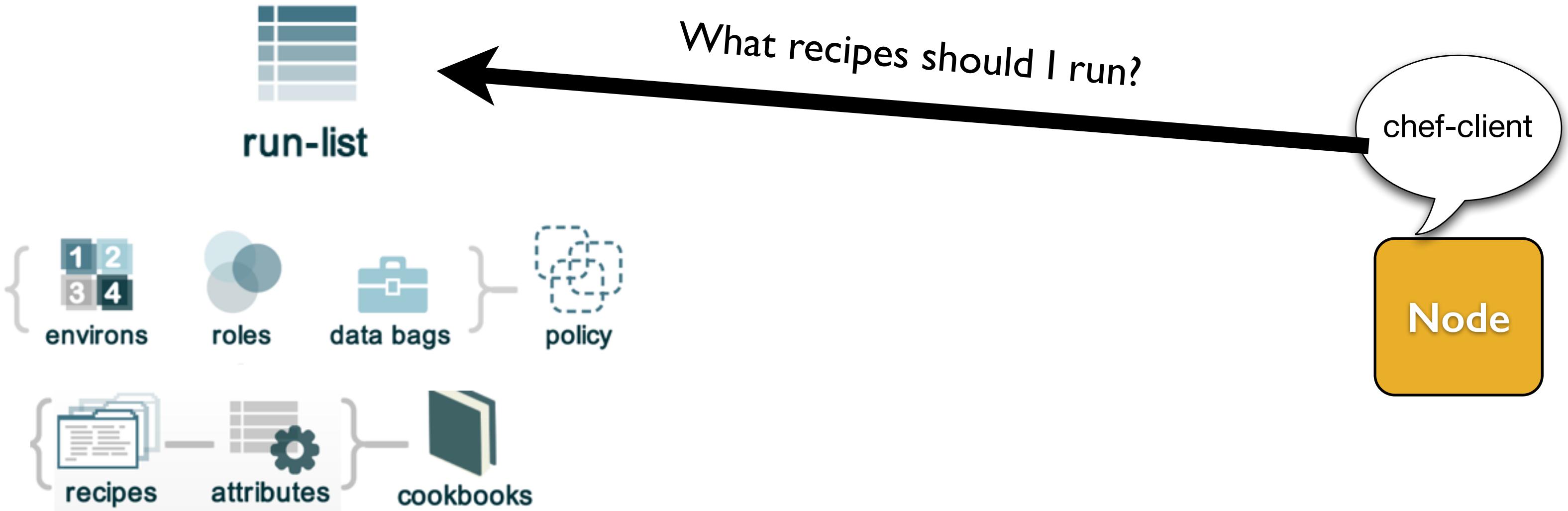
Cookbooks

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

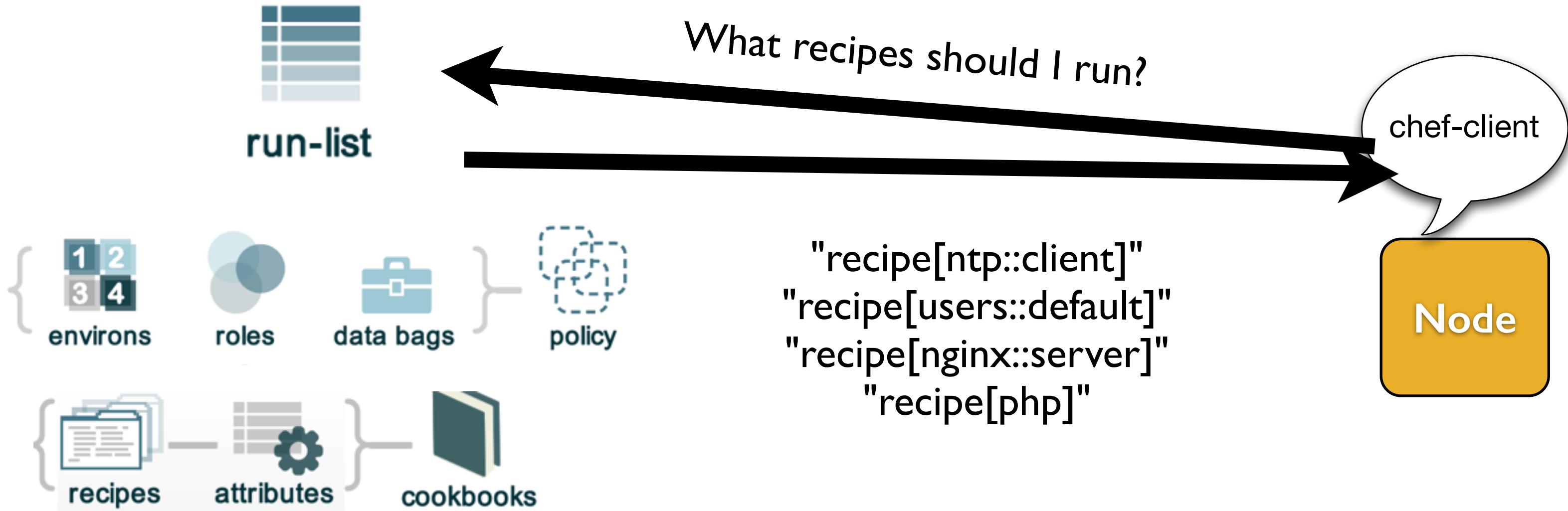


<http://www.flickr.com/photos/shutterhacks/4474421855/>

Nodes have a Run List

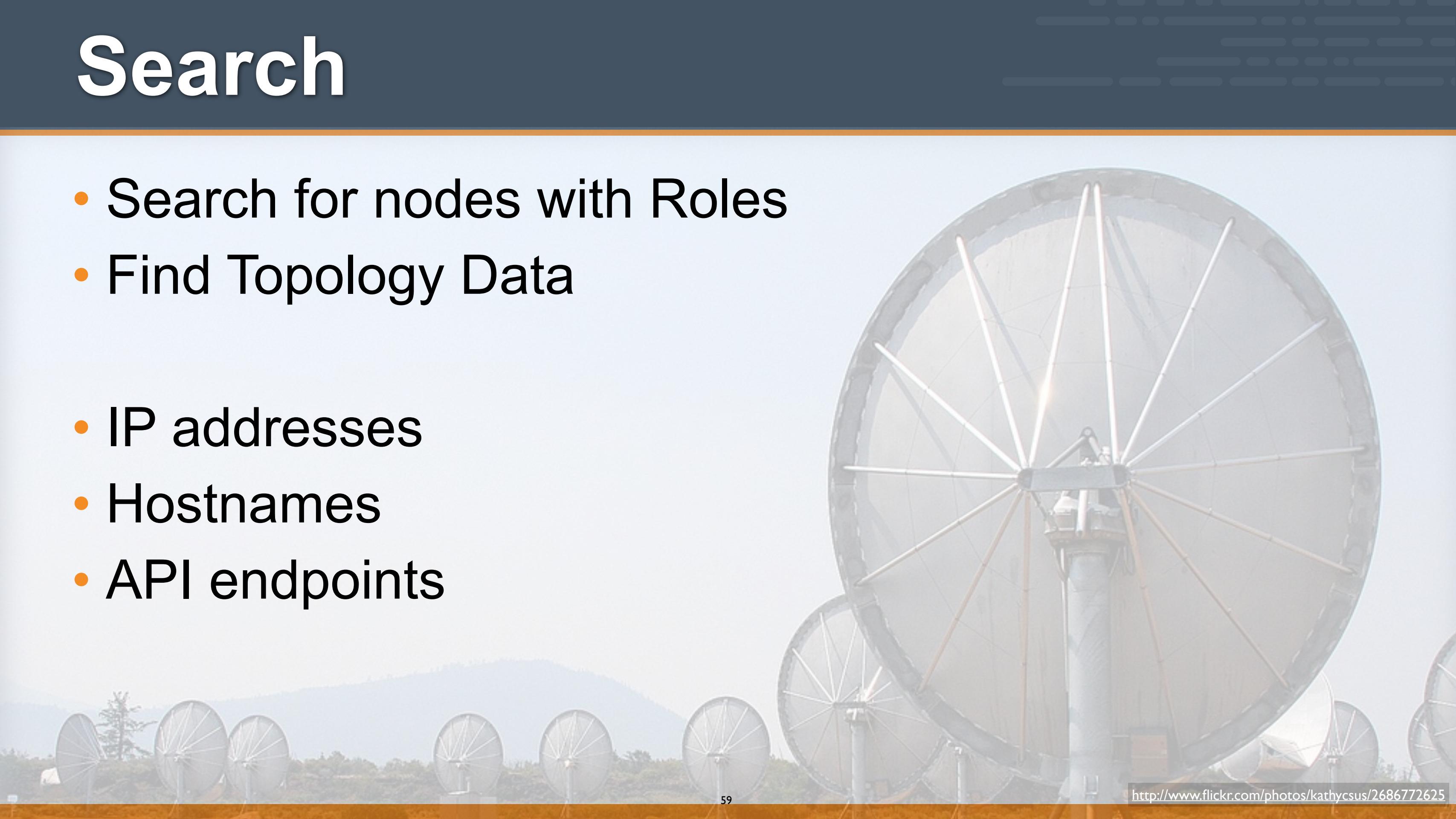


Run Lists: Desired State to apply



Search

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

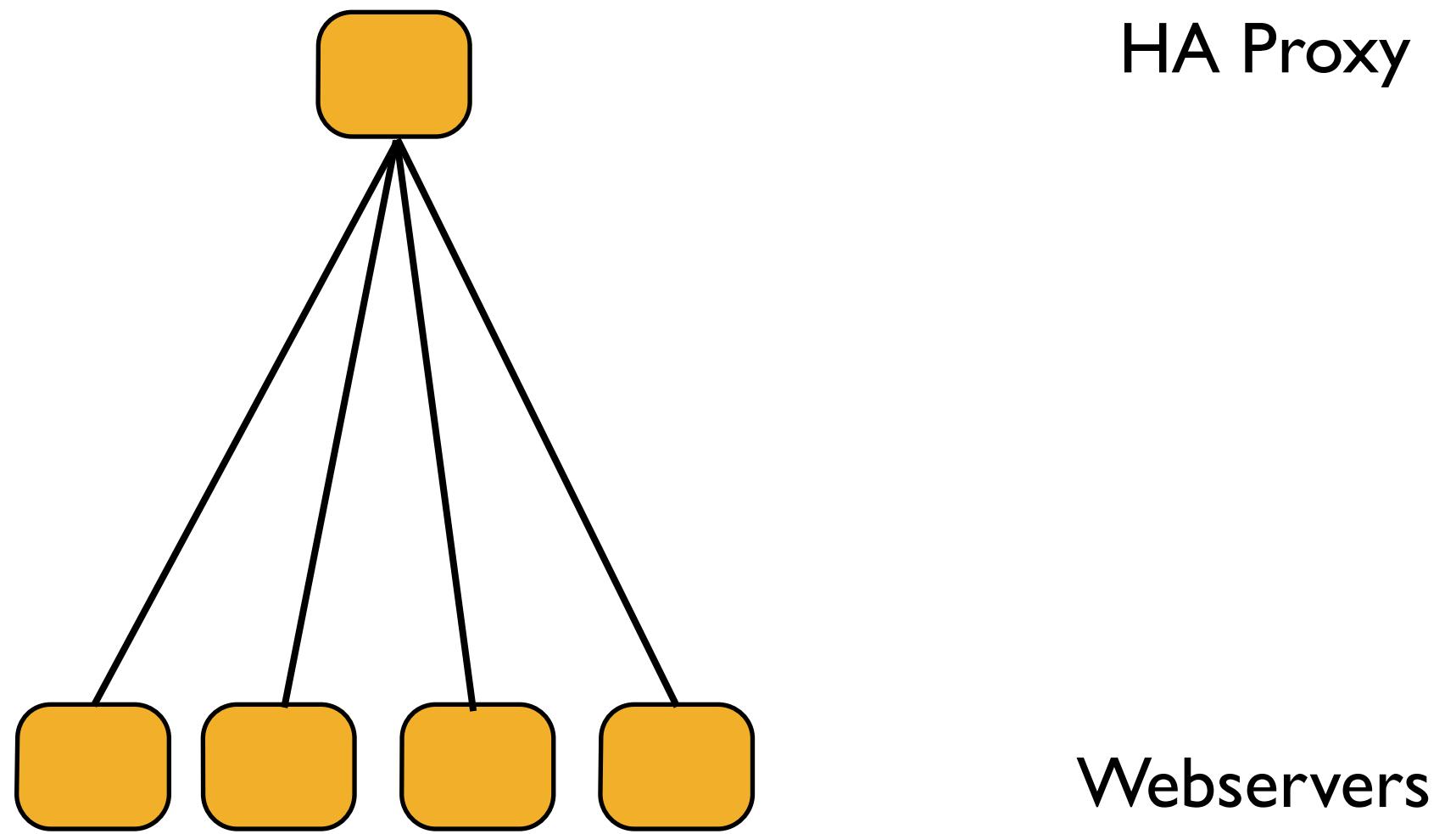


Search for Nodes

```
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
```

HAProxy Configuration

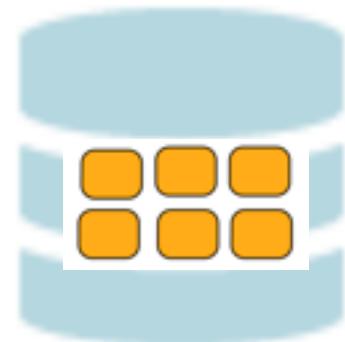


HAProxy Load Balancer

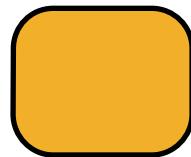
```
pool_members = search("node", "role:webserver")
```



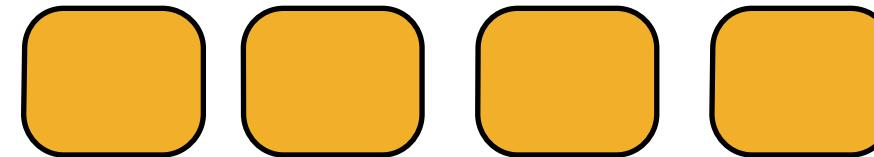
Chef Server



Search Index



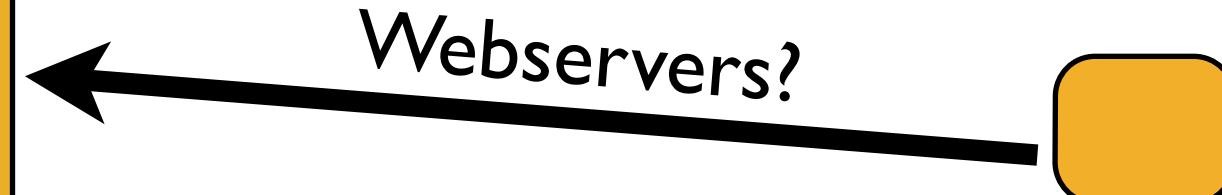
HA Proxy



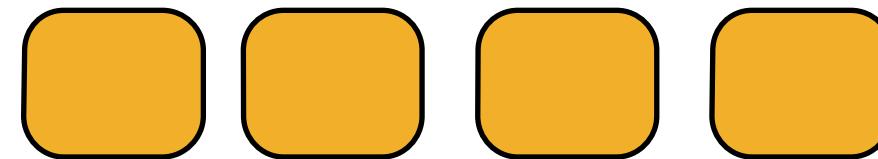
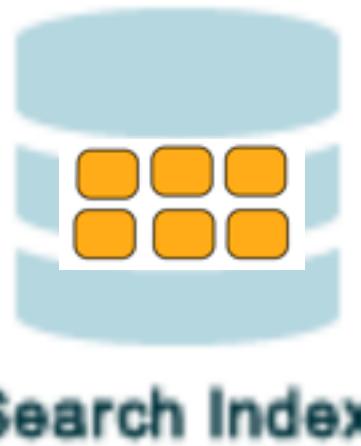
Webservers

HAProxy Load Balancer

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



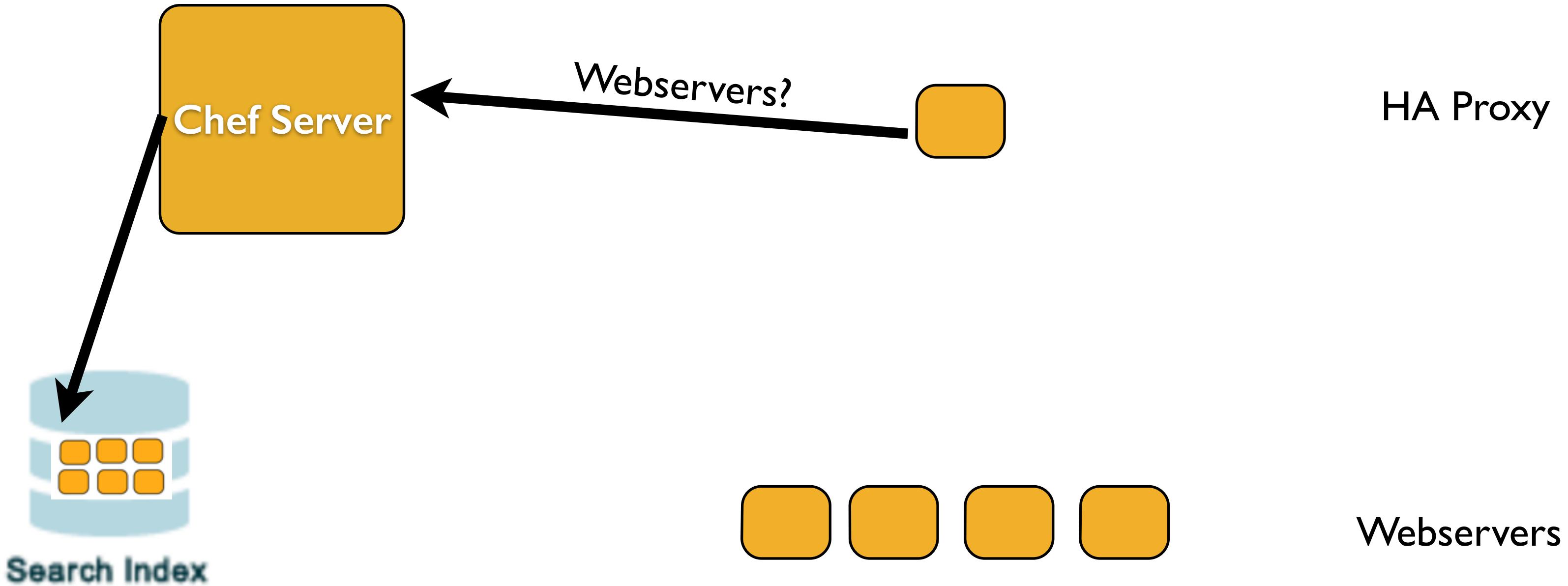
HA Proxy



Webservers

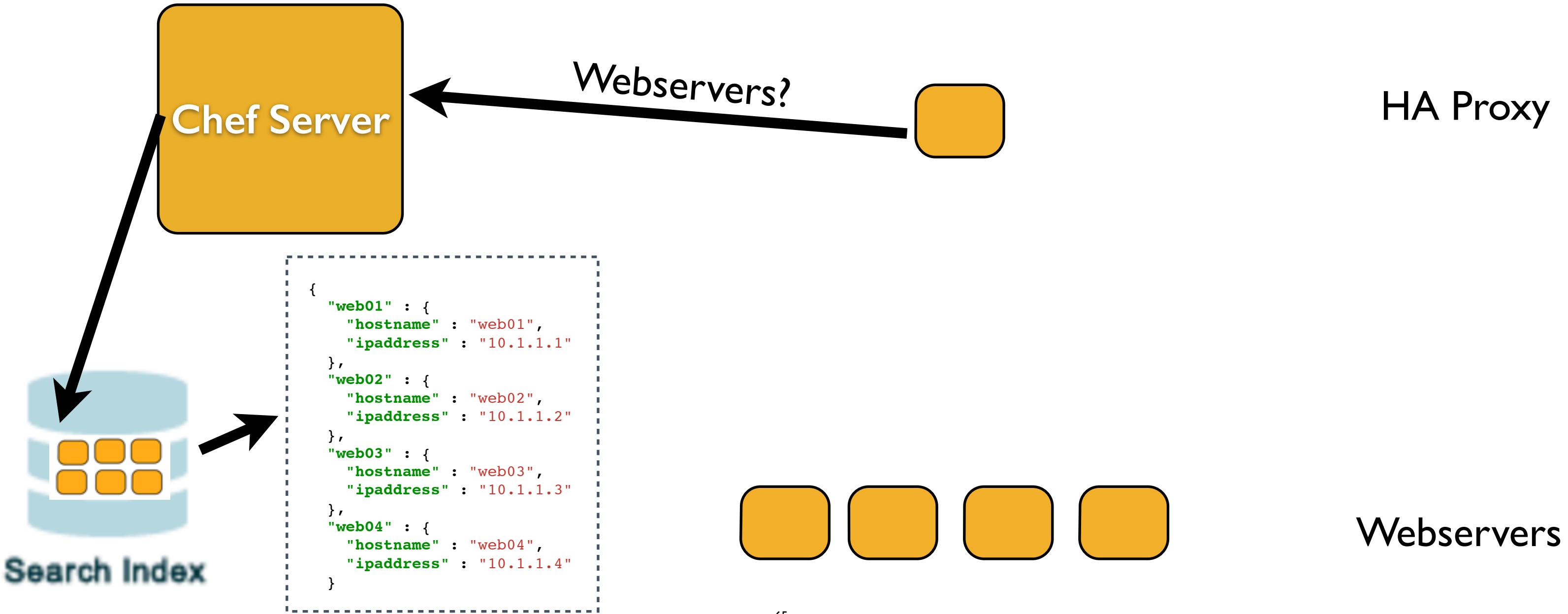
HAProxy Load Balancer

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



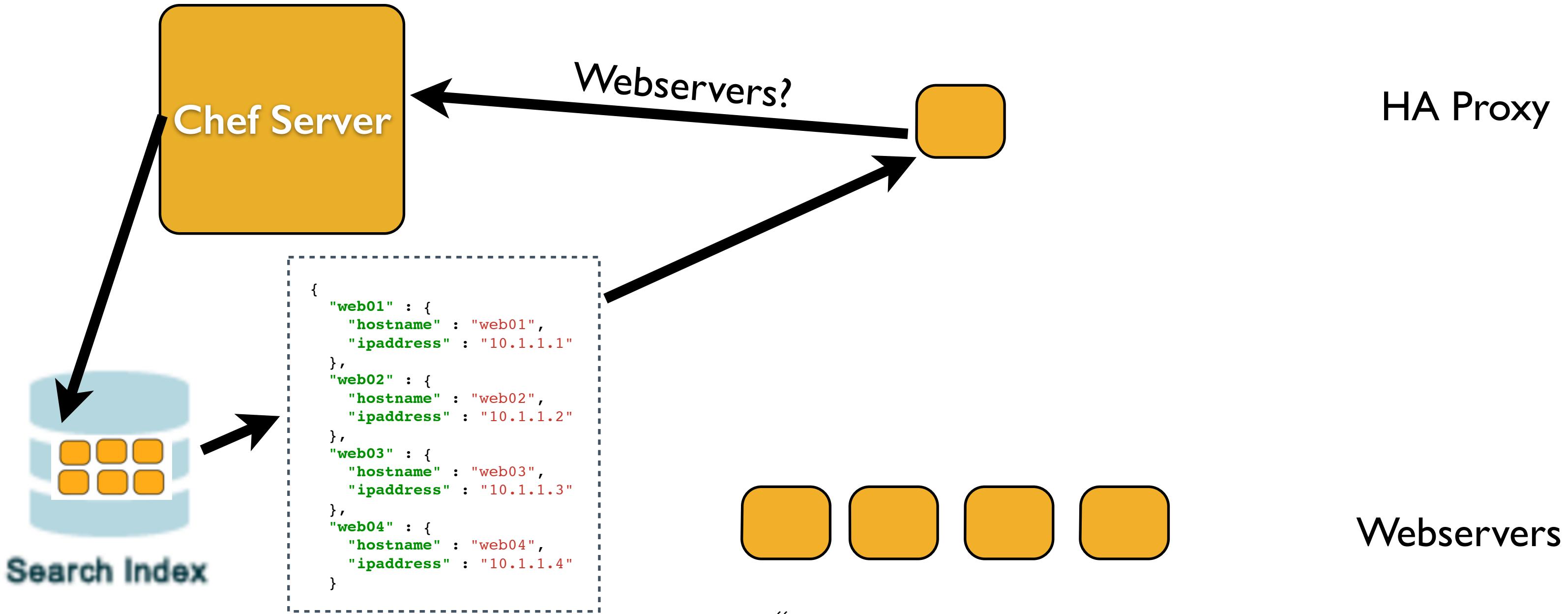
HAProxy Load Balancer

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



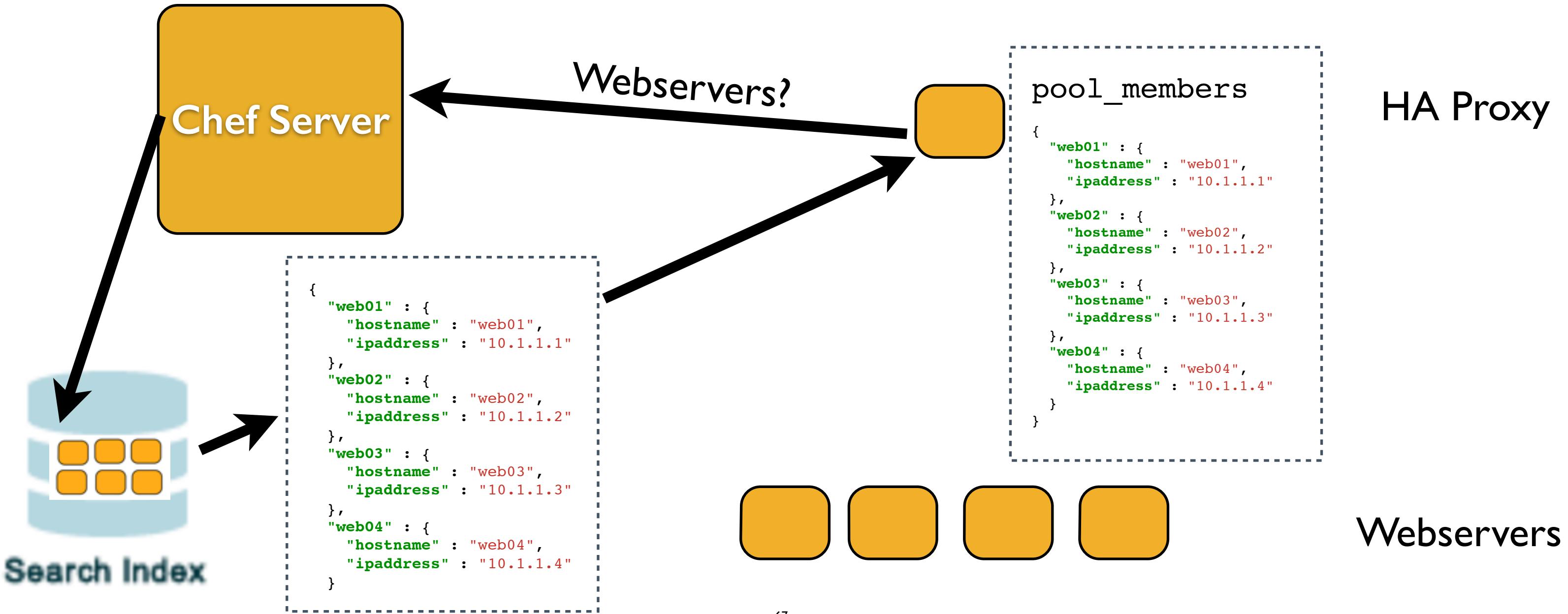
HAProxy Load Balancer

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



HAProxy Load Balancer

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



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 -%>
```

HAProxy Configuration

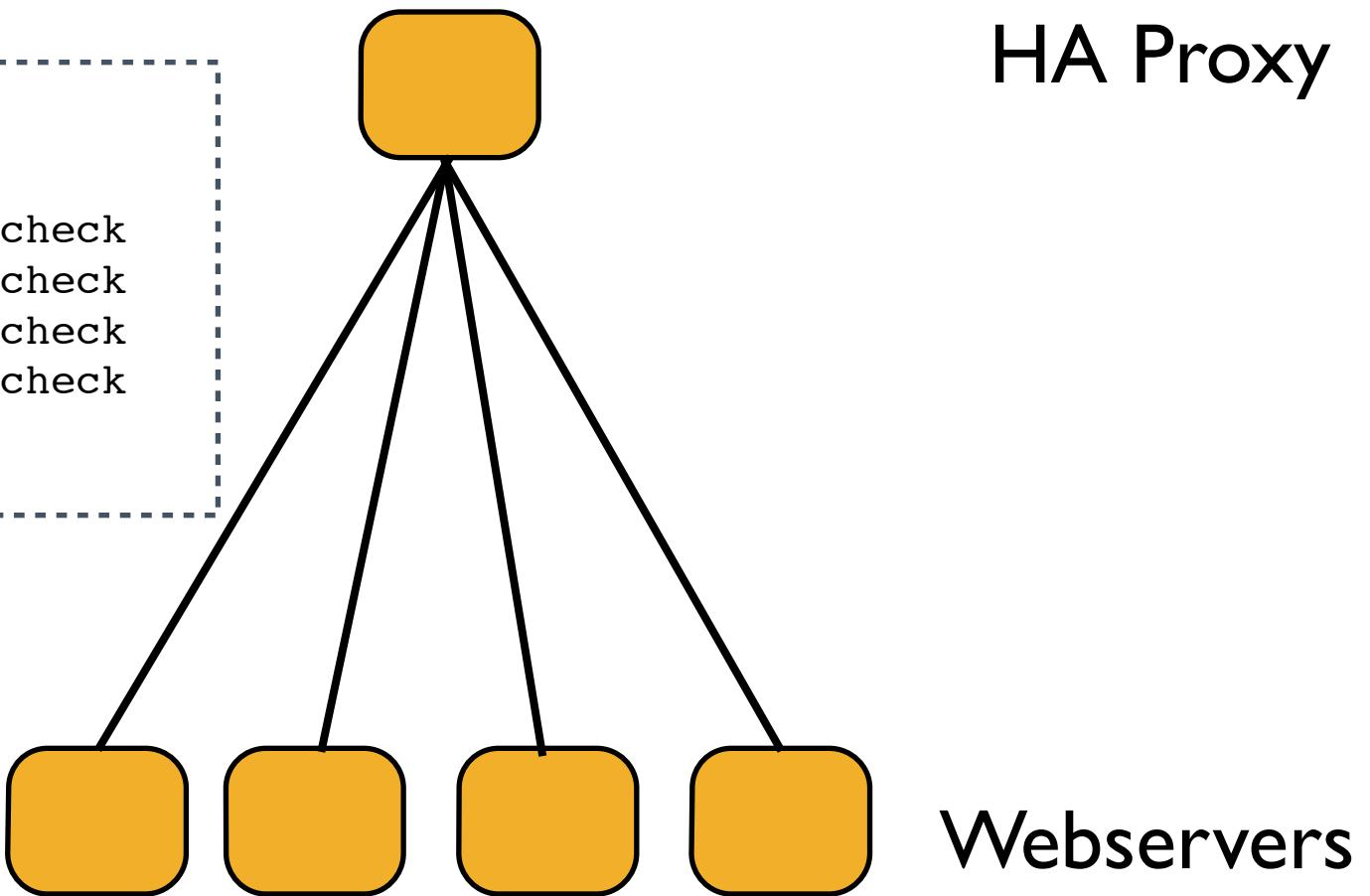
```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>:> weight 1 maxconn 1 check
<% end -%>
```

pool_members

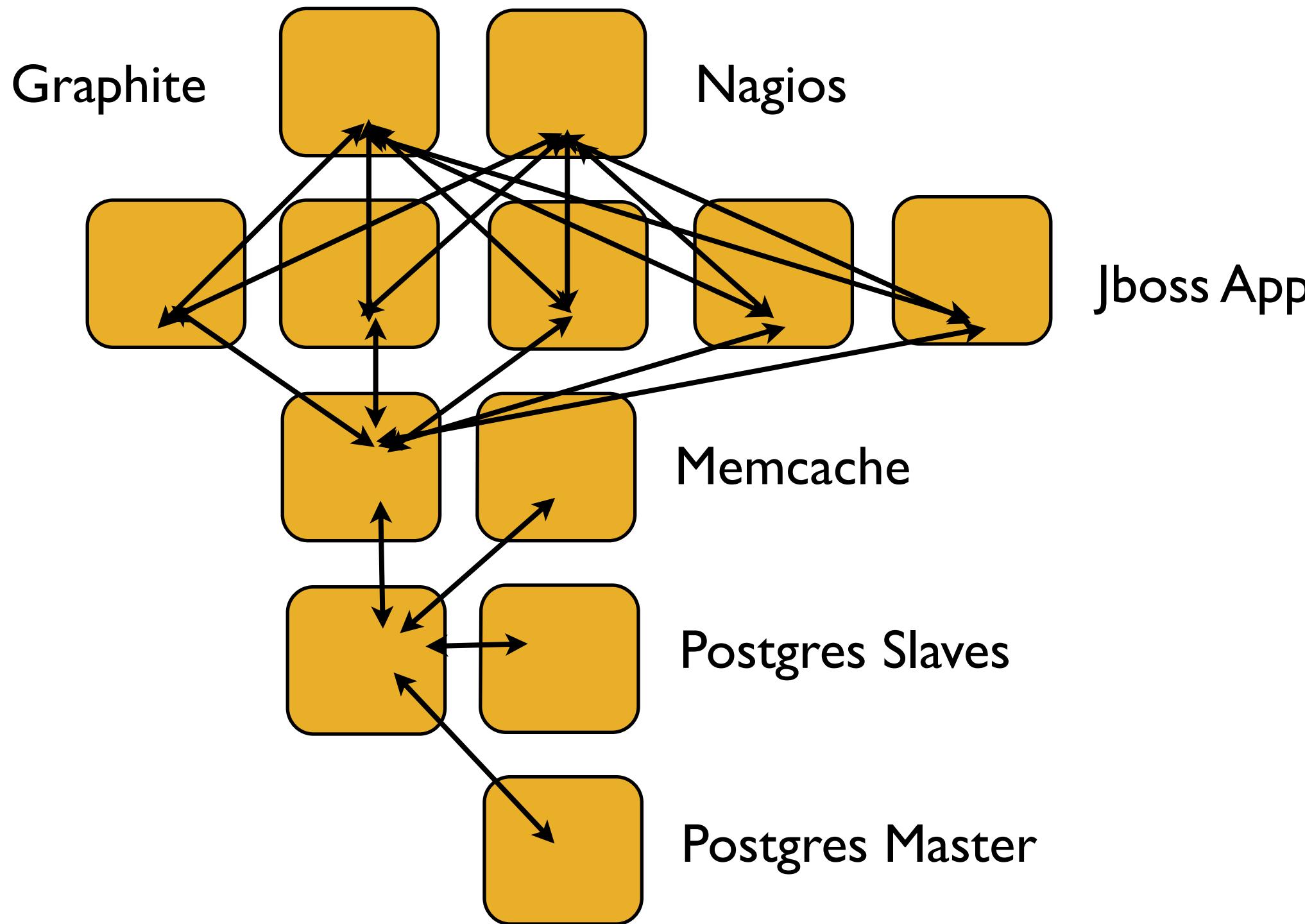
```
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  }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

haproxy.cfg

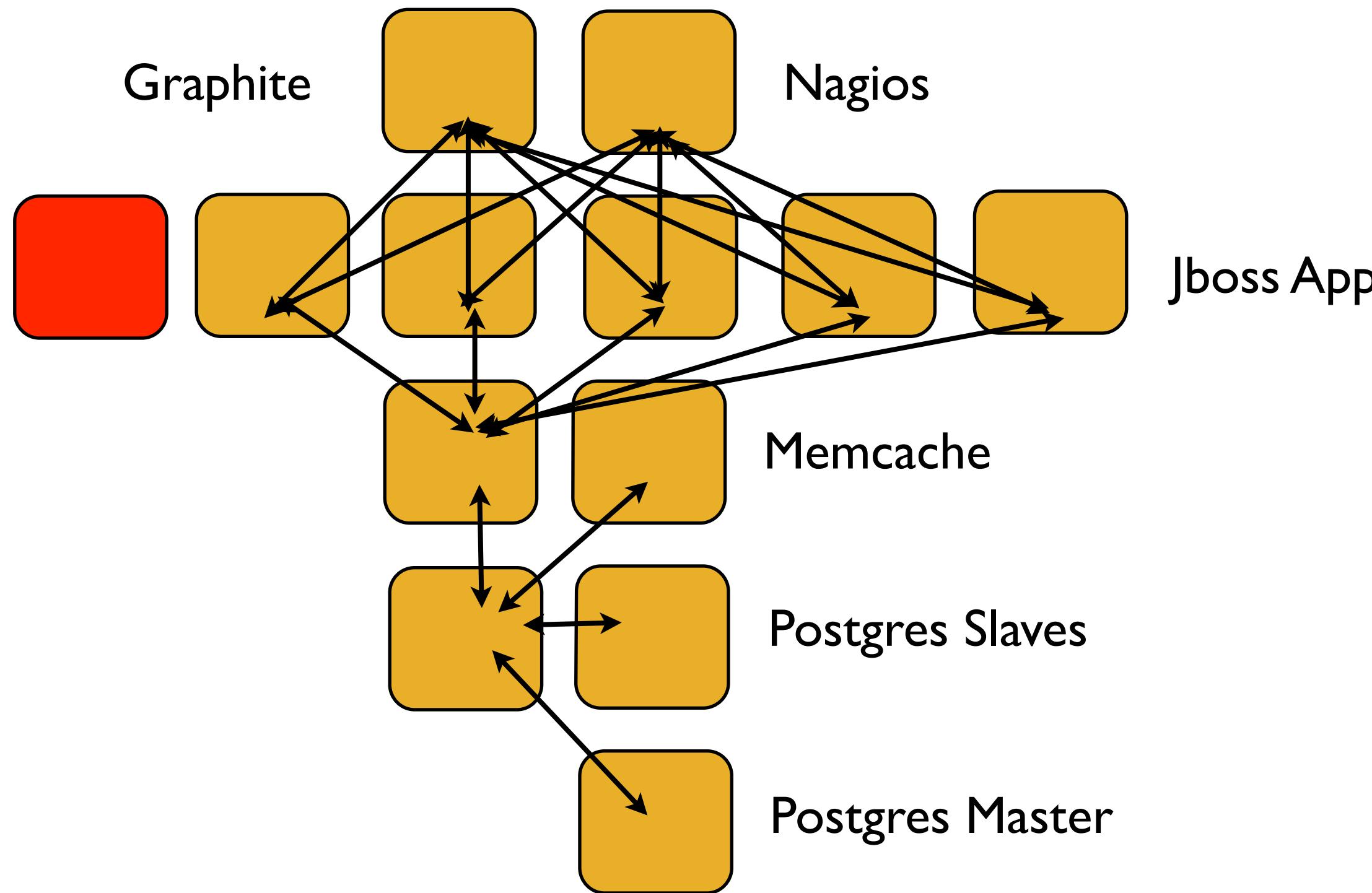
```
server web01 10.1.1.1 weight 1 maxconn 1 check
server web02 10.1.1.2 weight 1 maxconn 1 check
server web03 10.1.1.3 weight 1 maxconn 1 check
server web04 10.1.1.4 weight 1 maxconn 1 check
```



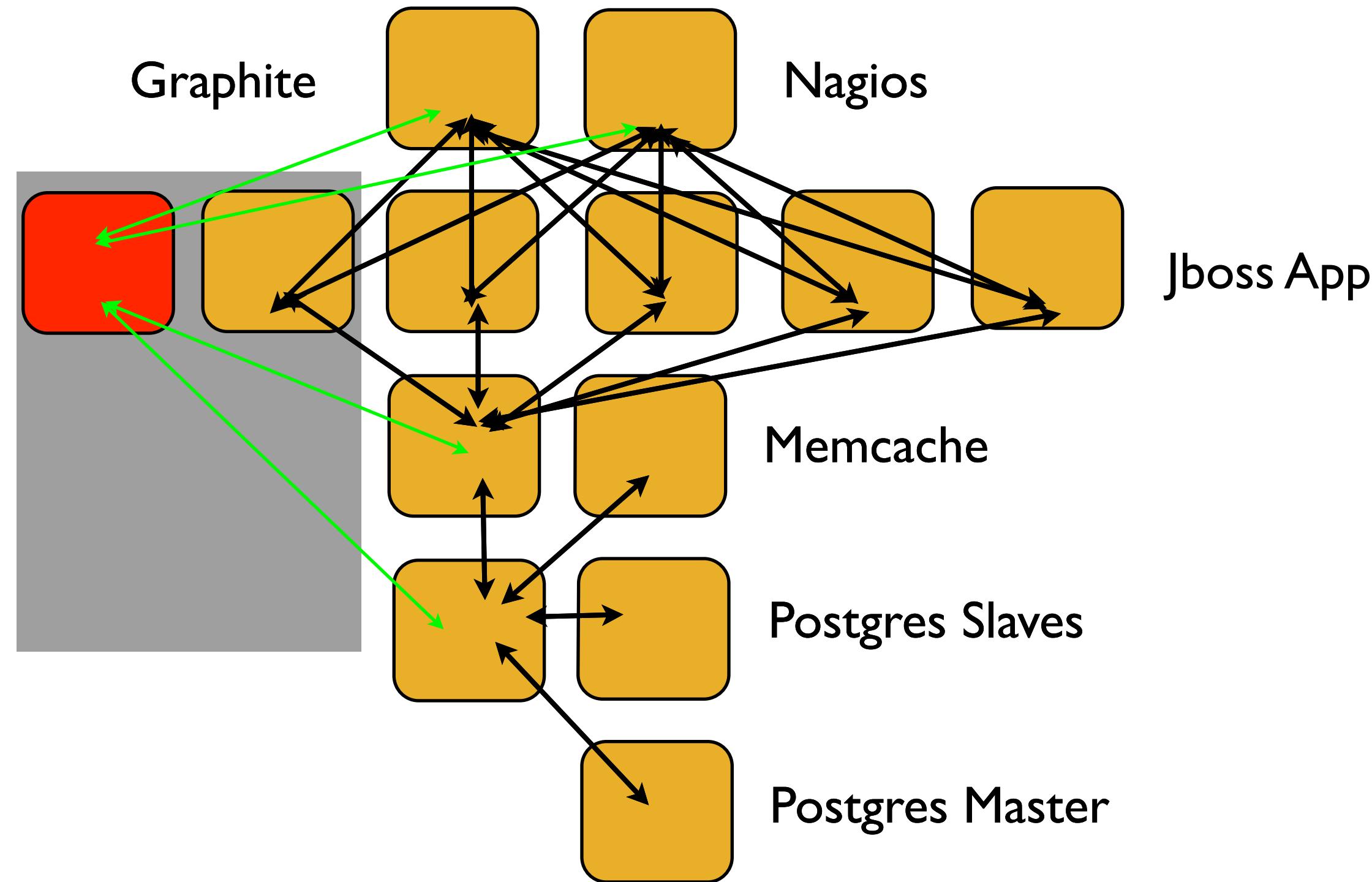
So when this...



...becomes this



...all of that happens



Managing configuration drift

- Configuration drift happens when:
 - The configuration of a server is no longer in the desired state:
 - manual out-of-band changes
 - automatic processes change system state
- Chef makes it easy to manage
 - Model the new requirements in your Chef configuration files
 - Run the chef-client to enforce your desired state

Design Tenets of Chef

- Whipuptitude - aptitude for whipping things up
- Manipulexity - manipulation of complex things
- Reasonability - the principle of least surprise
- Sane defaults - the principle of least surprise
- Flexibility - no guardrails

Summary

- What is a resource?
- What is a recipe?
- What is a cookbook?
- Why is search helpful in your configuration code?

Summary

- What questions can I help you answer?

Workstation Setup

Getting Started

AWS PopUp Loft - A Taste of Chef v2.2.0

Lesson Objectives

- After completing the lesson, you will be able to
 - Login to a pre-configured EC2 instance with Chef installed
 - Describe the basic setup needed to configure your own Chef workstation

Pre-Configured Workstation

- In the interest of time, we are going to use a pre-configured EC2 instance as our Chef workstation
- CentOS 6.6
 - us-east-1
 - ami-4d2fd426
 - m3.medium
 - Only needs SSH inbound

SSH to your remote workstation

- Are you able to log in to your instance?

http://bit.ly/instance_list

user: root

pass: chef&aws2015

ssh root@ec2-54-x-x-x.compute-1.amazonaws.com

What was done?

- Based on CentOS-6.4-x86_64-GA-EBS
 - ami-bf5021d6 (us-east-1)
- Yum update
- Enabled root user & password login
- Installed ChefDK
- Updated Chef Provisioning
- Created AWS config file
- Private SSH key for shared AWS account

Chef Development Kit

- Contains everything you need to start using Chef
 - The new "chef" tool
 - Chef-client distribution
 - Testing frameworks
 - Linting tools
- <https://downloads.chef.io/chef-dk/>

Chef Provisioning

- Repeatably create machines and infrastructure topologies using Chef code
- Plugin model lets you write bootstrappers for your favorite infrastructures, including VirtualBox, EC2, LXC, bare metal, and many more
- Today we are going to use the AWS plugin for Chef Provisioning
- <https://github.com/chef/chef-provisioning>

Checkpoint

- Can everyone log in to their workstation instance?

Summary

- What is the ChefDK?
- What is Chef Provisioning?

Summary

- What questions can I help you answer?

Resources

Fundamental Building Blocks

Lesson Objectives

- After completing the lesson, you will be able to
 - Describe the fundamental building blocks of Chef
 - Describe how to use the **package** resource
 - Use chef-apply to manage system packages

Resources

- Piece of the system and its desired state

Resources - Package

- Package that should be installed

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

Resources - Service

- Service that should be running and enabled to start on reboot

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

Resources - Cron

- Cron job that should be configured

```
cron "restart webserver" do
  hour "2"
  minute "0"
  command "service httpd restart"
end
```

Resources - User

- User that should be managed

```
user "nginx" do
  comment "nginx user"
  uid "500"
  gid "500"
  supports :manage_home => true
end
```

Resources - DSC

- A Desired State Configuration setting that should be applied

```
dsc_script "emacs" do
  code <<-EOH
    Environment 'texteditor'
  {
    Name = 'EDITOR'
    Value = 'c:\\\\emacs\\\\bin\\\\emacs.exe'
  }
EOH
end
```

Resources - Registry Key

- Registry Key that should be created

```
registry_key "HKEY_LOCAL_MACHINE\  
  \SOFTWARE\\Microsoft\\Windows\\  
  CurrentVersion\\Policies\\  
  System" do  
  values [ {  
    :name => "Enable LUA",  
    :type => :dword,  
    :data => 0  
  } ]  
end
```

Resources - Package

- Package that should be installed

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

Resources

- Piece of the system and its desired state
- <https://docs.chef.io/chef/resources.html>

Lab 1 - Install a text editor

- **Problem:** Our workstation does not have \$EDITOR installed
- **Success Criteria:** You can edit files with \$EDITOR
- \$EDITOR is your favorite command line text editor:
vim, emacs, or nano

SSH to your workstation

```
$ ssh root@54.164.75.30
```

```
The authenticity of host '54.164.75.30 (54.164.75.30)'  
can't be established.
```

```
RSA key fingerprint is c1:ec:ab:66:fb:22:4a:8f:c2:c5:9b:  
26:77:f3:dd:b3.
```

```
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '54.164.75.30' (RSA) to the list  
of known hosts.
```

```
root@54.164.75.30's password:
```

Welcome to your workstation

- ChefDK version 0.6.2 is installed
 - `chef --version`
 - Also shows you versions of most commonly used tools
 - `berks` is "berkshelf", a dependency resolution manager (<http://berkshelf.com>)
 - `kitchen` is "test kitchen", a test harness for Chef code (<http://kitchen-ci.org>)
- We are logging in as root for ease of use

Is \$EDITOR installed?

```
$ which vim
```

```
/usr/bin/which: no vim in (/opt/chefdk/bin:/root/.chefdk/gem/ruby/2.1.0/bin:/opt/chefdk/embedded/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin)
```

Is \$EDITOR installed?

```
$ which emacs
```

```
/usr/bin/which: no vim in (/opt/chefdk/bin:/root/.chefdk/gem/ruby/2.1.0/bin:/opt/chefdk/embedded/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin)
```

Is \$EDITOR installed?

```
$ which nano
```

```
/usr/bin/which: no vim in (/opt/chefdk/bin:/root/.chefdk/gem/ruby/2.1.0/bin:/opt/chefdk/embedded/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin)
```

chef-apply

- chef-apply is an executable program that allows you to work with resources
- Is included as a part of the ChefDK
- A great way to explore resources
- NOT how you will eventually use Chef in production

What does chef-apply do?

```
$ chef-apply --help
```

```
Usage: chef-apply [RECIPE_FILE] [-e RECIPE_TEXT] [-s]
      --[no-]color                                Use colored output, defaults to enabled
      -e, --execute RECIPE_TEXT                    Execute resources supplied in a string
      -l, --log_level LEVEL                      Set the log level (debug, info, warn,
                                                error, fatal)
      -s, --stdin                                    Execute resources read from STDIN
      -v, --version                                  Show chef version
      -W, --why-run                                 Enable whyrun mode
      -h, --help                                     Show this message
```

Install vim

```
$ chef-apply -e "package 'vim'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply recipe)
* yum_package[vim] action install
- install version 7.2.411-1.8.el6 of package vim-enhanced
```

Install emacs

```
$ chef-apply -e "package 'emacs'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply recipe)
* yum_package[emacs] action install
  - install version 23.1-25.el6 of package emacs
```

Install nano

```
$ chef-apply -e "package 'nano'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply recipe)
* yum_package[nano] action install
- install version 2.0.9-7.el6 of package nano
```

Resources

- Describe the desired state
- Do not need to tell Chef how to get there
- What happens if you re-run the chef-apply command?

Install vim

```
$ chef-apply -e "package 'vim'"
```

```
Recipe: (chef-apply cookbook) :: (chef-apply recipe)
* yum_package[vim] action install (up to date)
```

Test and Repair

- Resources follow a test and repair model

package "vim"

Test and Repair

- Resources follow a **test** and repair model

package "vim"

Test

Is vim installed?

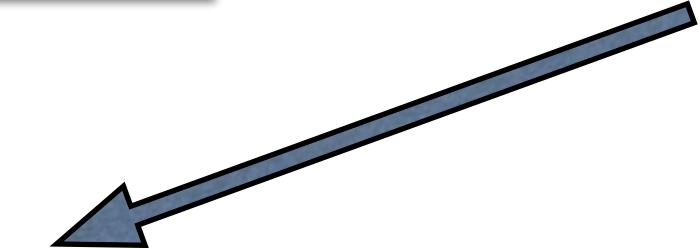
Test and Repair

- Resources follow a **test** and repair model

package "vim"

Test

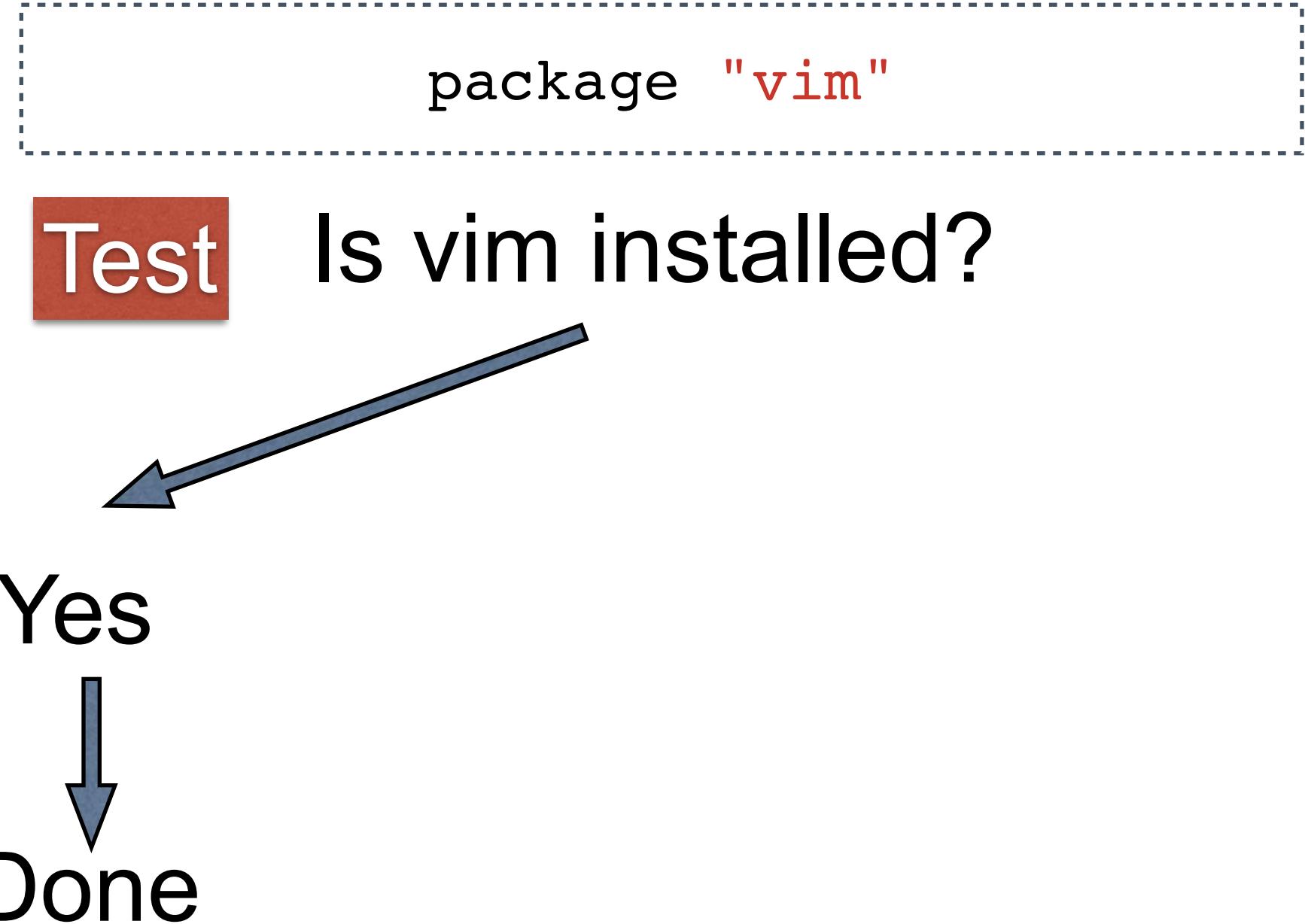
Is vim installed?



Yes

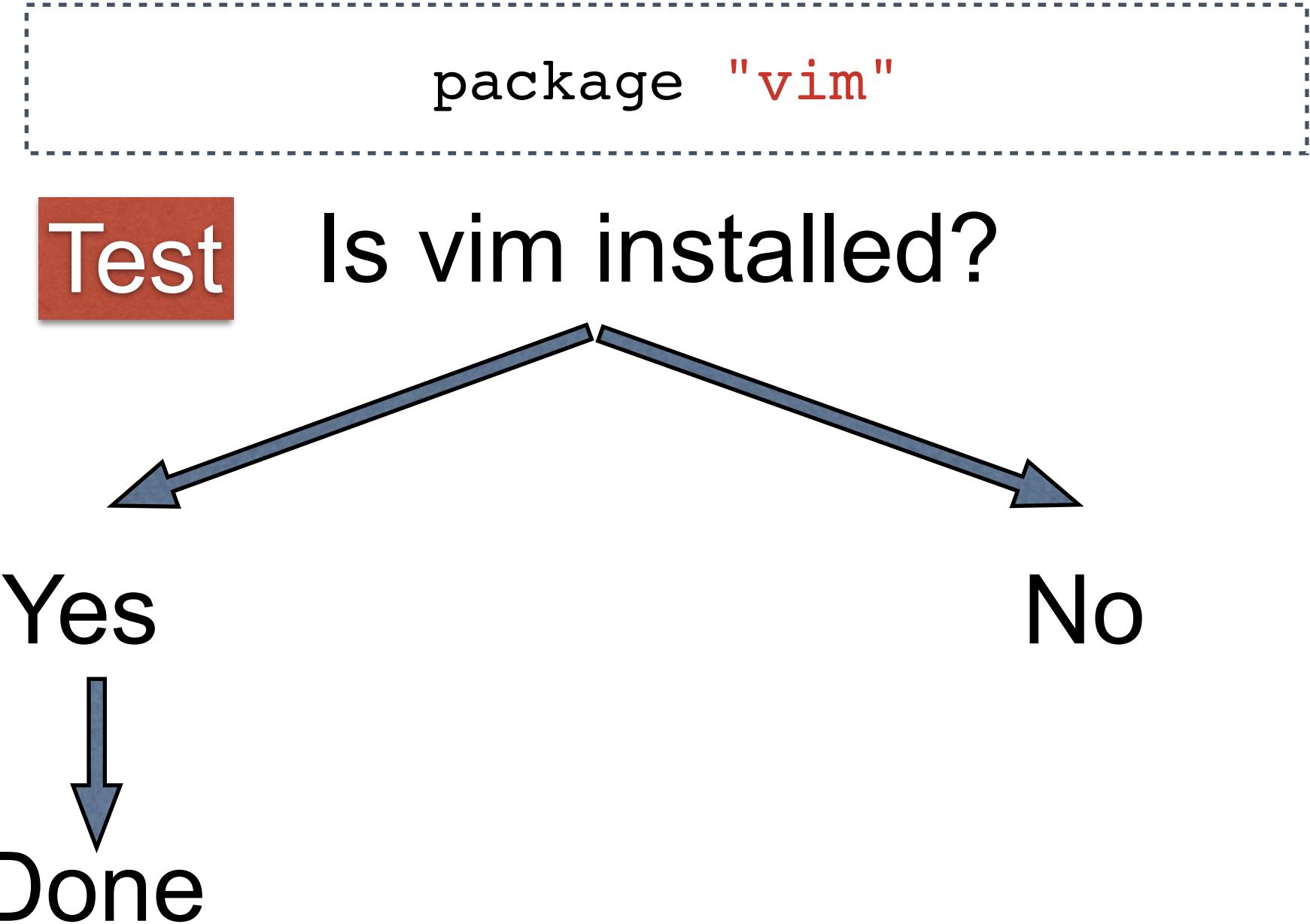
Test and Repair

- Resources follow a **test** and repair model



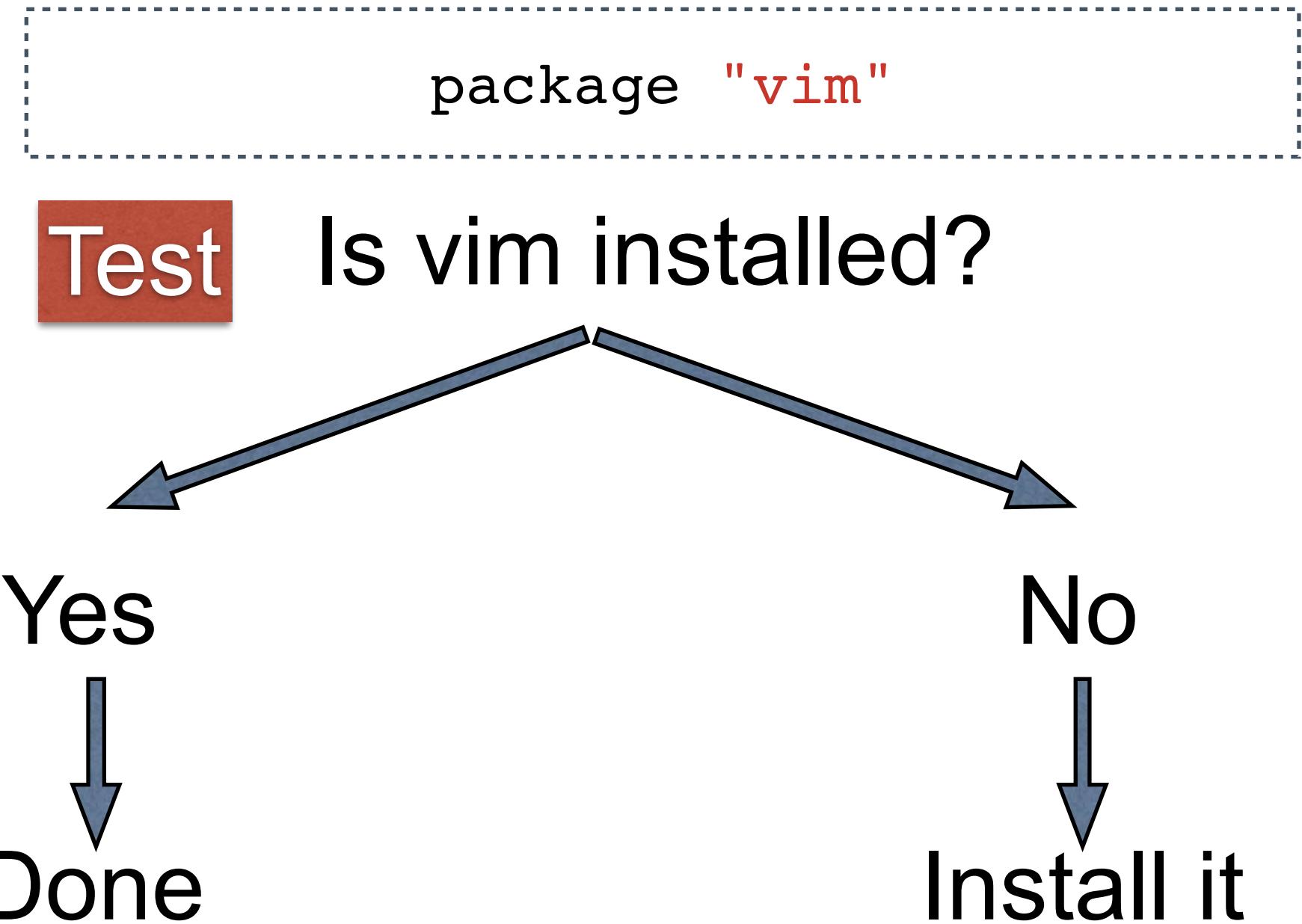
Test and Repair

- Resources follow a **test** and repair model



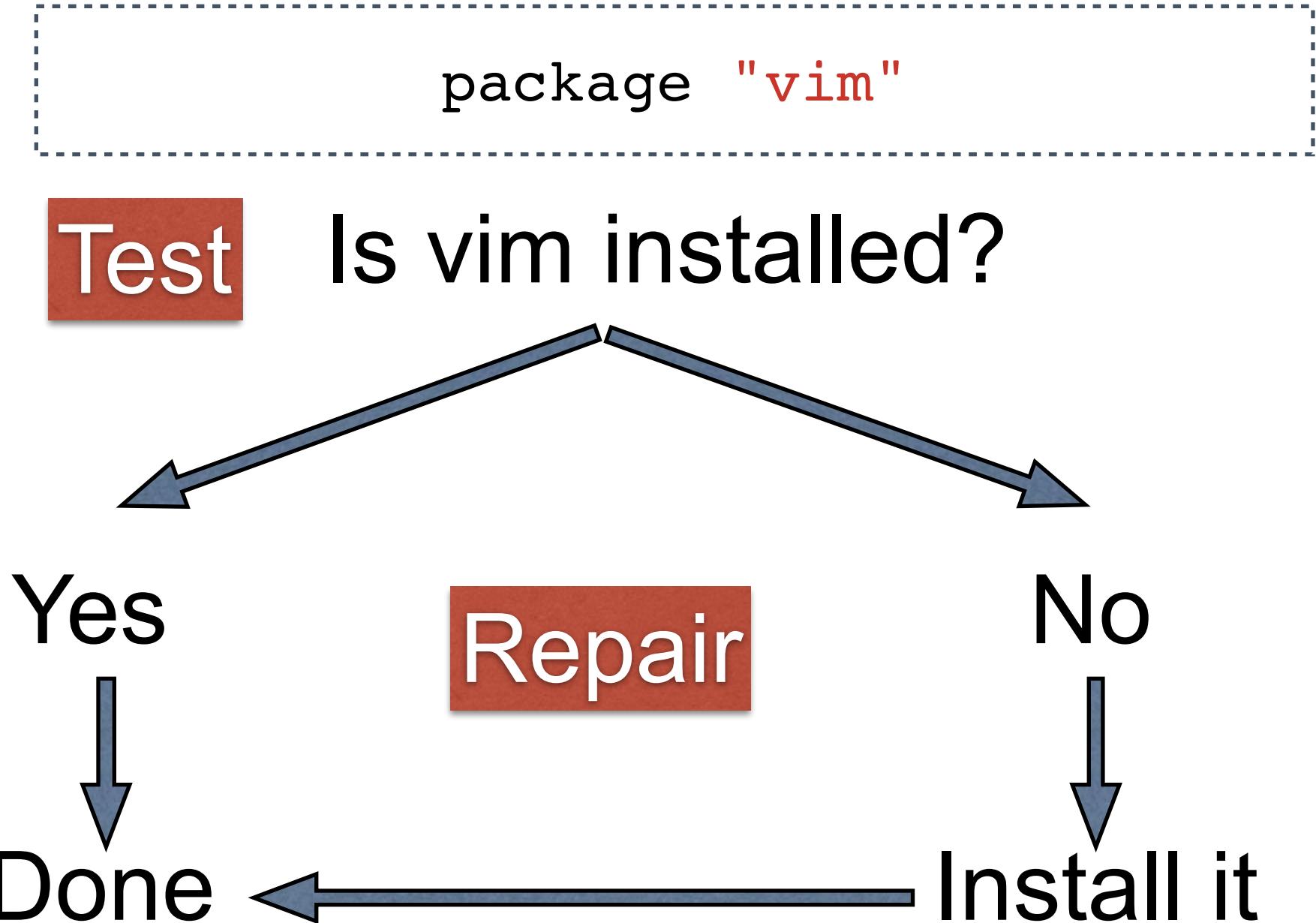
Test and Repair

- Resources follow a **test** and repair model



Test and Repair

- Resources follow a **test** and **repair** model



Resources: Test and Repair

- Resources follow a test & repair model
- Resource currently in the desired state? (test)
 - Yes - Do nothing
 - No - Bring the resource into the desired state (repair)

Resources

- package
- template
- service
- directory
- user
- group
- dsc_script
- registry_key
- powershell_script
- cron
- mount
- route
- ... and more!

Lab 2 - Hello World!

- **Problem:** Oops, we forgot to start with "hello world"
- **Success Criteria:** A file with "Hello, World!" content is available in our home directory

Hello, World!



OPEN IN EDITOR: ~/hello.rb

```
file "hello.txt" do
  action :create
  content "Hello, World!"
  mode "0644"
  owner "root"
  group "root"
end
```

SAVE FILE!

Apply hello.rb

```
$ chef-apply hello.rb
```

```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * file[hello.txt] action create
    - create new file hello.txt
    - update content in file hello.txt from none to dffd60
      --- hello.txt 2015-06-24 03:48:24.409701000 +0000
      +++ ./hello.txt 2015-06-24 03:48:24.409701000 +0000
      @@ -1 +1,2 @@
      +Hello, World!
    - change mode from '' to '0644'
    - change owner from '' to 'root'
    - change group from '' to 'root'
    - restore selinux security context
```

Read hello.txt

```
$ cat hello.txt
```

Hello, World!

Chef Resources

- Have a type

```
file "hello.txt"
```

Chef Resources

- Have a type
- Have a name

```
file "hello.txt"
```

Chef Resources

- Have a type
- Have a name
- Include details between keywords **do** and **end**

```
file "hello.txt" do  
end
```

Chef Resources

- Have a type
- Have a name
- Include details between keywords **do** and **end**
- Describe the state using keyword **action**

```
file "hello.txt" do
  action :create
end
```

Chef Resources — In Plain English

- The **TYPE** named **NAME** should be **ACTION**'ed
- The **file** named **"hello.txt"** should be created

```
file "hello.txt" do
  action :create
end
```

Chef Resources

- Have a type
- Have a name
- Include details between keywords **do** and **end**
- Describe the state using keyword **action**
- Include additional details about the state of the thing (attributes)

```
file "hello.txt" do
  action :create
  content "Hello, World!"
  mode "0644"
  owner "root"
  group "root"
end
```

Chef Resources — In Plain English

- The **TYPE** named **NAME** should be **ACTION'ed** with **ATTRIBUTES**

```
file "hello.txt" do
  action :create
  content "Hello, World!"
  mode "0644"
  owner "root"
  group "root"
end
```

Chef Resources — In Plain English

- The file named "hello.txt" should be created with content of "Hello, World!", permissions of 0644, owned by the be root user and root group

```
file "hello.txt" do
  action :create
  content "Hello, World!"
  mode "0644"
  owner "root"
  group "root"
end
```

Re-apply hello.rb

```
$ chef-apply hello.rb
```

```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
* file[hello.txt] action create (up to date)
```

Resources: Test and Repair

- Resources follow a test & repair model
- Resource currently in the desired state? (test)
 - Yes - Do nothing
 - No - Bring the resource into the desired state (repair)

What happens if...?

- Change the content of the file using your favorite text editor?
- Change the ownership of the file?
- Delete the file?

Resources

- package
- template
- service
- directory
- user
- group
- dsc_script
- registry_key
- powershell_script
- cron
- mount
- route

Resources

- What states can a file be in?
- What state will a file be in if you don't declare an action?
- What state will a package be in if you don't declare an action?
- Do you have to indent the attributes of a resource?
- What Chef tool allows us to easily explore resources?

Lab 3 - Manage a file

- The file named /etc/motd should have the contents "Property of COMPANY NAME\n", permissions of "0644", and owned by the group and user named "root"

Lab 3 - Manage a file

- The **file** named **/etc/motd** should have the contents "**Property of COMPANY NAME\n**", permissions of "**0644**", and owned by the group and user named "**root**"
- What questions can I help you answer?

Lab 3 Check-In

- Did anyone complete the last lab successfully?

Message of the Day



OPEN IN EDITOR: ~/motd.rb

```
file '/etc/motd' do
  action :create
  content "Property of Chef\n"
  mode "0644"
  owner "root"
  group "root"
end
```

SAVE FILE!

Apply hello.rb

```
$ chef-apply motd.rb
```

```
Recipe: (chef-apply cookbook)::(chef-apply recipe)
* file[/etc/motd] action create
  - update content in file /etc/motd from e3b0c4 to ab7caf
    --- /etc/motd 2010-01-12 13:28:22.000000000 +0000
    +++ /etc/.motd20150624-1998-1ozk7nw2015-06-24 03:54:19.419700998 +0000
    @@ -1 +1,2 @@
+Property of Chef
- restore selinux security context
```

Success!

- Congratulations, you just learned some Chef!

Describing Desired State

Recipes & Cookbooks

Lesson Objectives

- After completing the lesson, you will be able to
 - Use the "chef" command to generate a repository of chef code
 - Use the "chef" command to generate cookbooks
 - Describe how to use recipes
 - Use the chef-client command to apply recipes

Resources > Recipes > Cookbooks

- A resource is a piece of the system and its desired state
- A recipe is a collection of resources
- A cookbook is a collection of recipes
 - Cookbooks can also contain "ingredients"
 - A cookbook is a "package" of desired state information

Recipe - A collection of 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
```

Order Matters

- Resources are executed in order

1st



```
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
```

Order Matters

- Resources are executed in order

1st

2nd

```
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
```

Order Matters

- Resources are executed in order

1st

2nd

3rd

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

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

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

Cookbook

- A "package" for Chef code
- Typically map 1:1 to a piece of software or functionality
- Distribution unit
- Versioned
- Modular and re-usable

Abstracting Data from Desired State

- Desired State - Tomcat should be installed
- Data - It should be version 6 and listening on 8080

Abstracting Data from Desired State

- Desired State - A file should exist
- Data - The content of that file

Abstracting Data from Desired State

- Desired State - The desired state of the system
- Data - Details that may frequently change

Lab 4 - Manage Data & Desired State separately

- **Problem:** The configuration code of file `/etc/motd` currently has desired state intermingled with data (content)
- **Success Criteria:** We have configuration code that manages the desired state of `/etc/motd` separately from the data

Message of the day

- Desired State is code that describes the resource

```
file "/etc/motd" do
  content "Property of COMPANY NAME"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

Message of the day

- Data is content that may change independently of code changes

```
file "/etc/motd" do
  content "Property of COMPANY NAME"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

Cookbooks contain ingredients

- Cookbooks allow you to package up separate components in one distributable unit
- Cookbooks contain recipes (desired state)
- Cookbooks can also contain files to distribute (data)

Lab 4 - Manage Data & Desired State separately

- Required steps:
 - Create a repository to store our cookbooks
 - Create a new cookbook in that repository
 - Separate desired state from data in the recipe

The chef-repo

- Chef cookbooks should be stored in a version control system
- You have options for how best to manage your repository of chef code
- We will create a single chef-repo for all of our cookbooks

The "chef" command

- chef is an executable command line tool
 - generating cookbooks, recipes, and other things that make up your Chef code
 - ensuring RubyGems are downloaded properly for your development environment
 - verifying that all the Chef components are installed and configured correctly
- Included with ChefDK

What can "chef" generate?

```
$ chef generate --help
```

Usage: chef generate GENERATOR [options]

Available generators:

app	Generate an application repo
cookbook	Generate a single cookbook
recipe	Generate a new recipe
attribute	Generate an attributes file
template	Generate a file template
file	Generate a cookbook file
lwrp	Generate a lightweight resource/provider
repo	Generate a Chef policy repository
policyfile	Generate a Policyfile for use with the install/push commands
(experimental)	

How do we generate a repo?

```
$ chef generate repo --help
```

Usage: chef generate repo NAME [options]

-C, --copyright COPYRIGHT	Name of the copyright holder - defaults to 'The Authors'
-m, --email EMAIL	Email address of the author - defaults to 'you@example.com'
-a, --generator-arg KEY=VALUE	Use to set arbitrary attribute KEY to VALUE in the code_generator cookbook
code_generator cookbook	
-I, --license LICENSE	all_rights, apache2, mit, gplv2, gplv3 - defaults to all_rights
-p, --policy-only	Create a repository for policy only, not cookbooks
-g GENERATOR_COOKBOOK_PATH, --generator-cookbook	Use GENERATOR_COOKBOOK_PATH for the code_generator cookbook

Go home!

```
$ cd ~
```

Make a repo

```
$ chef generate repo chef-repo
```

```
Compiling Cookbooks...
Recipe: code_generator::repo
* directory[/root/chef-repo] action create
  - create new directory /root/chef-repo
  - restore selinux security context
* template[/root/chef-repo/LICENSE] action create
  - create new file /root/chef-repo/LICENSE
  - update content in file /root/chef-repo/LICENSE from none to aed48c
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/root/chef-repo/README.md] action create
  - create new file /root/chef-repo/README.md
  - update content in file /root/chef-repo/README.md from none to 767ead
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/root/chef-repo/Rakefile] action create
  - create new file /root/chef-repo/Rakefile
  - update content in file /root/chef-repo/Rakefile from none to 108932
    (diff output suppressed by config)
  - restore selinux security context
```

Go into the repo

```
$ ls -l ~/chef-repo
```

```
total 28
-rw-r--r-- 1 root root 974 May 10 00:22 cheffignore
drwxr-xr-x 3 root root 4096 May 10 00:22 cookbooks
drwxr-xr-x 3 root root 4096 May 10 00:22 data_bags
drwxr-xr-x 2 root root 4096 May 10 00:22 environments
-rw-r--r-- 1 root root 70 May 10 00:22 LICENSE
-rw-r--r-- 1 root root 3506 May 10 00:22 README.md
drwxr-xr-x 2 root root 4096 May 10 00:22 roles
```

Lab 4 - Manage Data & Desired State Separately

- Required steps:
 - ✓ Create a repository to store our cookbooks
 - Create a new cookbook in that repository
 - Separate desired state from data in the recipe

How do we generate a cookbook?

```
$ chef generate cookbook --help
```

```
Usage: chef generate cookbook NAME [options]
      -C, --copyright COPYRIGHT           Name of the copyright holder - defaults to 'The Authors'
      -m, --email EMAIL                  Email address of the author - defaults to 'you@example.com'
      -a, --generator-arg KEY=VALUE     Use to set arbitrary attribute KEY to VALUE in the
code_generator cookbook
      -I, --license LICENSE             all_rights, apache2, mit, gplv2, gplv3 - defaults to all_rights
      -g GENERATOR_COOKBOOK_PATH,       Use GENERATOR_COOKBOOK_PATH for the code_generator cookbook
      --generator-cookbook
```

Generate a "motd" cookbook

```
$ cd chef-repo
$ chef generate cookbook cookbooks/motd
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
* directory[/root/chef-repo/cookbooks/motd] action create
  - create new directory /root/chef-repo/cookbooks/motd
  - restore selinux security context
* template[/root/chef-repo/cookbooks/motd/metadata.rb] action create_if_missing
  - create new file /root/chef-repo/cookbooks/motd/metadata.rb
  - update content in file /root/chef-repo/cookbooks/motd/metadata.rb from none to c24a89
    (diff output suppressed by config)
  - restore selinux security context
* template[/root/chef-repo/cookbooks/motd/README.md] action create_if_missing
  - create new file /root/chef-repo/cookbooks/motd/README.md
  - update content in file /root/chef-repo/cookbooks/motd/README.md from none to 918996
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/root/chef-repo/cookbooks/motd/chefignore] action create
  - create new file /root/chef-repo/cookbooks/motd/chefignore
  - update content in file /root/chef-repo/cookbooks/motd/chefignore from none to 9727b1
    (diff output suppressed by config)
  - restore selinux security context
* cookbook_file[/root/chef-repo/cookbooks/motd/Berksfile] action create_if_missing
  - create new file /root/chef-repo/cookbooks/motd/Berksfile
  - update content in file /root/chef-repo/cookbooks/motd/Berksfile from none to 5ec92e
    (diff output suppressed by config)
  - restore selinux security context
```

Lab 4 - Manage Data & Desired State Separately

- Required steps:
 - ✓ Create a repository to store our cookbooks
 - ✓ Create a new cookbook in that repository
 - Separate desired state from data in the recipe

Copy your motd.rb

```
$ cat ~/motd.rb >> ~/chef-repo/cookbooks/motd/recipes/default.rb
```

Update the recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/motd/recipes/default.rb`

```
#  
# Cookbook Name:: motd  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
file "/etc/motd" do  
  content "Property of Chef\n"  
  action :create  
  mode "0644"  
  owner "root"  
  group "root"  
end
```

Which resource should we use?

Resources: [About Resources](#) | [Common Functionality](#) — **Resources:** [apt_package](#) | [bash](#) | [batch](#) | [breakpoint](#) | [chef_gem](#) | [chef_handler](#) | [cookbook_file](#) | [cron](#) | [deploy](#) | [directory](#) | [dpkg_package](#) | [dsc_script](#) | [easy_install_package](#) | [env](#) | [erl_call](#) | [execute](#) | [file](#) | [gem_package](#) | [git](#) | [group](#) | [http_request](#) | [ifconfig](#) | [link](#) | [log](#) | [mdadm](#) | [mount](#) | [ohai](#) | [package](#) | [powershell_script](#) | [registry_key](#) | [remote_directory](#) | [remote_file](#) | [route](#) | [rpm_package](#) | [ruby_block](#) | [script](#) | [service](#) | [subversion](#) | [template](#) | [user](#) | [yum_package](#) | [windows_package](#) — **Single Page:** [Resources and Providers](#)

- **cookbook_file**
- **file**
- **remote_file**
- **template**

cookbook_file

- A file stored in the cookbook contains the content of the file

```
motd/
├── Berksfile
├── chefignore
└── files
    └── default
        └── motd
├── metadata.rb
└── README.md
└── recipes
    └── default.rb
```

file

- The content is described inline in the recipe

```
file "/etc/motd" do
  content "Property of COMPANY NAME"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

remote_file

- The file is stored in a remote location, such as on a webserver

```
remote_file "/etc/motd" do
  source "http://bit.ly/motd"
  action :create
  mode "0644"
  owner "root"
  group "root"
end
```

template

- A template in the cookbook is used to render the content of the file

```
motd
├── Berksfile
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
└── templates
    └── default
        └── motd.erb
```

template

- A template is a mix of static and dynamic content

motd/templates/default/motd.erb

Property of <%= @company_name %>

Which resource should we use?

Resources: [About Resources](#) | [Common Functionality](#) — **Resources:** [apt_package](#) | [bash](#) | [batch](#) | [breakpoint](#) | [chef_gem](#) | [chef_handler](#) | [cookbook_file](#) | [cron](#) | [deploy](#) | [directory](#) | [dpkg_package](#) | [dsc_script](#) | [easy_install_package](#) | [env](#) | [erl_call](#) | [execute](#) | [file](#) | [gem_package](#) | [git](#) | [group](#) | [http_request](#) | [ifconfig](#) | [link](#) | [log](#) | [mdadm](#) | [mount](#) | [ohai](#) | [package](#) | [powershell_script](#) | [registry_key](#) | [remote_directory](#) | [remote_file](#) | [route](#) | [rpm_package](#) | [ruby_block](#) | [script](#) | [service](#) | [subversion](#) | [template](#) | [user](#) | [yum_package](#) | [windows_package](#) — **Single Page:** [Resources and Providers](#)

- **cookbook_file** - static file within the cookbook
- **file** - content managed inline
- **remote_file** - static file obtained from URL
- **template** - dynamic content rendered from ERB

template

- An ERB template stored as part of our cookbook

Update the recipe



OPEN IN EDITOR: ~/chef-repo/cookbooks/motd/recipes/default.rb

```
#  
# Cookbook Name:: motd  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
template "/etc/motd" do  
  action :create  
  source "motd.erb"  
  mode "0644"  
  owner "root"  
  group "root"  
end
```

How do we generate a template?

```
$ chef generate template --help
```

```
Usage: chef generate template [path/to/cookbook] NAME [options]
      -C, --copyright COPYRIGHT           Name of the copyright holder - defaults to 'The Authors'
      -m, --email EMAIL                  Email address of the author - defaults to 'you@example.com'
      -a, --generator-arg KEY=VALUE     Use to set arbitrary attribute KEY to VALUE in the
code_generator cookbook
      -I, --license LICENSE             all_rights, apache2, mit, gplv2, gplv3 - defaults to all_rights
      -s, --source SOURCE_FILE          Copy content from SOURCE_FILE
      -g GENERATOR_COOKBOOK_PATH,      Use GENERATOR_COOKBOOK_PATH for the code_generator cookbook
      --generator-cookbook
```

Create the ERB template

```
$ chef generate template cookbooks/motd motd -s /etc/motd
```

```
Compiling Cookbooks...
Recipe: code_generator::template
 * directory[cookbooks/motd/templates/default] action create
   - create new directory cookbooks/motd/templates/default
 * file[cookbooks/motd/templates/default/motd.erb] action create
   - create new file cookbooks/motd/templates/default/motd.erb
   - update content in file cookbooks/motd/templates/default/motd.erb from none to 166ed9
     (diff output suppressed by config)
```

Check the template



OPEN IN EDITOR: `~/chef-repo/cookbooks/motd/templates/default/motd.erb`

Property of Chef

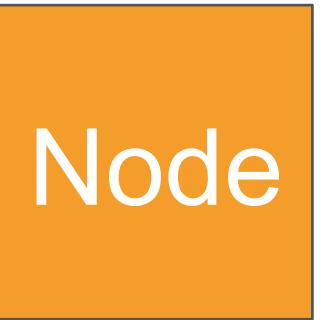
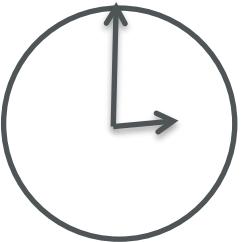
chef-apply

- chef-apply does not understand cookbooks, only resources and recipes
- We cannot use chef-apply to apply the desired state stored in our motd cookbook

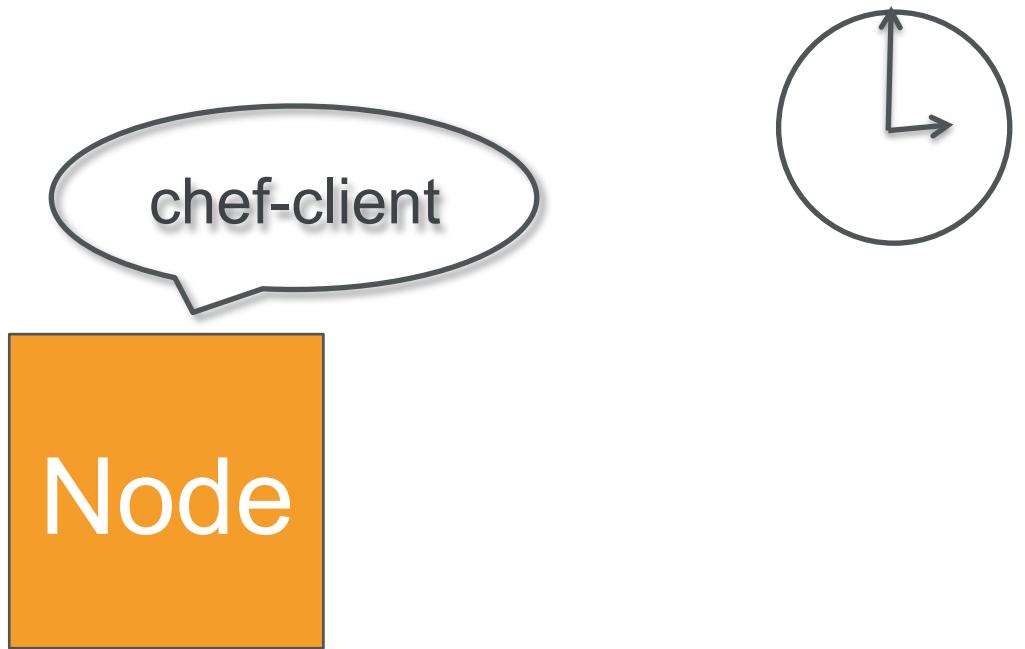
chef-client

- chef-client is an executable
 - performs all actions required to bring a node into the desired state
 - typically runs on a regular basis
 - daemon
 - cron
 - Windows service
 - Included with ChefDK

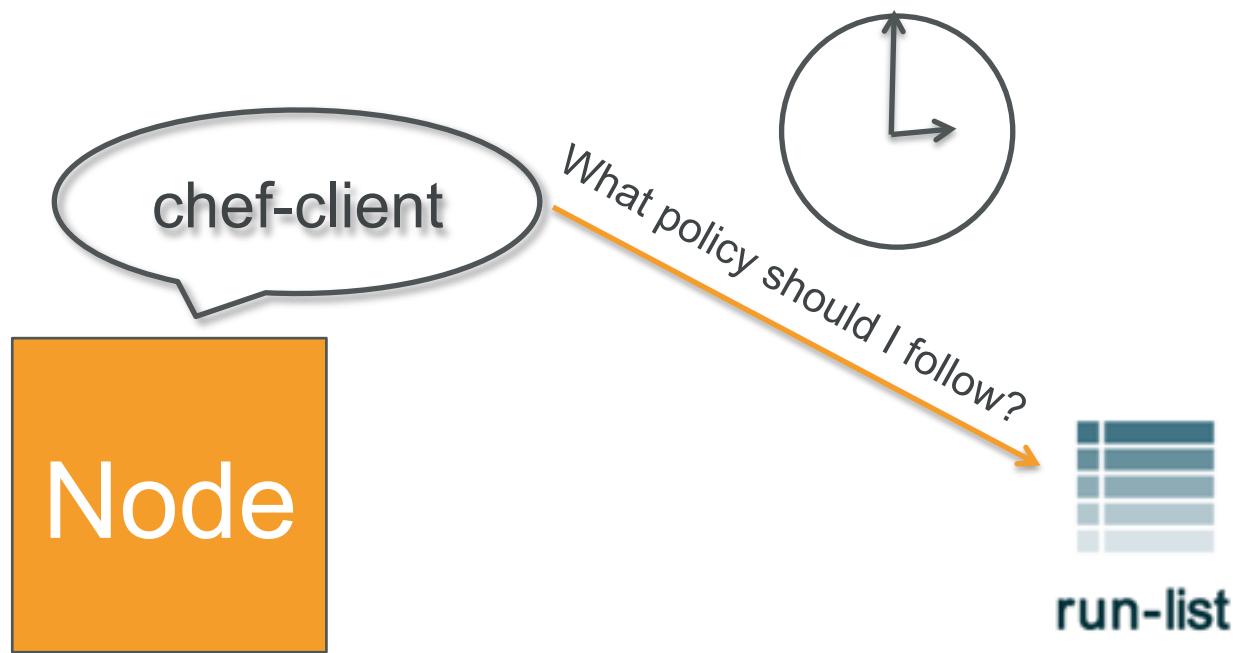
chef-client applying desired state



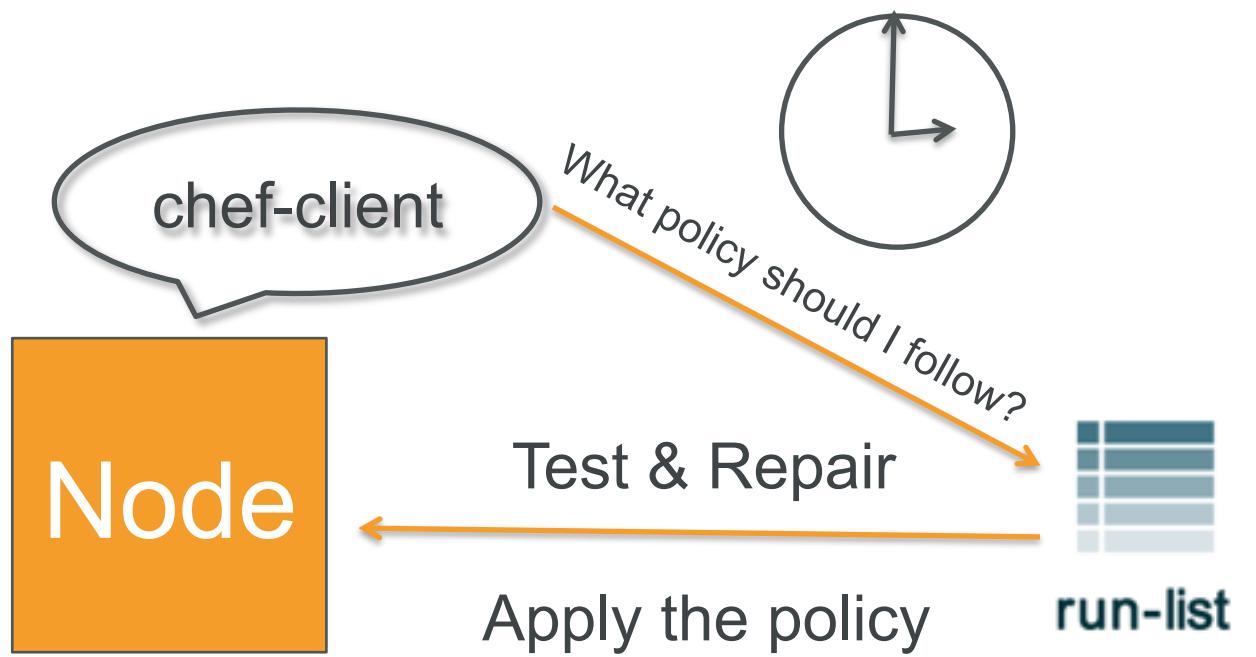
chef-client applying desired state



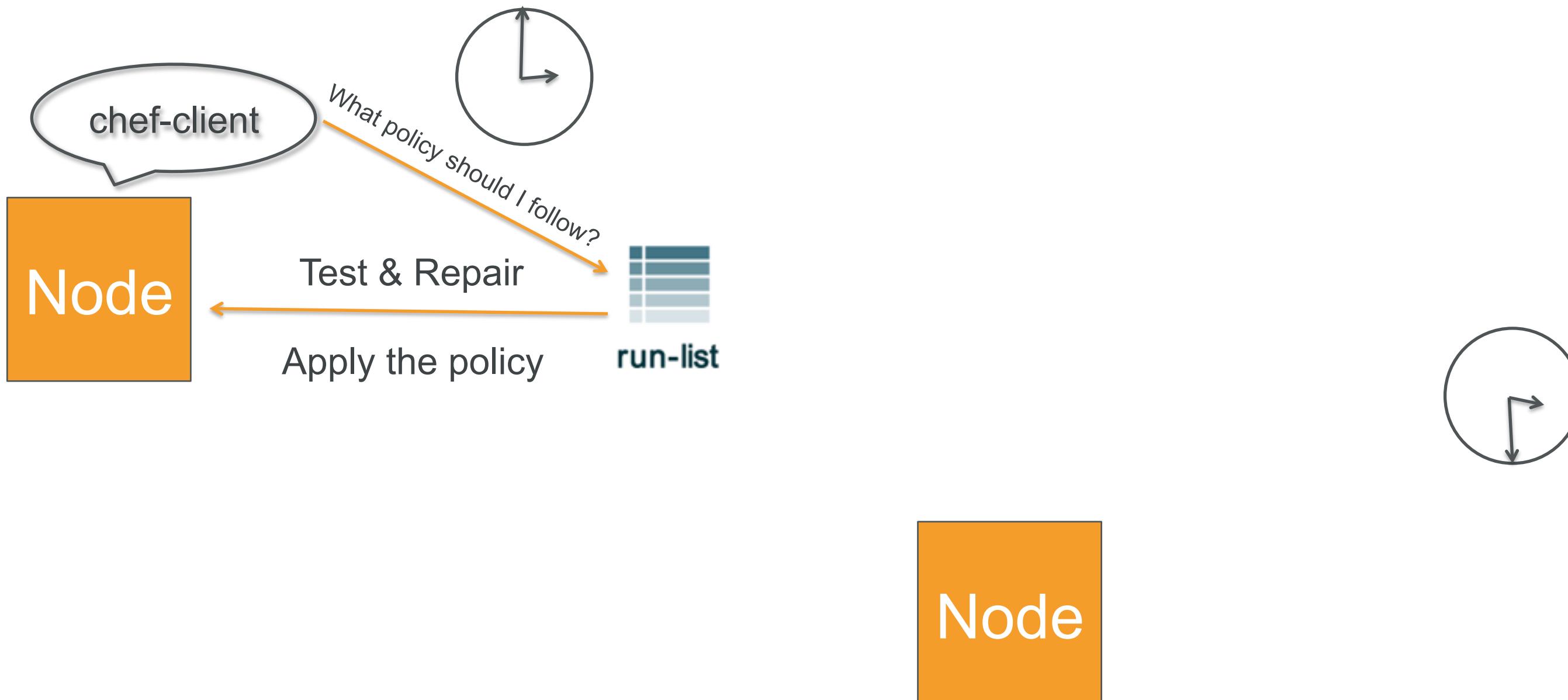
chef-client applying desired state



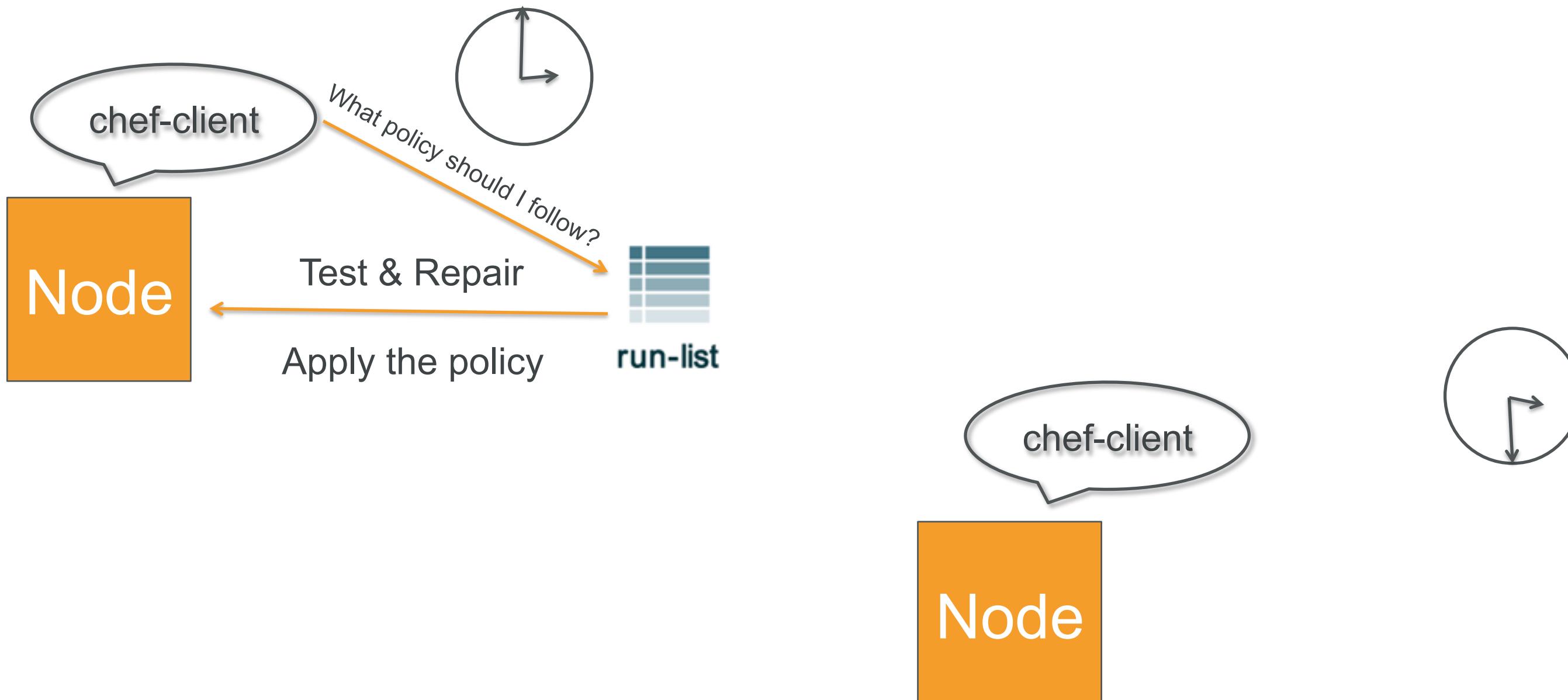
chef-client applying desired state



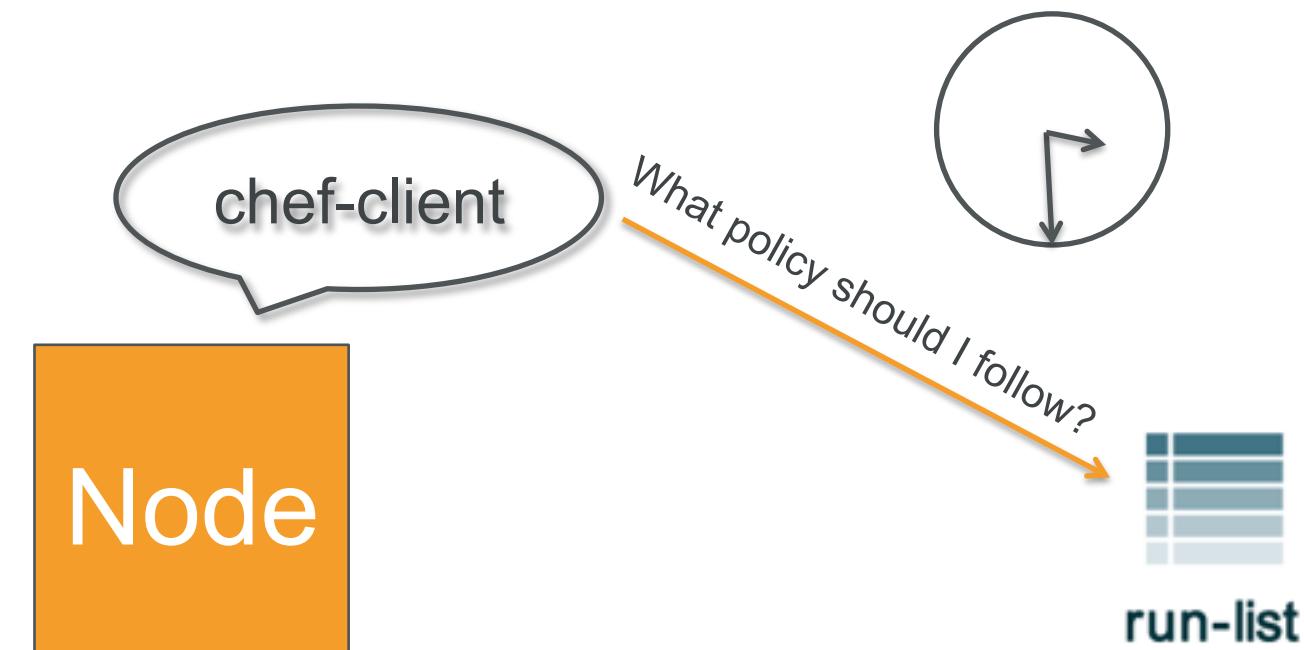
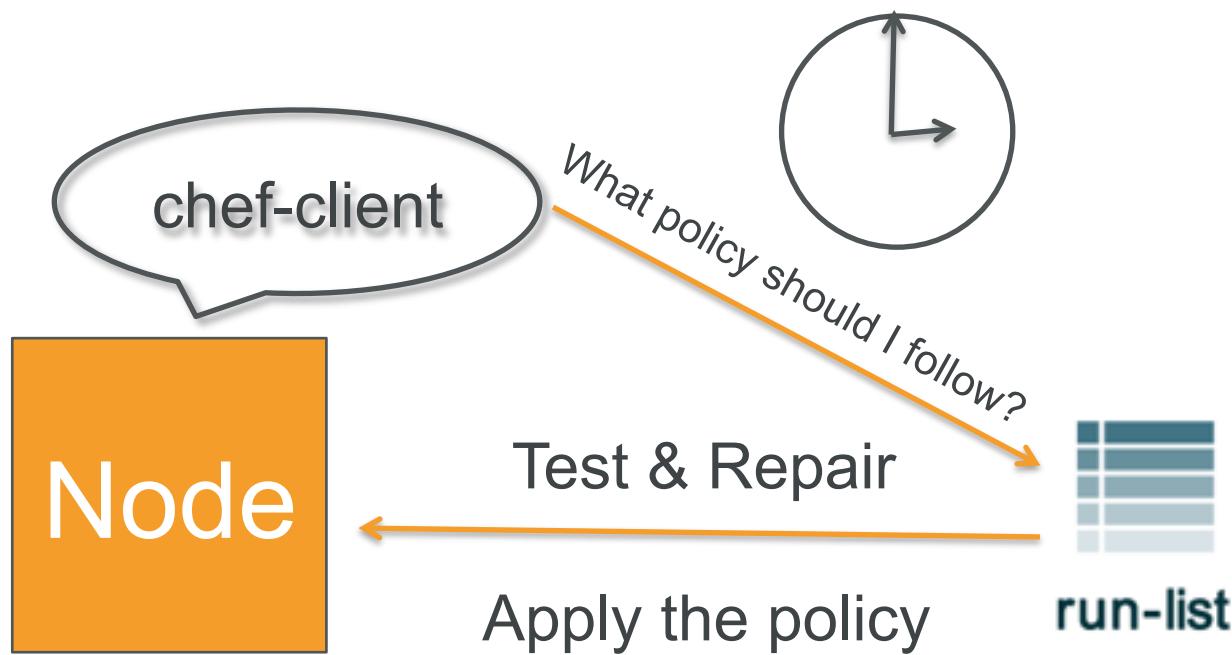
chef-client applying desired state



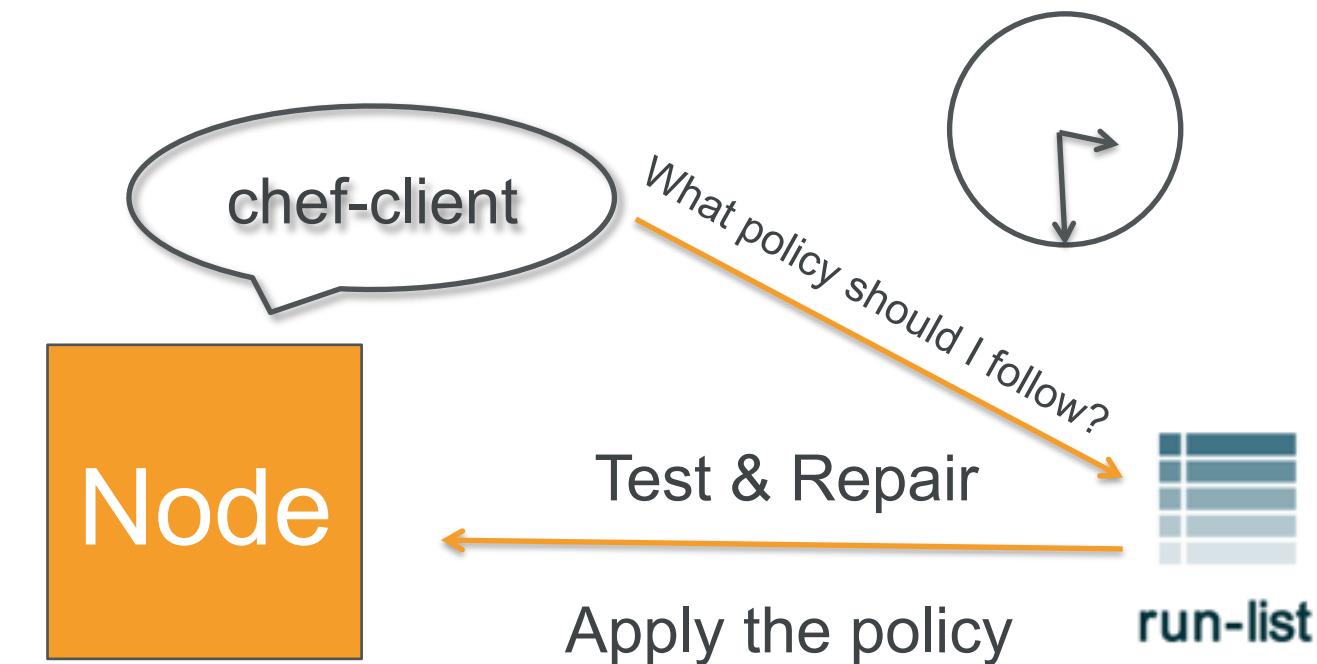
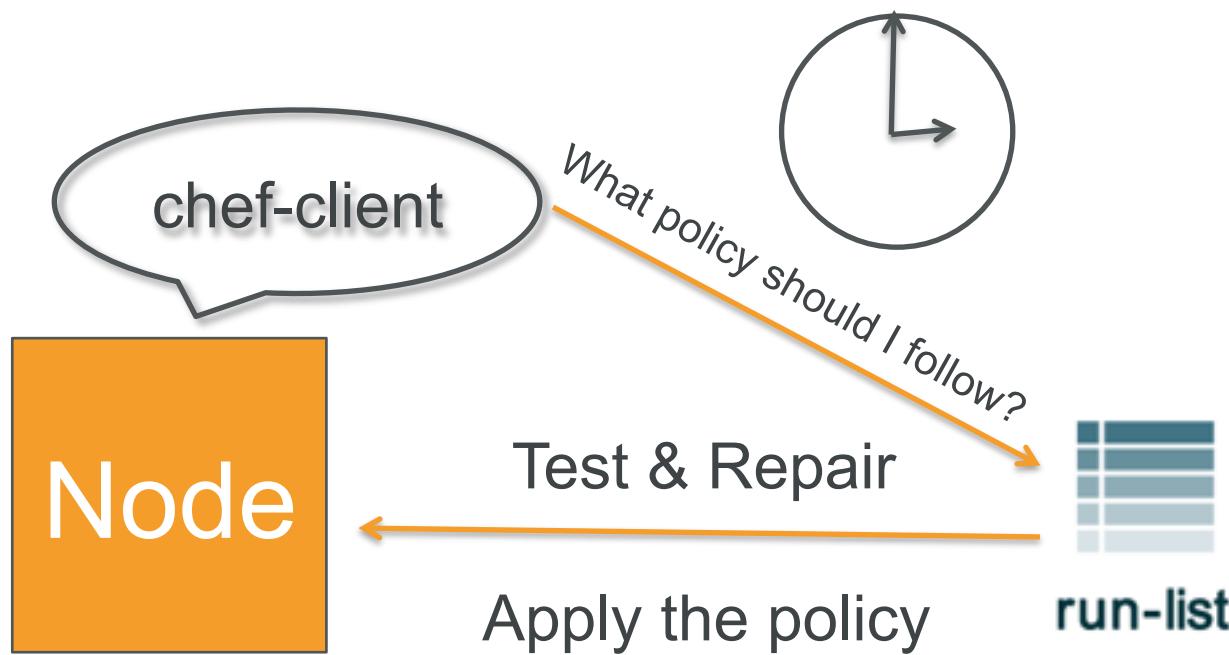
chef-client applying desired state



chef-client applying desired state



chef-client applying desired state



chef-client modes

- In conjunction with a Chef Server
- Local mode (no Chef Server)

chef-client privileges

- Usually run with elevated privileges
 - root
 - sudo
 - Administrator
- Can run as non-privileged user
 - Probably not what you want to do

Apply our recipe using chef-client

```
$ cd ~/chef-repo  
$ chef-client --local-mode -r "recipe[motd]"
```

```
Starting Chef Client, version 12.3.0  
resolving cookbooks for run list: [ "motd" ]  
Synchronizing Cookbooks:  
  - motd  
Compiling Cookbooks...  
Converging 1 resources  
Recipe: motd::default  
  * template[/etc/motd] action create (up to date)  
  
Running handlers:  
Running handlers complete  
Chef Client finished, 0/1 resources updated in 3.76464321 seconds
```

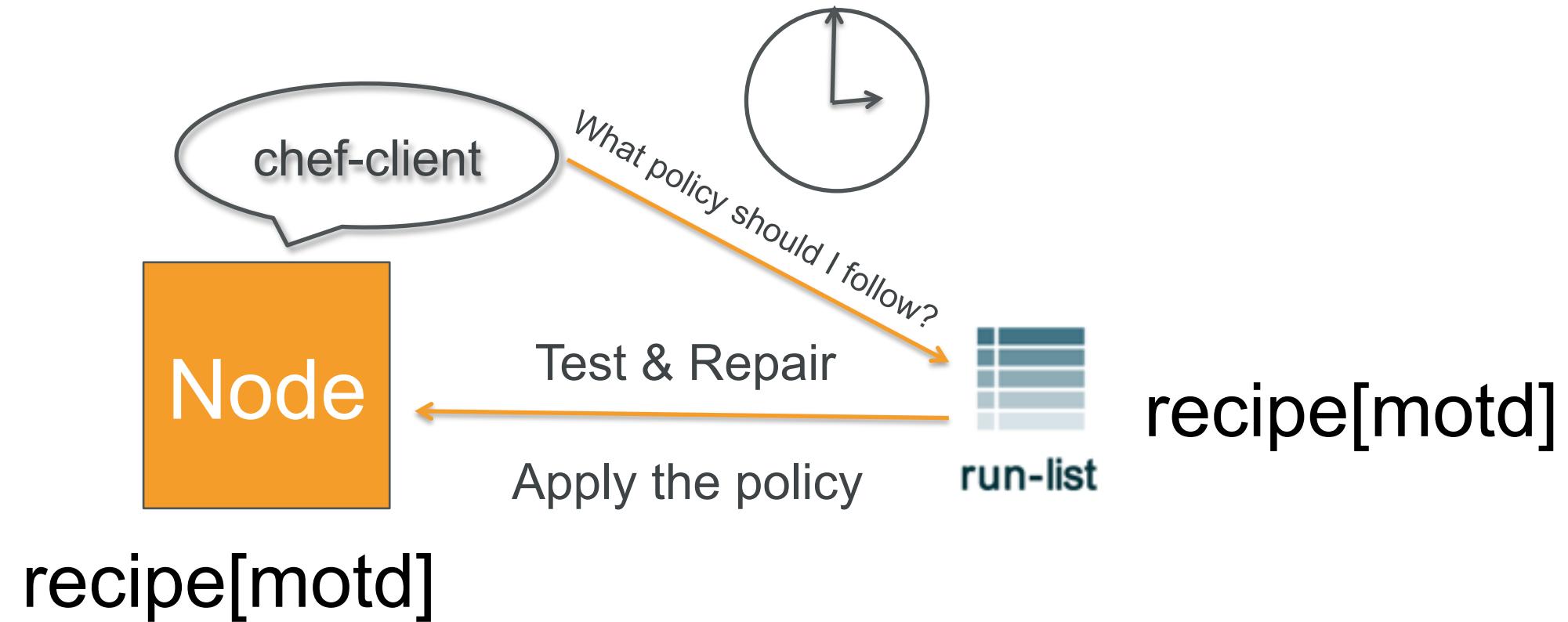
Pop Quiz

- Why did chef-client update zero resources?

Pop Quiz

- Why did chef-client update zero resources?
- Resources test & repair
- Remember: we generated the template from an existing file on the filesystem
- The content of /etc/motd is already the content we wanted

chef-client applying desired state



Lab 4 - Manage Data & Desired State Separately

- Required steps:
 - ✓ Create a repository to store our cookbooks
 - ✓ Create a new cookbook in that repository
 - ✓ Separate desired state from data in the recipe

Separating data from desired state

- Storing file content directly in a recipe is not typically the best choice
- We can manage that content separately using a different resource
 - `cookbook_file`
 - `remote_file`
 - `template`

Template resources

- An ERB template that generates files based on a mix of static content along with variables and logic to determine dynamic content
- If in doubt, start with a template

What would we do if...?

- The contents of the motd file should be pulled from a file in an S3 bucket?
- The motd file should have variable content based on different owners (i.e. different company names) using the same cookbook?

Summary

- What components can be in a cookbook?
- What program should we use to apply code from cookbooks?
- What resource should we look to first when managing a file?

Summary

- What questions can I help you answer?

Creating a simple web server

An Example Desired State

Lesson Objectives

- After completing the lesson, you will be able to
 - Use the three most common resources to accomplish configuration tasks
 - Describe the node object and how Chef sees your infrastructure
 - Display dynamic node data in a template

Lab 5 - Install an Apache webserver

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

Required steps

- Install Apache
- Write out the home page
- Start the Apache service
- Make sure the Apache service starts when the machine boots
- Stop and disable the iptables service
- *Please note we're teaching Chef primitives, not web server management! This is probably not the Apache HTTP server configuration you would use in production.*

Create a new webserver cookbook

```
$ chef generate cookbook cookbooks/webserver
```

```
Compiling Cookbooks...
Recipe: code_generator::cookbook
 * directory[/root/chef-repo/webserver] action create
   - create new directory /root/chef-repo/webserver
   - restore selinux security context
 * template[/root/chef-repo/webserver/metadata.rb] action create_if_missing
   - create new file /root/chef-repo/webserver/metadata.rb
   - update content in file /root/chef-repo/webserver/metadata.rb from none to 4c0e2d
     (diff output suppressed by config)
   - restore selinux security context
 * template[/root/chef-repo/webserver/README.md] action create_if_missing
   - create new file /root/chef-repo/webserver/README.md
   - update content in file /root/chef-repo/webserver/README.md from none to 5c3d3a
     (diff output suppressed by config)
   - restore selinux security context
 * cookbook_file[/root/chef-repo/webserver/chefignore] action create
   - create new file /root/chef-repo/webserver/chefignore
   - update content in file /root/chef-repo/webserver/chefignore from none to 9727b1
     (diff output suppressed by config)
   - restore selinux security context
```

Exercise: Edit the default recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/default.rb`

```
#  
# Cookbook Name:: webserver  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.
```

Which resource should we use?

Resources: [About Resources](#) | [Common Functionality](#) — **Resources:** [apt_package](#) | [bash](#) | [batch](#) | [breakpoint](#) | [chef_gem](#) | [chef_handler](#) | [cookbook_file](#) | [cron](#) | [deploy](#) | [directory](#) | [dpkg_package](#) | [dsc_script](#) | [easy_install_package](#) | [env](#) | [erl_call](#) | [execute](#) | [file](#) | [gem_package](#) | [git](#) | [group](#) | [http_request](#) | [ifconfig](#) | [link](#) | [log](#) | [mdadm](#) | [mount](#) | [ohai](#) | [package](#) | [powershell_script](#) | [registry_key](#) | [remote_directory](#) | [remote_file](#) | [route](#) | [rpm_package](#) | [ruby_block](#) | [script](#) | [service](#) | [subversion](#) | [template](#) | [user](#) | [yum_package](#) | [windows_package](#) — **Single Page:** [Resources and Providers](#)

- Install Apache
- Write out the home page
- Start the Apache service
- Make sure the Apache service starts when the machine boots
- Stop and disable the iptables service

Which resource should we use?

Resources: [About Resources](#) | [Common Functionality](#) — **Resources:** [apt_package](#) | [bash](#) | [batch](#) | [breakpoint](#) | [chef_gem](#) | [chef_handler](#) | [cookbook_file](#) | [cron](#) | [deploy](#) | [directory](#) | [dpkg_package](#) | [dsc_script](#) | [easy_install_package](#) | [env](#) | [erl_call](#) | [execute](#) | [file](#) | [gem_package](#) | [git](#) | [group](#) | [http_request](#) | [ifconfig](#) | [link](#) | [log](#) | [mdadm](#) | [mount](#) | [ohai](#) | [package](#) | [powershell_script](#) | [registry_key](#) | [remote_directory](#) | [remote_file](#) | [route](#) | [rpm_package](#) | [ruby_block](#) | [script](#) | [service](#) | [subversion](#) | [template](#) | [user](#) | [yum_package](#) | [windows_package](#) — **Single Page:** [Resources and Providers](#)

- Install Apache
- Write out the home page
- Start the Apache service
- Make sure the Apache service starts when the machine boots
- Stop and disable the iptables service

The holy trinity of config management

- Install software (**package**)
 - Manipulate files (**template**)
 - Manipulate services (**service**)
-
- You will see this pattern very often

Lab 5 - Install an Apache webserver

- Install Apache - the package is named "**httpd**"
- Write out the home page - **/var/www/html/index.html**
- Manage the apache service - named "**httpd**"
 - Make sure the service is started
 - Make sure the service is enabled to start on boot
- Manage the iptables service - named "**iptables**"
 - Make sure the service is stopped
 - Make sure the service is disabled to start on boot
- What questions can I help you answer?

Checkpoint - Lab 5



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/default.rb`

```
#  
# Cookbook Name:: webserver  
# Recipe:: default  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.  
  
package "httpd"  
  
service "httpd" do  
  action [ :enable, :start ]  
end  
  
template "/var/www/html/index.html" do  
  source "index.html.erb"  
end  
  
service "iptables" do  
  action [ :disable, :stop ]  
end
```

Create a template for index.html

```
$ chef generate template cookbooks/webserver index.html
```

```
Compiling Cookbooks...
Recipe: code_generator::template
  * directory[cookbooks/webserver/templates/default] action create
    - create new directory cookbooks/webserver/templates/default
  * template[cookbooks/webserver/templates/default/index.html.erb]
action create
  - create new file cookbooks/webserver/templates/default/
index.html.erb
  - update content in file cookbooks/webserver/templates/default/
index.html.erb from none to e3b0c4
  (diff output suppressed by config)
```

Checkpoint - Lab 5



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/template/default/index.html.erb`

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

Question: does it work???

- This is a great opportunity for test-driven development
- TDD is a way to automatically verify that your automation is producing expected results
- Test against cheap throwaway infrastructure
 - e.g. inside containers!

Question: does it work???

- ... but... TIME!
- Instead of TDD, today we will focus on Chef integration with AWS
- Let's test by applying to our CentOS workstation
- Time permitting, we can come back to test-driven development

Apply our recipe using chef-client

```
$ chef-client --local-mode -r "recipe[webserver]"
```

```
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: ["webserver"]
Synchronizing Cookbooks:
- webserver
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
* yum_package[httpd] action install
- install version 2.2.15-39.el6.centos of package httpd
* service[httpd] action enable
- enable service service[httpd]
* service[httpd] action start
- start service service[httpd]
* template[/var/www/html/index.html] action create
- create new file /var/www/html/index.html
- update content in file /var/www/html/index.html from none to e043d5
--- /var/www/html/index.html    2015-06-24 04:12:14.626700996 +0000
+++ /tmp/chef-rendered-template20150624-2698-1wmy21s 2015-06-24 04:12:14.625700994 +0000
@@ -1 +1,6 @@
+<html>
+ <body>
+   <h1>Hello, World!</h1>
+ </body>
+</html>
- restore selinux security context
* service[iptables] action disable
- disable service service[iptables]
* service[iptables] action stop
- stop service service[iptables]

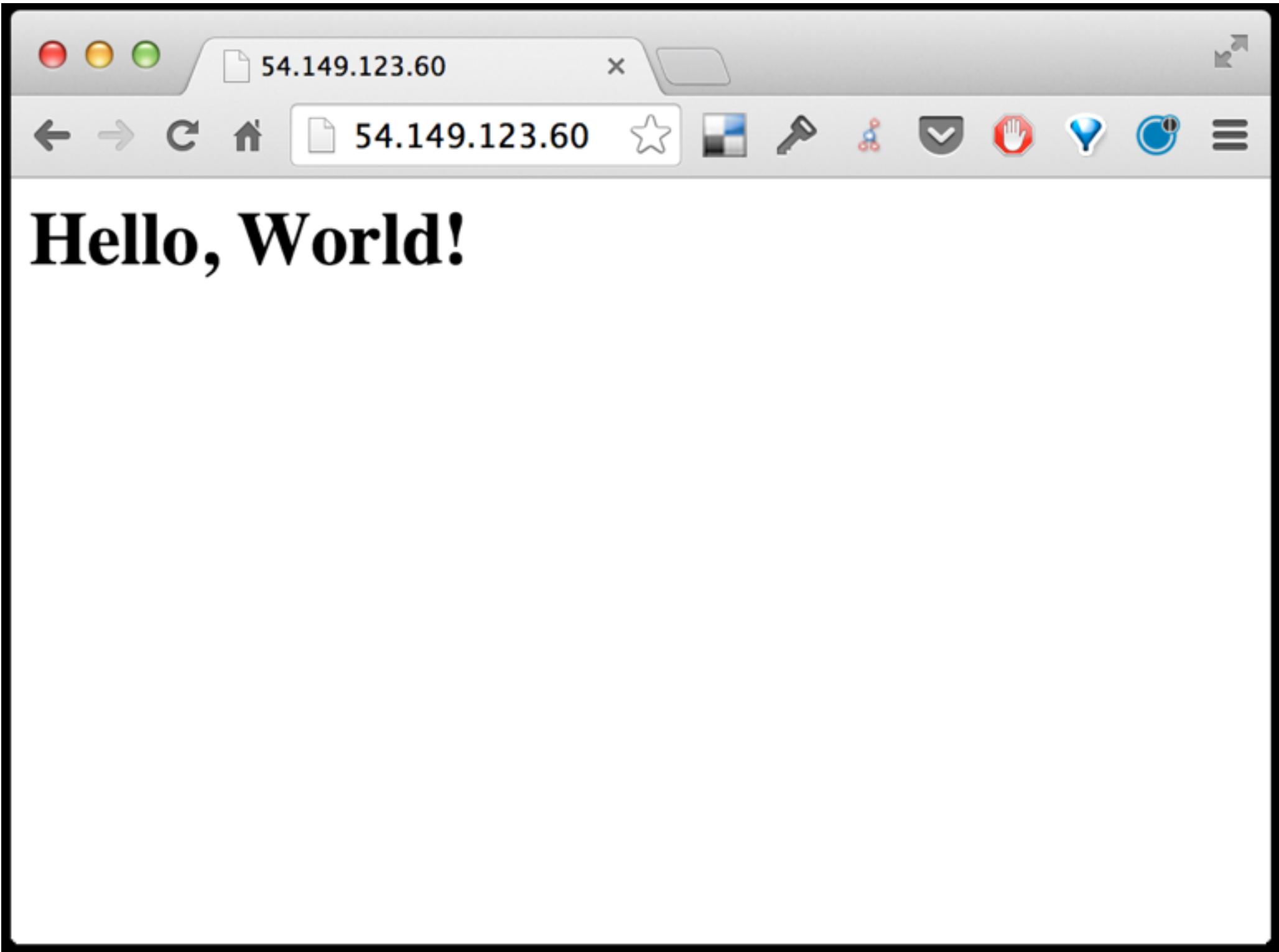
Running handlers:
Running handlers complete
Chef Client finished, 6/6 resources updated in 14.928064857 seconds
```

Display your custom homepage

```
$ curl http://localhost
```

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

Try it in a web browser



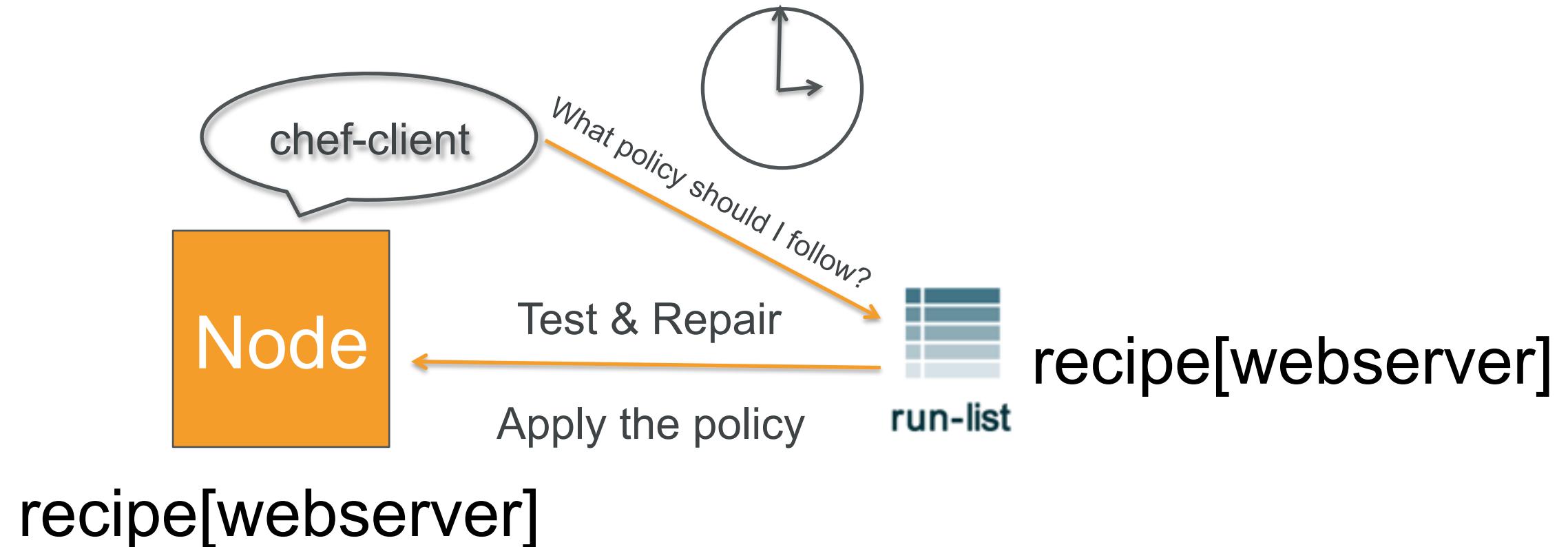
Run chef-client again

```
$ chef-client --local-mode -r "recipe[webserver]"
```

```
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: [ "webserver" ]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
* yum_package[httpd] action install (up to date)
* service[httpd] action enable (up to date)
* service[httpd] action start (up to date)
* template[/var/www/html/index.html] action create (up to date)
* service[iptables] action disable (up to date)
* service[iptables] action stop (up to date)

Running handlers:
Running handlers complete
Chef Client finished, 0/6 resources updated in 7.32233875 seconds
```

chef-client applying desired state



Everything is up to date!

Test & Repair

- Gives you the ability to roll out new desired state
- Helps avoid unintended changes being made to the system
- Prevent manual change
- Prevent natural process of configuration drift
- Encourage a culture of capturing all configuration

Lab 6 - Display custom node data

- **The Problem:** We want our home page to display the hostname of our node
- **Success Criteria:** We can see the hostname of our node in a web browser

How do we get our node name?

- We could hard code the hostname of our node
- That's a bummer because it doesn't scale
- chef-client keeps a record of every node it manages
- We should be able to get the name of any node managed by chef-client dynamically
- We can use the **knife** tool to view this chef-client data

knife

- is an executable program used from the command line to communicate with a Chef server
- can be used in local mode to mock Chef server functionality
- in local-mode is a great way to explore some Chef server functionality without committing to running a Chef server just yet
- is included as a part of the ChefDK

List nodes we know about

```
$ knife node list --local-mode
```

```
WARNING: No knife configuration file found  
ip-172-31-25-147.us-west-2.compute.internal
```

Node data from chef-client

- chef-client saves a node object at the end of a run
- In server mode, the Chef Server can be queried for information about the node
- In local mode, the node object is saved to the local filesystem
- A lot of this information comes from Ohai - we'll see Ohai later....

chef-client on our local vm

- We have a node record for our workstation because we ran chef-client locally
- Again, this sort of recursive management is not the way you'd do it in Production

Each node must have a unique name

- Every node Chef knows about must have a unique name within an organization
- Chef defaults to the *Fully Qualified Domain Name* of the server, i.e. in the format `server.domain.com`
- In EC2, this typically defaults to something that looks like `ip-x-x-x-x.region.compute.internal`

Show node details with knife

```
$ knife node show ip-172-31-25-147.us-west-2.compute.internal --local-mode
```

```
WARNING: No knife configuration file found
Node Name: ip-172-31-25-147.us-west-2.compute.internal
Environment: _default
FQDN: ip-172-31-25-147.us-west-2.compute.internal
IP: 172.31.25.147
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.6
Tags:
```

Pro-Tip: use \$(hostname -f)

- We can get our node's *Fully Qualified Domain Name* in the right format using the hostname command
- hostname -f returns the full FQDN

Show node details with knife

```
$ knife node show $(hostname -f) --local-mode
```

```
WARNING: No knife configuration file found
Node Name: ip-172-31-25-147.us-west-2.compute.internal
Environment: _default
FQDN: ip-172-31-25-147.us-west-2.compute.internal
IP: 172.31.25.147
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.6
Tags:
```

What is the Node object

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

Where is the Node object

- In server mode, node objects are stored and indexed on the Chef Server
- In local mode, node objects are stored on the local filesystem
- Node objects are stored in JSON

List existing node objects

```
$ ls ~/chef-repo/nodes
```

```
ip-172-31-25-147.us-west-2.compute.internal.json
```

Peek inside the node object

```
$ less ~/chef-repo/nodes/$(hostname -f).json
```

```
{
  "name": "ip-172-31-7-47.us-west-2.compute.internal",
  "normal": {
    "tags": [
      ]
  },
  "automatic": {
    "keys": {
      "ssh": {
        "host_dsa_public": "AAAAB3NzaC1kc3MAAACBAIKx4i9XbyCQVvt2vSt1kgNwLhHm5+kRHN3z
5s8q+zseajCk+OxVoMzWAY4fj/5UOXpmZ2IJuGMF21RouO7HcZorNxOK5Y/1ABOUHec3NdZKBuPJcR14774F
r2UpMFu5ugPlA0w+jgsS4HygHAharE2o5EFFrH9v09Pc0OTYqhpXAAAAFQDHer5ZDB1Jze5Vc5hokWNKxkrS
XwAAIAudW/pKR9Dh0G4Gqfw5hXdav/4tPQswvWpWorlThvlHTGfGnd8/W09jKFfTT/YMwqtvBLBqXbQApH6
kCMa8SfNXVJwpI6TTusRTV2MJrPSp/lnJ3Ya/Qan/QFvI+axgTmhh9qZNEp9LiUzyOHrjLsoP8K38JX89A1j
5gIFjw4gZgAAAIWRjt7EuD3S+rdkzqzWiXXy8t8taLMednUqV1cpweWg6nKzcIk0EG5r4Anp7VYatTaG9Ur
cR1XzCFsVgFULKA42r1skz1u8XLwZVhFR57j4A+NqyEiN17LHzPeQ/sfEVrZ67zWqkaxF3EdsqFMH1zhoE0q
IZYGjOwTg6jbgiEIIdw=="
    }
  }
}
```

Where does this data come from?

- Ohai is a discovery agent included with Chef
- It is executed every single time chef-client is run
- Extensible plugin model allows custom data
- Chef uses some information (e.g. *platform*) to make decisions about how to implement resources

Package Resource

```
package "git"
```

```
yum install git
```

```
apt-get install git
```

```
pacman sync git
```

```
pkg_add -r git
```

Providers are determined by node's platform

Ohai

```
"languages": {
  "ruby": {
    },
  "perl": {
    "version": "5.14.2",
    "archname": "x86_64-linux-gnu-thread-multi"
  },
  "python": {
    "version": "2.6.6",
    "builddate": "Jul 10
2013, 22:48:45"
  },
  "perl": {
    "version": "version",
    "archname": "x86_64-
linux-thread-multi"
  },
  "lua": {
    "version": "5.1.4"
  }
},
"kernel": {
  "name": "Linux",
  "release": "3.2.0-32-virtual",
  "version": "#1 SMP Wed Oct 16
18:37:12 UTC 2013",
  "machine": "x86_64",
  "modules": {
    "isofs": {
      "size": "70066",
      "refcount": "2"
    },
    "des_generic": {
      "size": "16604",
      "refcount": "0"
    }
  },
  "os": "GNU/Linux"
},
"os": "linux",
"os_version": "2.6.32-358.23.2.el6.x86_64",
"ohai_time": 1389105685.7735305,
"network": {
  "interfaces": {
    "lo": {
      "mtu": "16436",
      "flags": [
        "LOOPBACK", "UP", "LOWER_UP"
      ],
      "encapsulation": "Loopback",
      "addresses": {
        "127.0.0.1": {
          "family": "inet",
          "netmask": "255.0.0.0",
          "scope": "Node"
        },
        "::1": {
          "family": "inet6",
          "scope": "Node"
        }
      }
    },
    "eth0": {
      "type": "eth",
      "number": "0",
      "mac": "00:0C:29:4D:4E:00"
    }
  }
}
```

Run Ohai on node

```
$ ohai | more
```

```
{
  "languages": {
    "ruby": {
      "platform": "x86_64-linux",
      "version": "2.1.4",
      "release_date": "2014-10-27",
      "target": "x86_64-unknown-linux-gnu",
      "target_cpu": "x86_64",
      "target_vendor": "unknown",
      "target_os": "linux",
      "host": "x86_64-unknown-linux-gnu",
      "host_cpu": "x86_64",
      "host_os": "linux-gnu",
      "host_vendor": "unknown",
      "bin_dir": "/opt/chefdk/embedded/bin",
      "ruby_bin": "/opt/chefdk/embedded/bin/ruby",
      "language": "Ruby"
    }
  }
}
```

Node data

- To Chef, your infrastructure is just large hashes of data stored as JSON
- Much of that data comes from Ohai
 - e.g. The "languages" hash is detected via Ohai
- Some of that data is generated during a chef-client run
 - e.g. node name is generated by chef-client
- All of this data is indexed and available for search
- When using a Chef Server, all of your infrastructure configuration data is aggregated in one central location

Show specific attributes with knife

```
$ knife node show $(hostname -f) --local-mode -a ipaddress
```

```
WARNING: No knife configuration file found  
ip-172-31-7-47.us-west-2.compute.internal:  
  ipaddress: 172.31.7.47
```

Show specific attributes with knife

```
$ knife node show $(hostname -f) --local-mode -a cpu.0.model_name
```

```
WARNING: No knife configuration file found  
ip-172-31-7-47.us-west-2.compute.internal:  
cpu.0.model_name: Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
```

Show specific attributes with knife

```
$ knife node show $(hostname -f) --local-mode -a name
```

```
WARNING: No knife configuration file found
ip-172-31-25-147.us-west-2.compute.internal:
  name: ip-172-31-25-147.us-west-2.compute.internal
```

Include node name in the index.html template



OPEN IN EDITOR: ~/chef-repo/cookbooks/webserver/templates/default/index.html.erb

```
<html>
  <body>
    <h1>Hello from <%= node.name %></h1>
  </body>
</html>
```

Apply our recipe using chef-client

```
$ chef-client --local-mode -r "recipe[webserver]"
```

```
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
* yum_package[httpd] action install (up to date)
* service[httpd] action enable (up to date)
* service[httpd] action start (up to date)
* template[/var/www/html/index.html] action create
  - update content in file /var/www/html/index.html from e043d5 to c1d241
    --- /var/www/html/index.html 2015-06-24 04:56:04.354558911 +0000
    +++ /tmp/chef-rendered-template20150624-2027-53edcr 2015-06-24 04:56:21.195558911 +0000
    @@ -1,6 +1,5 @@
      <html>
        <body>
          - <h1>Hello, World!</h1>
          + <h1>Hello from ip-172-31-25-147.us-west-2.compute.internal
        </body>
      </html>
```

Congratulations!

- You have just deployed a dynamic website!



What could we do if...?

- We wanted to show network routes in a service status page?
- We needed to place this node's instance-id in a config file?

Summary

- How is infrastructure referenced by Chef?
- Where do we get most data in those JSON hashes?
- Which three resources are the most commonly found resources in configuration management?

Summary

- What questions can I help you answer?

Applying desired state to new EC2 instances

Chef Provisioning for AWS

Lesson Objectives

- After completing the lesson, you will be able to
 - Use Chef Provisioning to create and manage remote servers
 - Use Chef Provisioning to create and manage EC2 security groups
 - Describe the Chef bootstrapping process
 - Use Chef Provisioning to manage servers in parallel

Getting the hang of things

- So far, we've been making changes to our virtual workstation
- What if we destroy our workstation?
- Recursive management is not how you should normally do it
- Let's use Chef to provision new ec2 servers for us to manage

Chef Provisioning

- Manage machines and infrastructure components via new custom resources
- Handy for basic infrastructure "orchestration"
- Describe your infrastructure topology in a recipe
 - Test and repair also applies to infrastructure
 - <https://github.com/chef/chef-provisioning>
- Previously known as "chef-metal"

Chef Provisioning

- Plugin model lets you write resources for your own infrastructure (e.g. VirtualBox, EC2, LXC, bare metal, etc)
- Today, we will use the AWS driver for Chef Provisioning
 - <https://github.com/chef/chef-provisioning-aws>
 - Check out the docs/examples section
- Announced at re:Invent 2014

Chef Provisioning AWS

- Can manage more than just ec2 machines (more on that later)
- Can manage those resources across multiple AWS regions
- Can apply different options to different subsets of resources
- Is similar to CloudFormation
- Unlike JSON, allows you to programmatically manage your Infrastructure as Code

First, we need AWS access

- In today's workshop, we are going to use a shared AWS account
- You can use your own credentials for experimentation after we're done
 - We may not have time to troubleshoot AWS account settings if you use your own
 - You will need access to manage: Security Groups, EC2 Instances, Elastic Load Balancers

Edit your AWS config



OPEN IN EDITOR: ~/.aws/config

```
[default]
aws_access_key_id = AKIAAAABCCDDEEFFGGHH
aws_secret_access_key = "Abc0123dEf4GhIjk51Mn/OpQrSTUvXyz/A678bCD"
```

- Values on the slide are not real - cut/paste values from here
 - <http://bit.ly/popup-chef-creds>
- These are IAM credentials with EC2 and ELB access
- Credentials only used for this workshop - they'll be blown away afterward

SAVE FILE!

Next, let's use it

- Infrastructure management patterns in AWS may be complex
- Chef Provisioning allows you to combine provisioning, running system commands, managing content, and more to model deployment scenarios to meet your needs
- We will try some basic use cases to demonstrate the idea behind the Chef Provisioning for AWS driver

Lab 7 - Create a new webserver

- **The Problem:** We need test our webserver recipe somewhere that's not our workstation
- **Success Criteria:** We can see our custom homepage in a web browser on a different instance

Required steps

- Create a new recipe in the webserver cookbook
- Create a new security group with port 22/tcp open
- Create a new security group with port 80/tcp open
- Create a new ec2 instance
- Apply recipe[webserver] to that instance's run_list

Generate a new recipe

```
$ chef generate recipe cookbooks/webserver provision
```

```
Compiling Cookbooks...
Recipe: code_generator::recipe
  * directory[cookbooks/webserver/spec/unit/recipes] action create (up to date)
  * cookbook_file[cookbooks/webserver/spec/spec_helper.rb] action create_if_missing (up to date)
  * template[cookbooks/webserver/spec/unit/recipes/provision_spec.rb] action create_if_missing
    - create new file cookbooks/webserver/spec/unit/recipes/provision_spec.rb
    - update content in file cookbooks/webserver/spec/unit/recipes/provision_spec.rb from none to 58eeb1
      (diff output suppressed by config)
  * template[cookbooks/webserver/recipes/provision.rb] action create
    - create new file cookbooks/webserver/recipes/provision.rb
    - update content in file cookbooks/webserver/recipes/provision.rb from none to b18bca
      (diff output suppressed by config)
```

Recipe Naming

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

Exercise: Edit the default recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
#  
# Cookbook Name:: webserver  
# Recipe:: provision  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.
```

Set a unique name



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
#  
# Cookbook Name:: webserver  
# Recipe:: provision  
#  
# Copyright (c) 2015 The Authors, All Rights Reserved.
```

```
name = "YOURNAME"
```

- Substitute your name (or a workshop unique string) here
- We are using this to generate unique resource names in our shared AWS account
- To use this value in a string, call it as "**#{name}**"

Specify how to use chef provisioning



OPEN IN EDITOR: ~chef-repo/cookbooks/webserver/recipes/provision.rb

```
name = "YOURNAME"

require "chef/provisioning/aws_driver"

with_driver "aws::us-east-1" do
end
```

- States that we are using the AWS driver
- Then specifies a regional sub-driver

Create new Security Groups and rules



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
with_driver "aws::us-east-1" do

  aws_security_group "#{name}-ssh" do
    inbound_rules '0.0.0.0/0' => 22
  end

  aws_security_group "#{name}-http" do
    inbound_rules '0.0.0.0/0' => 80
  end

end
```

- The AWS driver provides a new resource "`aws_security_group`"
- We set two groups to demo functionality

Add global ec2 options



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
inbound_rules '0.0.0.0/0' => 80
end

with_machine_options({
  :aws_tags => {"belongs_to" => name},
  :ssh_username => "root",
  :bootstrap_options => {
    :image_id => "ami-bf5021d6",
    :instance_type => "t1.micro",
    :key_name => "aws-popup-chef",
    :security_group_ids => [ "#{name}-ssh", "#{name}-http" ]
  },
  :convergence_options => {
    :chef_version => "12.2.1",
  }
})

end
```

with_machine_options

```
with_machine_options({  
  :aws_tags => {"belongs_to" => name},  
  :ssh_username => "root",  
  :bootstrap_options => {  
    :image_id => "ami-bf5021d6",  
    :instance_type => "t1.micro",  
    :key_name => "aws-popup-chef",  
    :security_group_ids => ["#{name}-ssh", "#{name}-http"]  
  },  
  :convergence_options => {  
    :chef_version => "12.2.1",  
  }  
})
```

- Allows us to define options all of our machines will inherit
- Can set individual machine options to override
- "**aws_tags**" applies a hash to ec2-tagable resources

with_machine_options

```
with_machine_options({
  :aws_tags => {"belongs_to" => name},
  :ssh_username => "root",
  :bootstrap_options => {
    :image_id => "ami-bf5021d6",
    :instance_type => "t1.micro",
    :key_name => "aws-popup-chef",
    :security_group_ids => ["#{name}-ssh", "#{name}-http"]
  },
  :convergence_options => {
    :chef_version => "12.2.1",
  }
})
```

- SSH is the bootstrap transport mechanism (more later)
- CentOS 6.5 AMI in AWS Marketplace
- Private SSH key is in your ~/.ssh directory
- We can also adjust any other Chef installation parameters

Create a new ec2 machine



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
:bootstrap_options => {
    :image_id => "ami-bf5021d6",
    :instance_type => "t1.micro",
    :key_name => "aws-popup-chef",
    :security_group_ids => [ "#{name}-ssh", "#{name}-http" ]
},
:convergence_options => {
    :chef_version => "12.2.1",
}
}

machine "#{name}-webserver-1" do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
end
```

machine resource

```
machine "#{name}-webserver-1" do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
```

- Defines a new EC2 instance with the AWS driver
- Apply "recipe[webserver]" to the node's run_list
- Apply Chef tag "my-webserver"
- Always converge this node by telling it to run chef-client
- Inherits from with_machine_options if no options are specified

The provision recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
name = "YOURNAME"

require "chef/provisioning/aws_driver"

with_driver "aws::us-east-1" do

  aws_security_group "#{name}-ssh" do
    inbound_rules '0.0.0.0/0' => 22
  end

  aws_security_group "#{name}-http" do
    inbound_rules '0.0.0.0/0' => 80
  end

  with_machine_options({
    :aws_tags => {"belongs_to" => name},
    :ssh_username => "root",
    :bootstrap_options => {
      :image_id => "ami-bf5021d6",
      :instance_type => "t1.micro",
      :key_name => "aws-popup-chef",
      :security_group_ids => ["#{name}-ssh", "#{name}-http"]
    },
    :convergence_options => {
      :chef_version => "12.2.1",
    }
  })

  machine "#{name}-webserver-1" do
    recipe "webserver"
    tag "my-webserver"
    converge true
  end
end
```

- Replace **YOUR_NAME** with a unique string
- Replace **image_id** with a CentOS 6 AMI if you didn't use us-east-1
- Replace **key_name** with a valid SSH key name from your own AWS account
- If you are feeling hardcore, type it
- <http://bit.ly/provision1>

Run the webserver::provision recipe

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: [ "webserver::provision" ]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 3 resources
Recipe: webserver::provision
* aws_security_group[YOURNAME-ssh] action create
  - Creating new SG YOURNAME-ssh in us-east-1
  - authorize 0.0.0.0/0 to send traffic to group YOURNAME-ssh (sg-66917f01) on port_range 22..22 with protocol tcp
* aws_security_group[YOURNAME-http] action create
  - Creating new SG YOURNAME-http in us-east-1
  - authorize 0.0.0.0/0 to send traffic to group YOURNAME-http (sg-65917f02) on port_range 80..80 with protocol tcp
* machine[YOURNAME-webserver-1] action converge
  - Create YOURNAME-webserver-1 with AMI ami-bf5021d6 in us-east-1
  - applying tags {"belongs_to"=>"YOURNAME"}
  - create node YOURNAME-webserver-1 at chefzero://localhost:8889
  - add normal.tags = ["my-webserver"]
```

recipe output (continued)

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
- add normal.chef_provisioning = {"reference"=>{"driver_version"=>"1.2.1", "allocated_at"=>"2015-06-24 06:16:25 UTC", "host_node"=>"chefzero://localhost:8889/nodes/", "image_id"=>"ami-bf5021d6", "instance_id"=>"i-7c36f4af", "key_name"=>"aws-popup-chef", "ssh_username"=>"root"}, "driver_url"=>"aws::us-east-1"}
- update run_list from [] to ["recipe[webserver]"]
- waiting for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be ready ...
- been waiting 0/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be ready ...
- been waiting 10/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be ready ...
- been waiting 20/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be ready ...
- YOURNAME-webserver-1 is now ready
- waiting for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be connectable (transport up and running) ...
- been waiting 0/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be connectable ...
- been waiting 10/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be connectable ...
- been waiting 20/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be connectable ...
- been waiting 30/120 -- sleeping 10 seconds for YOURNAME-webserver-1 (i-7c36f4af on aws::us-east-1) to be connectable ...
- YOURNAME-webserver-1 is now connectable[2015-06-24T06:18:59+00:00] ERROR: Error with SCP: SCP did not finish successfully
(1): scp: /etc/chef/client.pem: No such file or directory

- generate private key (2048 bits)
- create directory /etc/chef on YOURNAME-webserver-1
```

recipe output (continued)

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
- write file /etc/chef/client.pem on YOURNAME-webserver-1
- create client YOURNAME-webserver-1 at clients
- create client YOURNAME-webserver-1 at clients
-   add public_key = "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqybM38dCPZY5vj4YIUMN
\ngCVpVaAp7FWMeD2SNwxxxyDhHZWKei8nRcy09w6i0+1ETbzT5Bj13ejH77agX6CQE\nPLENwfXi4ZU4Eq/
QNQpD32R0qlgw1iZ3XIhx7+bt8+J24K64yDXdy/j8dUCD6Qx\n7BN0TjaleHrjJjrXzNlvk
+javTFgJCFvM9nJzmj3eswu0sYnGRw84zE56jkRiHrw
\nGa4bfB9FxDgBva0kVeQiYUNEv58O9bbQJ4zYPx9xi1EVNP7xbOWZzNRiwKShokEG\nmuWz209PDorZq+zsB/
HGiOXxTrIiVHreD04UA3Ogy0/+AzjdGSNO7hOrK2gumgw\nEwIDAQAB\n-----END PUBLIC KEY-----\n"
- create directory /etc/chef/ohai/hints on YOURNAME-webserver-1
- write file /etc/chef/ohai/hints/ec2.json on YOURNAME-webserver-1
- write file /etc/chef/client.rb on YOURNAME-webserver-1
- write file /tmp/chef-install.sh on YOURNAME-webserver-1
- run 'bash -c ' bash /tmp/chef-install.sh -v 12.2.1'' on YOURNAME-webserver-1
```

recipe output (continued)

```
$ chef-client --local-mode -r 'recipe[webserver>::provision']'
```

```
Starting Chef Client, version 12.2.1
resolving cookbooks for run list: ["webserver"]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
  * yum_package[httpd] action install
    - install version 2.2.15-39.el6.centos of package httpd
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
    - start service service[httpd]
  * template[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to e9fcf3
      --- /var/www/html/index.html      2015-06-24 06:19:53.618880140 +0000
      +++ /tmp/chef-rendered-template20150624-1054-av5vdg      2015-06-24 06:19:53.617880140 +0000
      @@ -1 +1,6 @@
      +<html>
      + <body>
      +   <h1>Hello from YOURNAME-webserver-1
```

recipe output (continued)

```
$ chef-client --local-mode -r 'recipe[webserver>::provision']'
```

```
      - restore selinux security context
* service[iptables] action disable
  - disable service service[iptables]
* service[iptables] action stop
  - stop service service[iptables]
```

Running handlers:

Running handlers complete

Chef Client finished, 6/6 resources updated in 22.119149318
seconds

- run 'chef-client -l auto' on YOURNAME-webserver-1

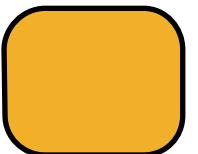
Running handlers:

Running handlers complete

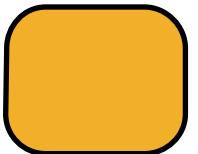
Chef Client finished, 3/3 resources updated in 216.931960867 seconds

What just happened?

```
chef-client --local-mode -r 'recipe[webserver::provision]'
```



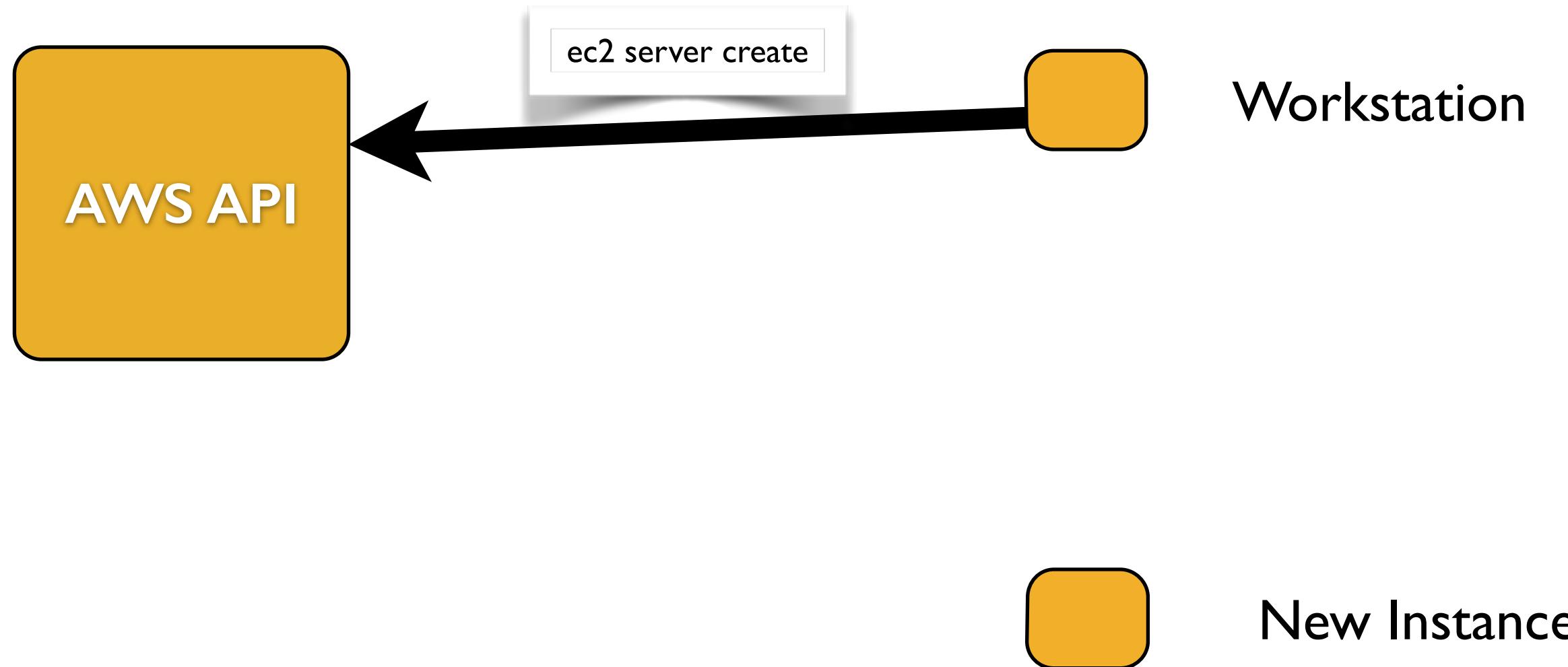
Workstation



New Instance

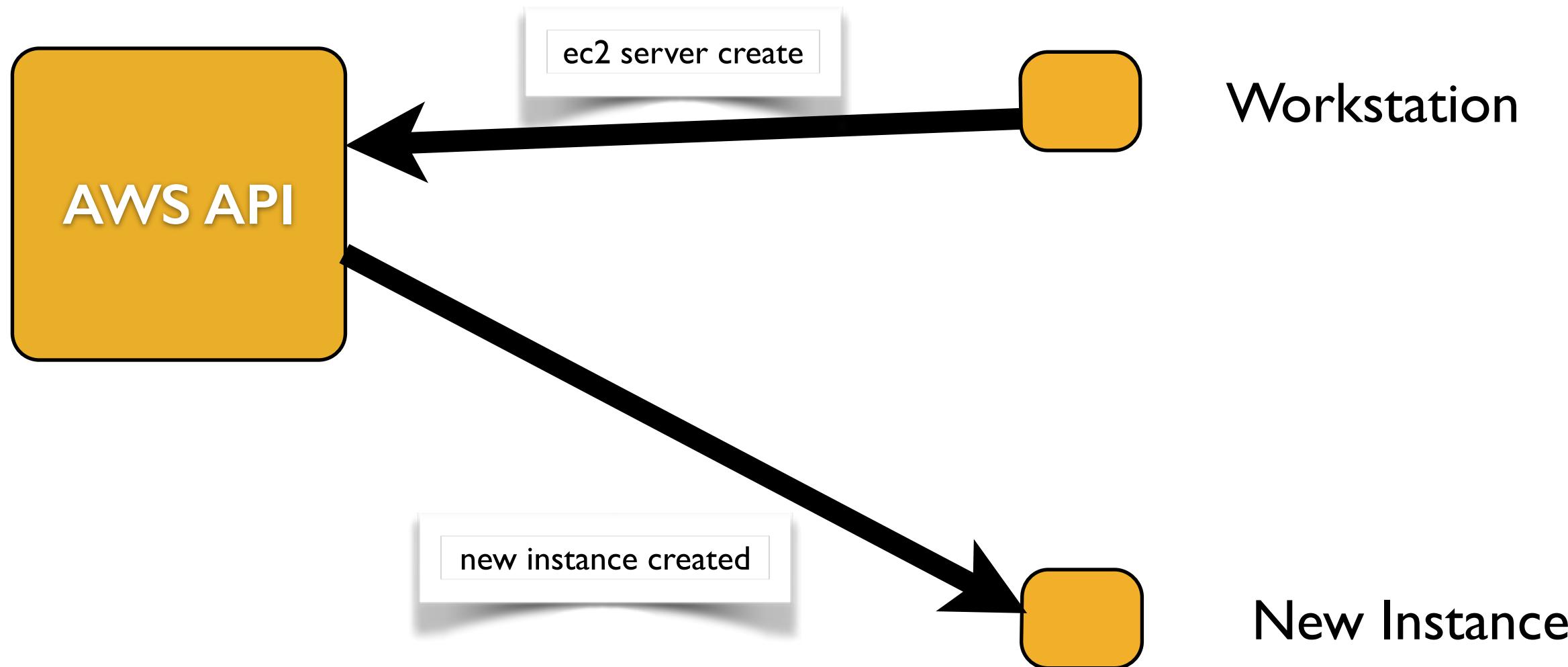
What just happened?

```
chef-client --local-mode -r 'recipe[webserver::provision]'
```



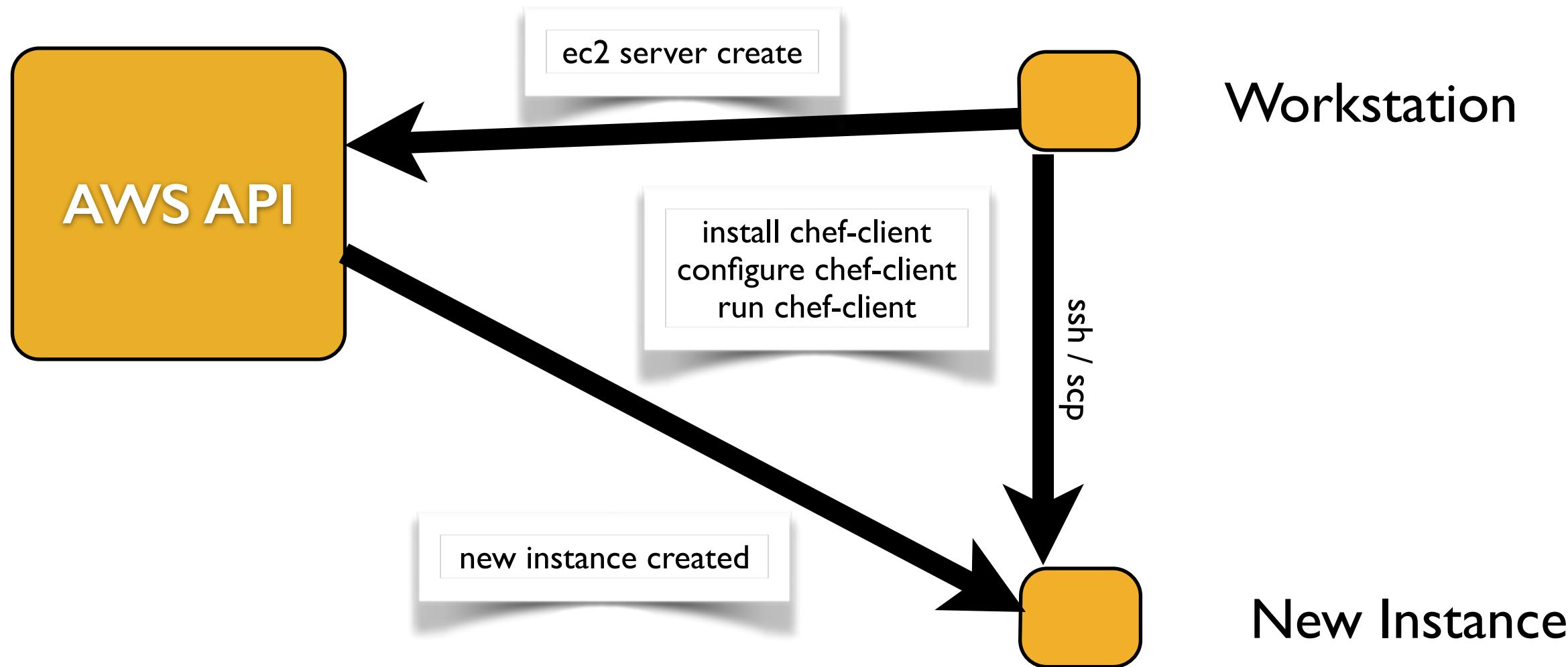
What just happened?

```
chef-client --local-mode -r 'recipe[webserver::provision]'
```



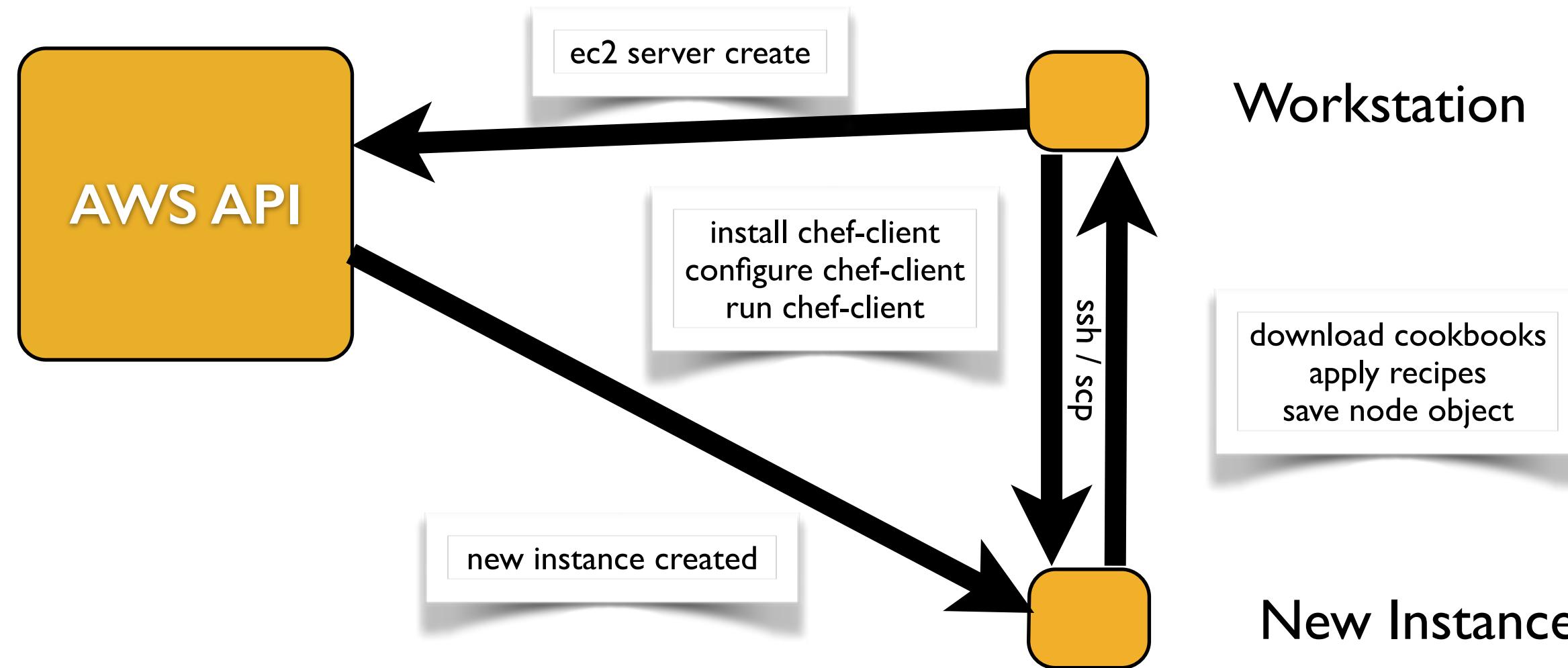
What just happened?

```
chef-client --local-mode -r 'recipe[webserver::provision]'
```



What just happened?

```
chef-client --local-mode -r 'recipe[webserver::provision]'
```



What just happened?

- Bootstrap script downloads Chef packages from
<http://chef.io/chef/install.sh>
- In this case, chef-client is configured to use your workstation as a mock Chef Server
- New EC2 downloads cookbooks, applies the recipes, and saves a node object back to your workstation

Where is the new instance?

- Remember, there's a new node object back on our workstation
- We could look inside the node object JSON
- We can retrieve the specific attribute we want via the knife command

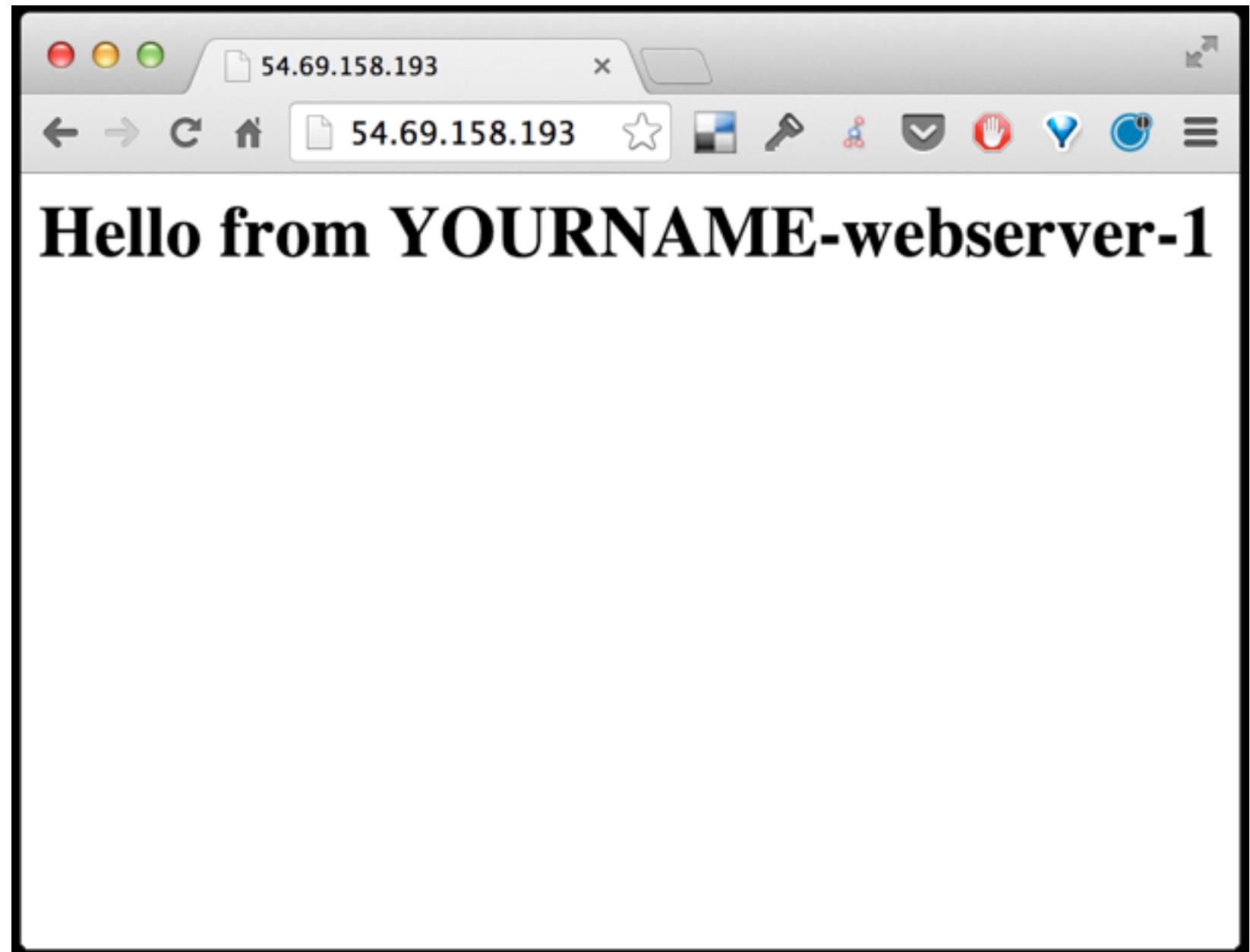
Find your new node's IP

```
$ knife node show YOURNAME-webserver-1 --local-mode -a ec2.public_ipv4
```

```
WARNING: No knife configuration file found  
YOURNAME-webserver-1:  
ec2.public_ipv4: 54.69.158.193
```

Congratulations!

- You have just deployed your EC2 node with Chef!



Lab 8 - Create a new 2 more webservers

- **The Problem:** Why only have one when someone else is paying for it? Create 2 additional webservers
- **Success Criteria:** We can see content from our additional web servers in a web browser

Required steps

- Edit the `webserver::provision` recipe
- Add two additional **machine** resources
- Name the new machines `#{{name}}-webserver-2` and `#{{name}}-webserver-3`
- Reuse the **my-webserver** tag (we will use this later)
- What questions can I help you answer?

Edit the provision recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
machine "#{name}-webserver-1" do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
```

```
machine "#{name}-webserver-2" do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
```

```
machine "#{name}-webserver-3" do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
```

```
end
```

Run the webserver::provision recipe

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
[2015-01-11T21:45:51+00:00] WARN: No config file found or specified on command line, using command line options.
Starting Chef Client, version 11.18.0.rc.1
resolving cookbooks for run list: ["webserver::provision"]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 5 resources
Recipe: webserver::provision
 * aws_security_group[YOURNAME-ssh] action create
 * aws_security_group[YOURNAME-http] action create
 * machine[YOURNAME-webserver-1] action converge[2015-01-11T21:45:58+00:00] WARN: Server YOURNAME-webserver-1 was created with SSH
username and machine_options specifies username root. Using root. Please edit the node and change the
chef_provisioning.location.ssh_username attribute if you want to change it.
(up to date)
 * machine[YOURNAME-webserver-2] action converge
  - Create YOURNAME-webserver-2 with AMI ami-b6bdde86 in us-west-2
  - create node YOURNAME-webserver-2 at http://localhost:8889
  - add normal.tags = ["my-webserver"]
  - add normal.chef_provisioning = {"location"=>{"driver_url"=>"aws:aws", "driver_version"=>"0.1.3", "allocated_at"=>"2015-01-11
21:46:01 UTC", "host_node"=>"http://localhost:8889/nodes/", "image_id"=>"ami-b6bdde86", "instance_id"=>"i-b7ea8dbb"}}
```

But we can go faster

- While that works, it's pretty repetitive
 - We can use a simple loop to shorten our code
- It's also serial... webserver 1 converges, then we create webserver2, then webserver 3 once webserver2 completes
 - We can use the **machine_batch** resource to make them go in parallel

Using loops

- What if you want to modify some aspect of all your machines, such as by changing their tags or the recipes they run?
- We can easily forget to update all of them
- If you add a fourth or even a fifth machine, you'll have even more code to maintain
- Chef is built on Ruby, so you can use a loop to create identical servers.

Edit the provision recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
:convergence_options => {
  :chef_version => "12.2.1",
}

1.upto(3) do |n|
  instance = "#{name}-webserver-#{n}"
  machine instance do
    recipe "webserver"
    tag "my-webserver"
    converge true
  end
end
end
```

machine_batch

- Provisioning clusters with a lot of machines would be horrifically slow if you had to do it one machine at a time
- machine_batch will provision all machines defined as attributes in parallel
- <https://github.com/chef/chef-provisioning/tree/master/docs/blogs>

Edit the provision recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
:convergence_options => {
  :chef_version => "12.2.1",
}

machine_batch do
  1.upto(3) do |n|
    instance = "#{name}-webserver-#{n}"
    machine instance do
      recipe "webserver"
      tag "my-webserver"
      converge true
    end
  end
end
end
```

Edit the provision recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
:convergence_options => {
  :chef_version => "12.2.1",
}

machine_batch do
  1.upto(3) do |n|
    instance = "#{name}-webserver-#{n}"
    machine instance do
      recipe "webserver"
      tag "my-webserver"
      converge true
    end
  end
end
end
```

- If you are feeling hardcore, type it
- <http://bit.ly/provision2>

Run the webserver::provision recipe

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
[YOURNAME-webserver-2] resolving cookbooks for run list: ["webserver"]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 4 resources
Recipe: webserver::default
  * yum_package[httpd] action install
[YOURNAME-webserver-3] (up to date)
  * service[httpd] action enable (up to date)
  * service[httpd] action start (up to date)
  * template[/var/www/html/index.html] action create (up to date)
  * service[iptables] action disable
[YOURNAME-webserver-1] (up to date)
  * service[httpd] action enable
[YOURNAME-webserver-3] (up to date)
  * service[iptables] action stop
[YOURNAME-webserver-1] (up to date)
  * service[httpd] action start
[YOURNAME-webserver-3] (up to date)
[YOURNAME-webserver-1] (up to date)
  * template[/var/www/html/index.html] action create (up to date)
  * service[iptables] action disable
```

Summary

- Even though we already provisioned our machines, we can see an improvement in speed with parallel convergence output from chef-client
- **machine_batch** will wait for all machines in the code block to complete successfully before moving on with processing your recipe

Extra Credit Lab

- Write a recipe to destroy your webservers
- Run the provision recipe to see a big improvement in speed
- Take an extended break!

Desired state for other AWS Services

Beyond EC2

Lesson Objectives

- After completing the lesson, you will be able to
 - Use Chef Provisioning to manage ELBs
 - Associate or dissociate instances with a load balancer from a Chef recipe
 - Describe the state of managing other AWS services with Chef Provisioning's AWS driver

Chef Provisioning AWS

- So far, we've only used Chef Provisioning's AWS driver to manage ec2 instances
- Application stacks in AWS often rely on other AWS services
- Similar to CloudFormation, you can manage AWS resources beyond ec2 with Chef Provisioning
- Unlike JSON, Chef Provisioning allows you to programmatically manage your entire application topology as **Code**

A simple example

- While this example is simple, we can demonstrate how multiple AWS services may be managed and how interdependencies passed between them
- Time permitting, we can try to build something a bit more useful at the end of class

Lab 9 - Create an ELB for your webservers

- **The Problem:** We need to balance traffic across all of our webservers
- **Success Criteria:** We can start see all of our web servers serving traffic behind an ELB

Required steps

- Create a list of all the webserver instances that you have created
- Use the **load_balancer** resource to create a new ELB
- Pass the list of your webservers to the **load_balancer** resource

Create a list of instances



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
:convergence_options => {
  :chef_version => "12.2.1",
}
}

# track all the instances we need to make
webservers = 1.upto(3).map { |n| "#{name}-webserver-#{n}"}

machine batch do
  webservers.each do |instance|
    machine instance do
      recipe "webserver"
      tag "my-webserver"
      converge true
    end
  end
end
end
```

Create a load balancer resource



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
machine instance do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
end
end

load_balancer "#{name}-webserver-lb" do
  machines webservers
end
end
```

- The `load_balancer` resource manages ELBs in AWS
- You may set `load_balancer_options` for `load_balancers`

Create a load balancer resource



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/provision.rb`

```
machine instance do
  recipe "webserver"
  tag "my-webserver"
  converge true
end
end
end
```

```
load_balancer "#{name}-webserver-lb" do
  machines webservers
end
end
```

- If you are feeling hardcore, type it
- <http://bit.ly/provision3>

- The **load_balancer** resource manages ELBs in AWS
- You may set **load_balancer_options** for load_balancers

Run the webserver::provision recipe

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
* load_balancer[YOURNAME-webserver-lb] action create
  - create load balancer YOURNAME-webserver-lb in us-east-1
  - attach subnets #<AWS::EC2::Subnet:0x000000079b3050>, #<AWS::EC2::Subnet:
0x00000007b888d0>, #<AWS::EC2::Subnet:0x00000007d610d0>
  - with listeners
[{:protocol=>:http, :port=>80, :instance_port=>80, :instance_protocol=>:http}]
  - add machines YOURNAME-webserver-1, YOURNAME-webserver-2, YOURNAME-webserver-3
  - create data bag item YOURNAME-webserver-lb at chefzero://localhost:8889
  - add reference = {"driver_version"=>"1.2.1", "allocated_at"=>"2015-06-24 07:09:32
UTC"}
  - add driver_url = "aws::us-east-1"
```

Running handlers:

Running handlers complete

Chef Client finished, 2/4 resources updated in 19.987331511 seconds

Where is our ELB?

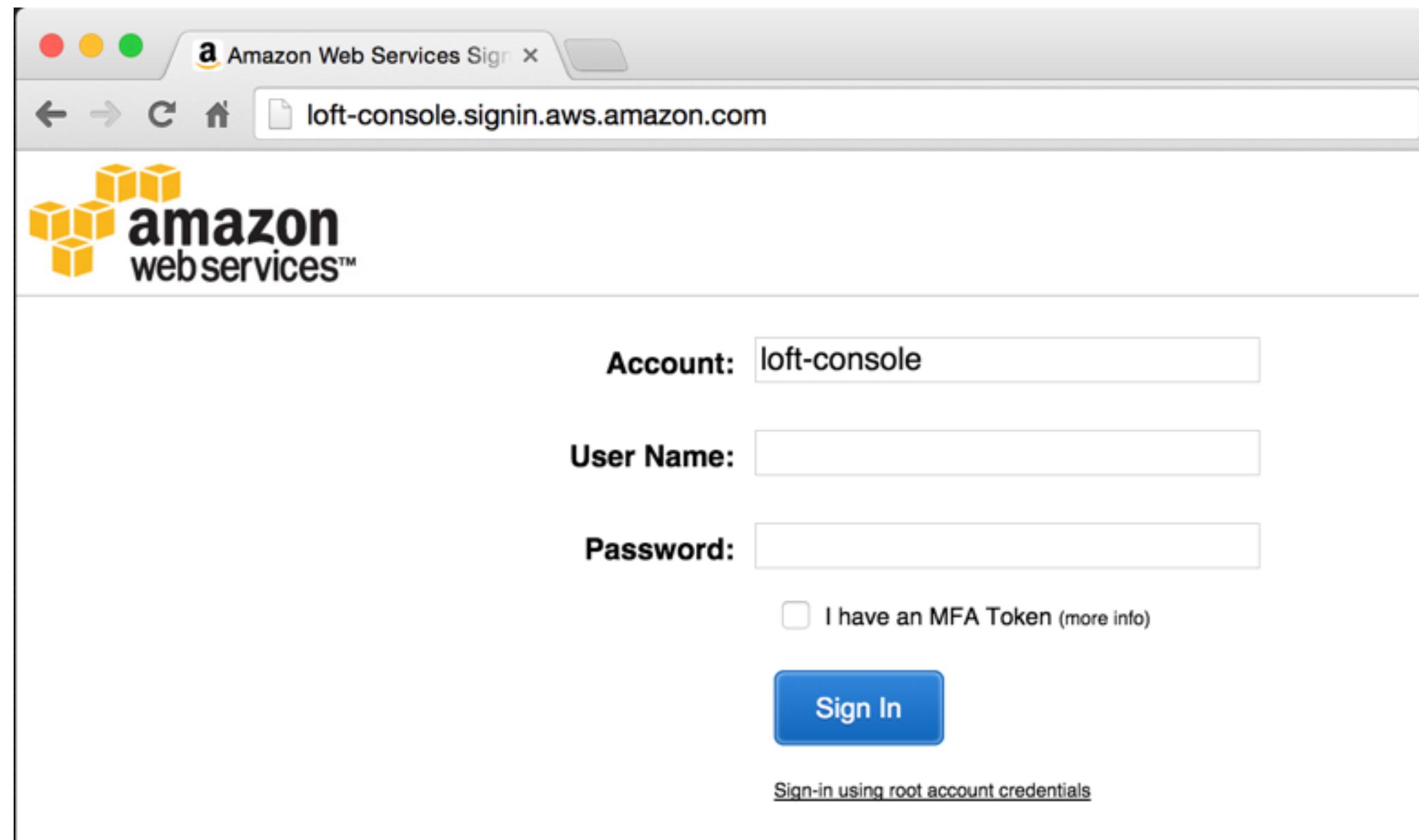
- Your servers are now load balanced
- But how do we verify that?
- Go up-vote this issue... go ahead, we'll wait :-)
 - [https://github.com/chef/chef-provisioning-aws/](https://github.com/chef/chef-provisioning-aws/issues/27)
issues/27
- Today, the best way to get the public DNS name for the new load balancer is from the EC2 Management Console

IAM to the rescue

<http://loft-console.signin.aws.amazon.com>

user: loft-console

pass: chef&aws2015



Get the ELB hostname

The screenshot shows the AWS EC2 Management Console interface. On the left, there's a sidebar with navigation links: EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES (with sub-links Instances, Spot Requests, Reserved Instances), IMAGES (with sub-links AMIs, Bundle Tasks), ELASTIC BLOCK STORE (with sub-links Volumes, Snapshots), NETWORK & SECURITY (with sub-links Security Groups, Elastic IPs, Placement Groups), and Load Balancers (which is highlighted with an orange circle). The main content area is titled "Create Load Balancer" and shows a table of load balancers. One row is selected, showing details: Load Balancer Name: YOURNAME-webserver-lb, DNS Name: YOURNAME-webserver-lb-1455672726.us-east-1.elb.amazonaws.com (A Record), Port Configuration: 80 (HTTP) forwarding to 80 (...), Availability Zones: us-east-1c, and 3 Instances. Below this, a detailed view for the selected load balancer shows tabs for Description, Instances, Health Check, Monitoring, Security, Listeners, and Tags. The Description tab is active, displaying the DNS name and a note about using CNAME records or Route 53.

Load balancer: YOURNAME-webserver-lb

DNS Name: YOURNAME-webserver-lb-1455672726.us-east-1.elb.amazonaws.com (A Record)

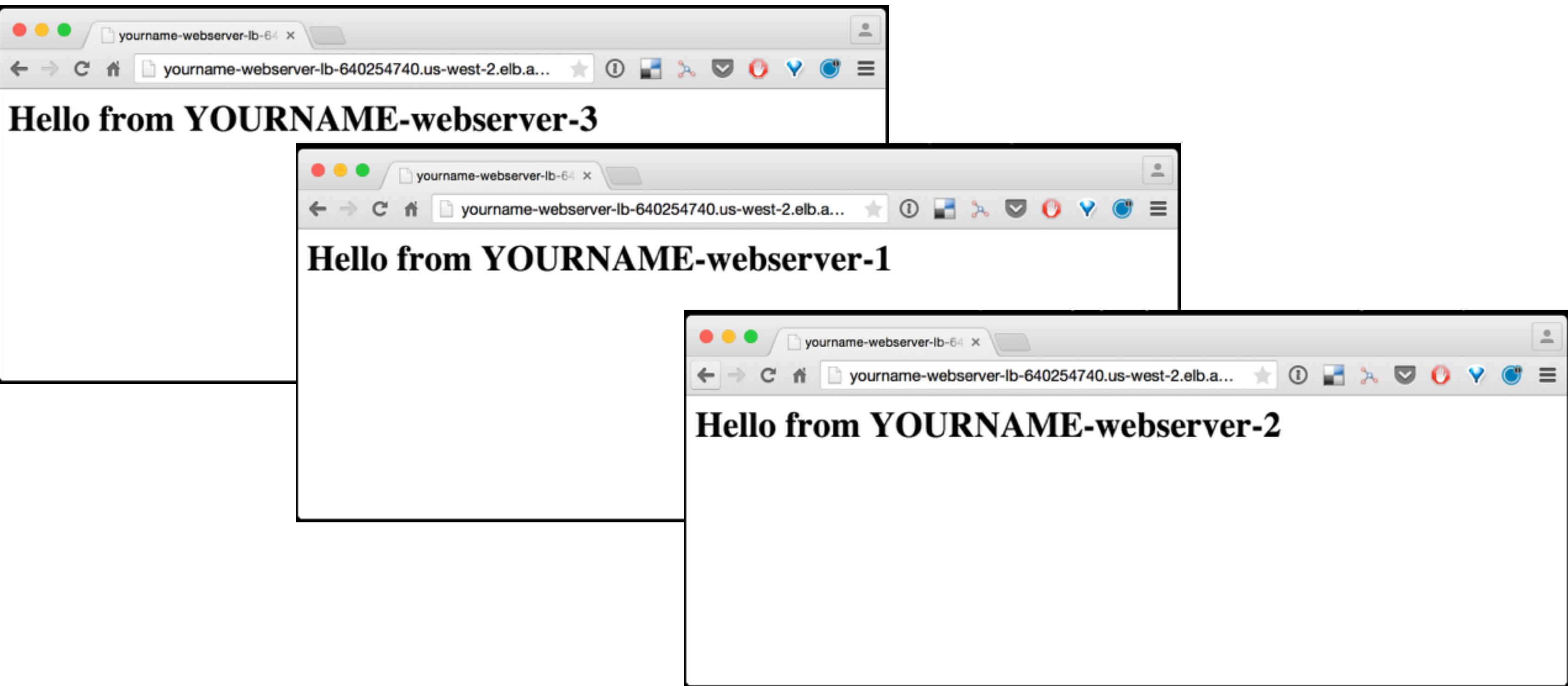
Note: Because the set of IP addresses associated with a LoadBalancer can change over time, you should never create an "A" record with any specific IP address. If you want to use a friendly DNS name for your load balancer instead of the name generated by the Elastic Load Balancing service, you should create a CNAME record for the LoadBalancer DNS name, or use Amazon Route 53 to create a hosted zone. For more information, see [Using](#)

Feedback English © 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Checkpoint

- Does everyone have their ELB hostname?

See your instances behind the ELB



Infrastructure Topology

- Our instances are up and associated with a load balancer
- Typically, our application layer might connect to a data layer or any number of intermediary services
- This same approach may be used to create other common layers our applications need in the correct order
- Sometimes nodes in the same layer may need to know about one another

Interconnectivity Data

- For a more useful example, let's see if we can set things up so the webservers all know about each other
- We can update our homepage so it displays the hostname of all other nodes that should be in this load balancer's network
- We can do this using Chef's **search** functionality

Lab 10 - Interconnect your web servers

- **The Problem:** Each webserver needs to display info about the other webservers in its group
- **Success Criteria:** We can see info about all of our webservers no matter which node we get behind our ELB

Chef Search

- Chef can dynamically search for data about your infrastructure
- In Production, we would likely want a Chef Server as a central source of truth
- For this bootcamp, we're using Chef Zero
- We can search using local-mode

List existing node objects

```
$ ls ~/chef-repo/nodes
```

```
ip-172-31-25-147.us-west-2.compute.internal.json  
YOURNAME-webserver-1.json  
YOURNAME-webserver-2.json  
YOURNAME-webserver-3.json
```

Search for all tagged nodes

```
$ knife search node "tags:my-webserver" --local-mode
```

```
3 items found

Node Name: YOURNAME-webserver-1
Environment: _default
FQDN: ip-172-31-18-60.us-west-2.compute.internal
IP: 52.10.8.115
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.5
Tags: my-webserver

Node Name: YOURNAME-webserver-2
Environment: _default
FQDN: ip-172-31-27-65.us-west-2.compute.internal
IP: 52.24.169.211
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.5
Tags: my-webserver

Node Name: YOURNAME-webserver-3
Environment: _default
FQDN: ip-172-31-31-39.ec2.internal
IP: 52.24.105.200
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.5
Tags: my-webserver
```

Search for all tagged nodes except one

```
$ knife search node "tags:my-webserver AND NOT name:YOURNAME-webserver-3" --local-mode
```

```
3 items found

Node Name: YOURNAME-webserver-1
Environment: _default
FQDN: ip-172-31-18-60.us-west-2.compute.internal
IP: 52.10.8.115
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.5
Tags: my-webserver

Node Name: YOURNAME-webserver-2
Environment: _default
FQDN: ip-172-31-27-65.us-west-2.compute.internal
IP: 52.24.169.211
Run List: recipe[webserver]
Roles:
Recipes: webserver, webserver::default
Platform: centos 6.5
Tags: my-webserver
```

How search works

- Chef Server indexes data about nodes, clients, roles, environments, and `data_bags` and makes it available for search
- Search is driven by SOLR
 - https://docs.chef.io/knife_search.html
- Search returns full objects that match the search query
- If the search works via knife, it will work in your recipe

Using search in a template

- So far, we've only used node attributes in a template
- The **template** resource allows us to pass in custom variables
 - Handy functionality for dynamic config files
 - We will pass a variable called "webservers" that is equal to the result of a search similar to what we just tried -- except each node filter itself.

Search for my-webservers



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/default.rb`

```
package "httpd"

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

template "/var/www/html/index.html" do
  source "index.html.erb"
  variables({
    :webservers => search(:node, "tags:my-webserver AND NOT name:#{node.name}")
  })
end

service "iptables" do
  action [ :disable, :stop ]
end
```

Render data from the machines variable



OPEN IN EDITOR: [~/chef-repo/cookbooks/apache/template/default/index.html.erb](#)

```
<html>
  <body>
    <h1>Hello from <%= node.name %></h1>
    <p>The other webservers on this load balancer are:
      <ul>
        <% @webservers.each do |server| %>
          <li><%= server.name %></li>
        <% end %>
      </ul>
    </p>
  </body>
</html>
```

Render data from the machines variable



OPEN IN EDITOR: `~/chef-repo/cookbooks/apache/template/default/index.html.erb`

```
<html>
  <body>
    <h1>Hello from <%= node.name %></h1>
    <p>The other webservers on this load balancer are:
      <ul>
        <% @webservers.each do |server| %>
          <li><%= server.name %></li>
        <% end %>
      </ul>
    </p>
  </body>
</html>
```

- If you are feeling hardcore, type it
- <http://bit.ly/lbmachines>

Run the webserver::provision recipe

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
[YOURNAME-webserver-1] (up to date)
  * service[httpd] action enable
[YOURNAME-webserver-3] (up to date)
  * template[/var/www/html/index.html] action create
    - update content in file /var/www/html/index.html from 216db0 to 5315e7
      --- /var/www/html/index.html      2015-05-10 03:13:09.086491462 +0000
      +++ /tmp/chef-rendered-template20150510-2587-1cin6bb 2015-05-10 04:00:40.131491462 +0000
      @@ -1,6 +1,12 @@
        <html>
          <body>
            - <h1>Hello from YOURNAME-webserver-3</h1>
            + <h1>Hello from YOURNAME-webserver-3</h1>
            + <p>The other webservers on this load balancer are:
            +   <ul>
            +     <li>YOURNAME-webserver-1</li>
            +     <li>YOURNAME-webserver-2</li>
            +   </ul>
            + </p>
          </body>
        </html>
[YOURNAME-webserver-2] (up to date)
  * service[httpd] action start
```

See your instances behind the ELB



Lab 10 Summary

- Obviously, this is a contrived example
- But this is the basis of functionality you can use to manage deployments of complex infrastructure topologies
- Used in tandem with other Chef constructs like environments, roles, and test-kitchen you can write robust, repeatable, and testable infrastructure

AWS Services with Chef Provisioning

- AWS Services you can currently manage
 - ElastiCache
 - SQS Queues
 - SNS Topics
 - ELBs
 - Security Groups
 - EC2 Instances
 - S3 Buckets
 - Autoscaling Groups
 - SSH Key pairs
 - Launch configs
 - VPCs
 - AMIs
- AWS driver is under active development
 - Expect more supported services

Upcoming AWS Services

- Currently, the following service additions have been triaged
 - IAM roles & policies
 - RDS instances
 - Route53 zones
 - S3 files
 - Full EC2 coverage (e.g. Placement Groups, Spot Instances)

How you can help

- Contribute to Chef Provisioning
<https://github.com/chef/chef-provisioning-aws/issues>
- Fill out our AWS Services survey
<http://bit.ly/chef-aws-feedback>
- We need AWS user feedback
 - How do you use Chef Provisioning?
 - What AWS services are used in your stacks?
 - How can we make it better?

How you can help

- Do you use AWS a lot? Do you like Chef? Are you opinionated about usage patterns?
- Help Chef shape its AWS UX
- Reach out directly to this guy

George Miranda

Global Partner Evangelist

gmiranda@chef.io

[@gmiranda23](https://twitter.com/gmiranda23)



Current State of AWS Service Support

- AWS support with Chef pre-dates things like libcloud
- The AWS cookbook focused on EC2 (instance only)
- The Chef Provisioning Driver now uses the AWS SDK RubyGem
- Available services may be covered by SDK v1 API or SDK v2 API
- Some things (e.g. RDS or Route53) may rely on external cookbooks (for now)

What could we do if...?

- We want to make custom rules for autoscaling instances?
- We want to manage my private cloud in a manner similar to CloudFormation?
- I want to share a replica of Production with project collaborators?

Summary

- How does Chef Provisioning manage infrastructure components?
- What benefits do you see for test and repair of infrastructure?
- How can Chef Provisioning support your custom cloud technology?

Summary

- What questions can I help you answer?

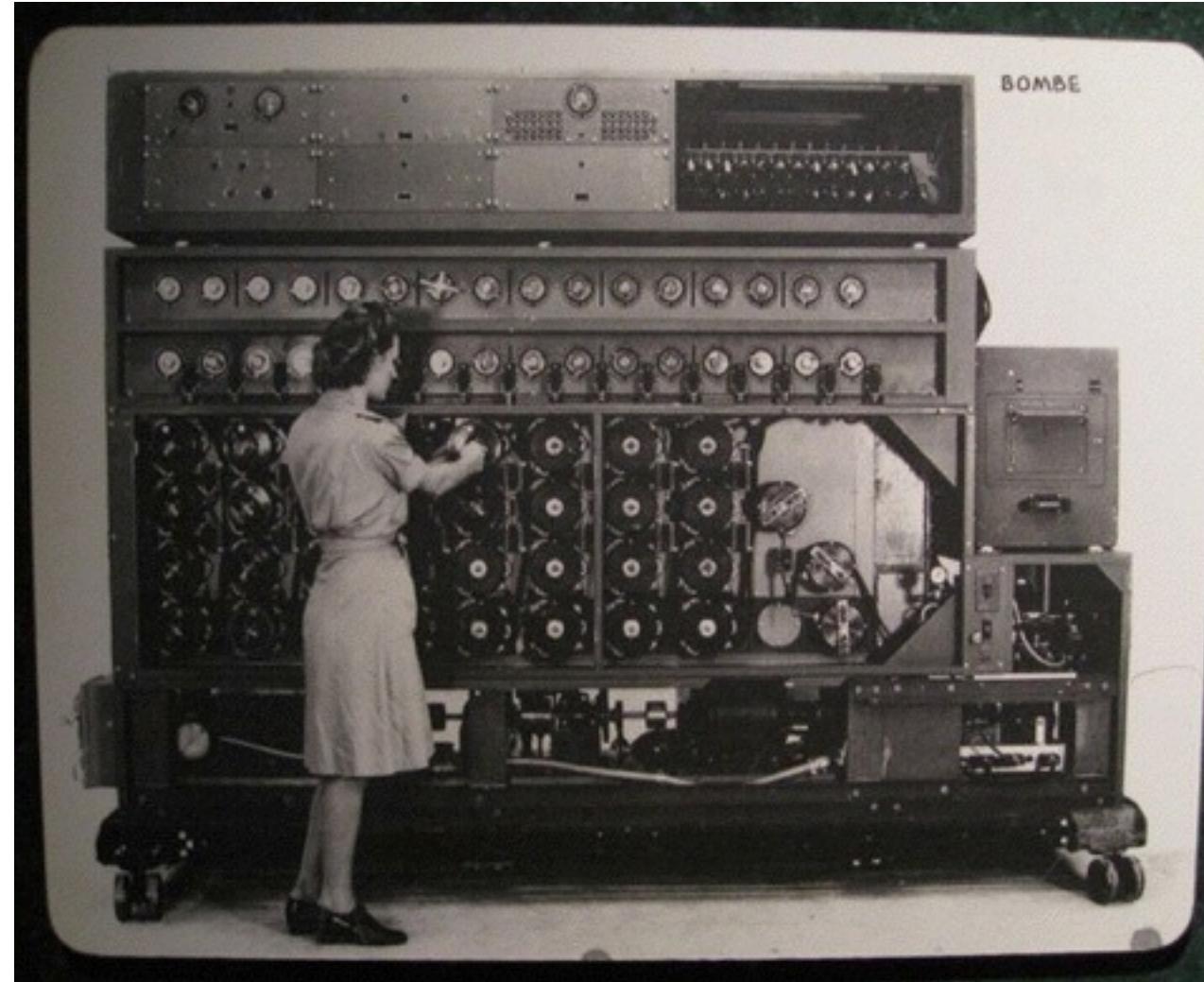
Healing your infrastructure

Recovering from Disaster

Lesson Objectives

- After completing the lesson, you will be able to
 - Describe what components are needed to recover from disasters using Chef
 - Rebuild the application infrastructure we've just created
 - Clean up your work from the AWS account we're sharing

Chef is Infrastructure as Code



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

- Remember this slide?
- Programmatically provision and configure servers
- Treat like any other code base
- Reconstruct business from **code repository, data backup, and compute resources**

Lab 11 - Simulate a disaster

- **The Problem:** We haven't practiced reconstructing our business using code we've written
- **Success Criteria:** We can recover the state of our application after a disaster

Lab 11 - Required steps

- Create a new recipe named **destroy** in the web servers cookbook to only manage webservers 2 to 3
- Destroy those two web servers
- Use the provision recipe to test and repair your infrastructure
- Observe how search results are impacted
- What questions can I help you answer?

Generate a new recipe

```
$ chef generate recipe cookbooks/webserver destroy
```

```
Compiling Cookbooks...
Recipe: code_generator::recipe
  * template[cookbooks/webserver/recipes/destroy.rb] action
    create
      - create new file cookbooks/webserver/recipes/destroy.rb
      - update content in file cookbooks/webserver/recipes/
        destroy.rb from none to e3b0c4
        (diff output suppressed by config)
      - restore selinux security context
```

Edit the destroy recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
name = "YOURNAME"

require 'chef/provisioning/aws_driver'
with_driver 'aws::us-east-1'

machine_batch do
  action :destroy
  machines 2.upto(3).map { |n| "#{name}-webserver-#{n}" }
end
```

Edit the destroy recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
name = "YOURNAME"

require 'chef/provisioning/aws_driver'
with_driver 'aws::us-east-1'

machine_batch do
  action :destroy
  machines 2.upto(3).map { |n| "#{name}-webserver-#{n}" }
end
```

- If you are feeling hardcore, type it
- http://bit.ly/machine_destroy

Destroy your web servers

```
$ chef-client --local-mode -r 'recipe[webserver::destroy]'
```

```
Starting Chef Client, version 12.3.0
resolving cookbooks for run list: [ "webserver::destroy" ]
Synchronizing Cookbooks:
  - webserver
Compiling Cookbooks...
Converging 1 resources
Recipe: webserver::destroy
* machine_batch[default] action destroy
  - [YOURNAME-webserver-3] delete instance aws_instance[YOURNAME-webserver-3] (i-84498b57) in VPC vpc-8b26daee in us-east-1
  - [YOURNAME-webserver-2] delete instance aws_instance[YOURNAME-webserver-2] (i-27488af4) in VPC vpc-8b26daee in us-east-1
  - [YOURNAME-webserver-3] waited 0/300s for i-84498b57 status to change to [:terminated]...
  - [YOURNAME-webserver-2] waited 0/300s for i-27488af4 status to change to [:terminated]...
  - [YOURNAME-webserver-3] waited 5/300s for i-84498b57 status to change to [:terminated]...
  - [YOURNAME-webserver-2] waited 5/300s for i-27488af4 status to change to [:terminated]...
  - [YOURNAME-webserver-3] waited 10/300s for i-84498b57 status to change to [:terminated]...
  - [YOURNAME-webserver-2] waited 10/300s for i-27488af4 status to change to [:terminated]...
  - [YOURNAME-webserver-3] waited until instance aws_instance[YOURNAME-webserver-3] (i-84498b57) is :terminated
  - [YOURNAME-webserver-3] delete node YOURNAME-webserver-3 at chefzero://localhost:8889
  - [YOURNAME-webserver-3] delete client YOURNAME-webserver-3 at clients
  - [YOURNAME-webserver-2] waited 15/300s for i-27488af4 status to change to [:terminated]...
  - [YOURNAME-webserver-2] waited until instance aws_instance[YOURNAME-webserver-2] (i-27488af4) is :terminated
  - [YOURNAME-webserver-2] delete node YOURNAME-webserver-2 at chefzero://localhost:8889
  - [YOURNAME-webserver-2] delete client YOURNAME-webserver-2 at clients

Running handlers:
Running handlers complete
Chef Client finished, 1/1 resources updated in 23.440084985 seconds
```

List all current nodes via knife

```
$ knife node list --local-mode
```

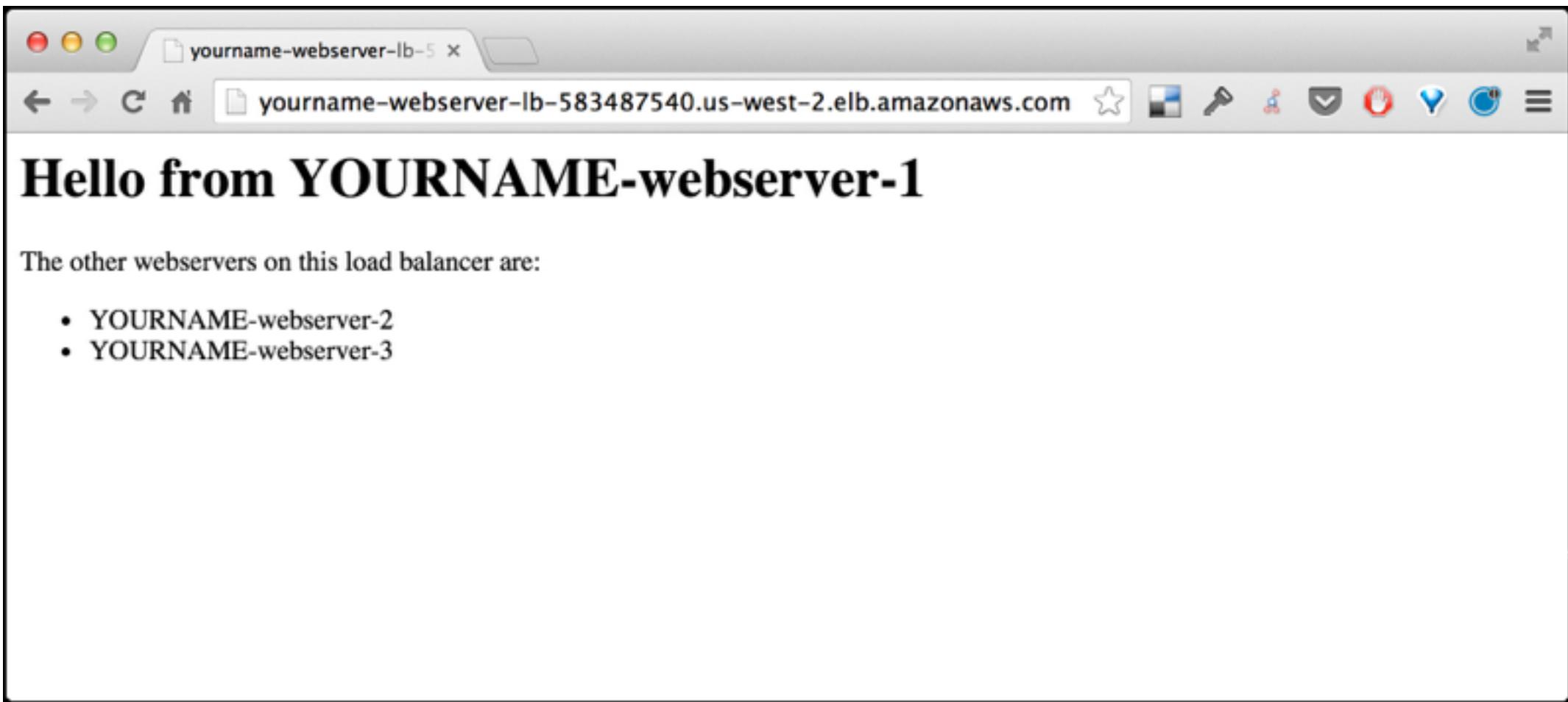
```
YOURNAME-webserver-1  
ip-172-31-25-147.us-west-2.compute.internal
```

Show all node objects

```
$ ls ~/chef-repo/nodes
```

```
ip-172-31-25-147.us-west-2.compute.internal.json  
YOURNAME-webserver-1.json
```

See your instances behind the ELB



WAT?!?!

- But we got rid of web servers 2 & 3
- Why do you think they are still there?

Stale configuration

- Webserver-1 has not converged since webserver-2 and webserver-3 were destroyed
- chef-client should typically run on a regular interval
- You must explicitly enable chef-client to run automatically (check out the chef-client cookbook)

Extra Credit Lab

- Set up a recipe that ensures the machine named **#{name}-webserver-1** will converge after your other machines are destroyed
- What questions can I help you answer?

Edit the destroy recipe



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
name = "YOURNAME"

require 'chef/provisioning/aws_driver'
with_driver 'aws::us-east-1'

machine_batch do
  action :destroy
  machines 2.upto(3).map { |n| "#{name}-webserver-#{n}" }
end

machine "#{name}-webserver-1" do
  converge true
end
```

Run chef-client

```
$ chef-client --local-mode -r 'recipe[webserver::destroy]'
```

```
* template[/var/www/html/index.html] action create
  - update content in file /var/www/html/index.html from 74e204 to a25407
    --- /var/www/html/index.html 2015-06-24 13:25:24.712880139 +0000
    +++ /tmp/chef-rendered-template20150625-19315-foxd3p 2015-06-25
      01:40:26.815880139 +0000
      @@ -3,8 +3,6 @@
      <h1>Hello from YOURNAME-webserver-1</h1>
      <p>The other webservers on this load balancer are:
        <ul>
        - <li>YOURNAME-webserver-2</li>
        - <li>YOURNAME-webserver-3</li>
        </ul>
        </p>
      </body>
      - restore selinux security context
* service[iptables] action disable (up to date)
* service[iptables] action stop (up to date)
```

Chef is explicit

- Chef is not a magic pony
- Chef will not automagically take corrective action
- Chef can help your infrastructure recover from failure but you must be explicit about exactly which actions you want Chef to take

Heal your infrastructure

- How can we recover from this failure?

Repair your infrastructure

```
$ chef-client --local-mode -r 'recipe[webserver::provision]'
```

```
- [YOURNAME-webserver-3] add normal.tags = ["my-webserver"]
- [YOURNAME-webserver-3] add normal.chef_provisioning = {"location"=>{"driver_url"=>"aws:aws",
"driver_version"=>"0.1.3", "allocated_at"=>"2015-01-12 05:06:26 UTC", "host_node"=>"http://localhost:8889/
nodes/", "image_id"=>"ami-b6bdde86", "instance_id"=>"i-ae6701a2"}}
- [YOURNAME-webserver-3] update run_list from [] to ["recipe[webserver]"] [2015-01-12T05:06:26+00:00] WARN:
Server YOURNAME-webserver-1 was created with SSH username and machine_options specifies username root. Using .
Please edit the node and change the chef_provisioning.location.ssh_username attribute if you want to change it.

- [YOURNAME-webserver-2] waiting for YOURNAME-webserver-2 (i-b76600bb on aws:aws) to be ready ...
- [YOURNAME-webserver-3] waiting for YOURNAME-webserver-3 (i-ae6701a2 on aws:aws) to be ready ...
- [YOURNAME-webserver-2] been waiting 0/120 -- sleeping 10 seconds for YOURNAME-webserver-2 (i-b76600bb on
aws:aws) to be ready ...
- [YOURNAME-webserver-3] been waiting 0/120 -- sleeping 10 seconds for YOURNAME-webserver-3 (i-ae6701a2 on
aws:aws) to be ready ...
[YOURNAME-webserver-1] [2015-01-12T05:06:32+00:00] WARN:
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
SSL validation of HTTPS requests is disabled. HTTPS connections are still
encrypted, but chef is not able to detect forged replies or man in the middle
attacks.
```

See your instances behind the ELB



Lab 12 - Clean Up After Yourself

- **The Problem:** I'm getting billed for your idle usage!
- **Success Criteria:** You have destroyed all your AWS instances

Required steps

- Modify the `webservers::destroy` recipe to destroy all three web servers, sec groups, and the ELB
- What questions can I help you answer?

Expand the loop



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
name = "YOURNAME"

require 'chef/provisioning/aws_driver'
with_driver 'aws::us-east-1'

machine_batch do
  action :destroy
  machines 1.upto(3).map { |n| "#{name}-webserver-#{n}" }
end
```

Remove the extra credit lab



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
name = "YOURNAME"

require 'chef/provisioning/aws_driver'
with_driver 'aws::us-east-1'

machine_batch do
  action :destroy
  machines 1.upto(3).map { |n| "#{name}-webserver-#{n}" }
end

machine "#{name}-webserver-1" do
  converge true
end
```

Destroy the Load Balancer



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
name = "YOURNAME"

require 'chef/provisioning/aws_driver'
with_driver 'aws::us-east-1'

machine_batch do
  action :destroy
  machines 1.upto(3).map { |n| "#{name}-webserver-#{n}" }
end

load_balancer "#{name}-webserver-lb" do
  action :destroy
end
```

Destroy the Security Groups



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
machine_batch do
  action :destroy
  machines 1.upto(3).map { |n| "#{name}-webserver-#{n}" }
end
```

```
load_balancer "#{name}-webserver-lb" do
  action :destroy
end
```

```
aws_security_group "#{name}-ssh" do
  action :destroy
end
```

```
aws_security_group "#{name}-http" do
  action :destroy
end
```

Destroy the Security Groups



OPEN IN EDITOR: `~/chef-repo/cookbooks/webserver/recipes/destroy.rb`

```
machine_batch do
  action :destroy
  machines 1.upto(3).map { |n| "#{name}-webserver-#{n}" }
end
```

```
load_balancer "#{name}-webserver-lb" do
  action :destroy
end
```

```
aws_security_group "#{name}-ssh" do
  action :destroy
end
```

```
aws_security_group "#{name}-http" do
  action :destroy
end
```

- If you are feeling hardcore, type it
- http://bit.ly/machine_destroy2

Clean up your work

```
$ chef-client --local-mode -r 'recipe[webserver]::destroy'
```

```
- [ YOURNAME-webserver-2] waited until instance aws_instance[YOURNAME-webserver-2] (i-7c18d0af) is :terminated
- [ YOURNAME-webserver-2] delete node YOURNAME-webserver-2 at chefzero://localhost:8889
- [ YOURNAME-webserver-2] delete client YOURNAME-webserver-2 at clients
- [ YOURNAME-webserver-3] waited 45/300s for i-eb19d138 status to change to [:terminated]...
- [ YOURNAME-webserver-3] waited until instance aws_instance[YOURNAME-webserver-3] (i-eb19d138) is :terminated
- [ YOURNAME-webserver-3] delete node YOURNAME-webserver-3 at chefzero://localhost:8889
- [ YOURNAME-webserver-3] delete client YOURNAME-webserver-3 at clients
- [ YOURNAME-webserver-1] waited 45/300s for i-7c36f4af status to change to [:terminated]...
- [ YOURNAME-webserver-1] waited until instance aws_instance[YOURNAME-webserver-1] (i-7c36f4af) is :terminated
- [ YOURNAME-webserver-1] delete node YOURNAME-webserver-1 at chefzero://localhost:8889
- [ YOURNAME-webserver-1] delete client YOURNAME-webserver-1 at clients
* load_balancer[YOURNAME-webserver-lb] action destroy
  - Deleting EC2 ELB chefzero://localhost:8889/data/load_balancer/YOURNAME-webserver-lb
  - delete data bag item YOURNAME-webserver-lb at chefzero://localhost:8889
* aws_security_group[YOURNAME-ssh] action destroy
  - delete aws_security_group[YOURNAME-ssh] in us-east-1
* aws_security_group[YOURNAME-http] action destroy
  - delete aws_security_group[YOURNAME-http] in us-east-1
```

Running handlers:

Running handlers complete

Chef Client finished, 4/4 resources updated in 54.678963177 seconds

Wrap Up

Recap & Next Steps

AWS PopUp Loft - A Taste of Chef v2.2.0

Course Objectives

- Upon completion of this course you will be able to
 - Automate common infrastructure tasks with Chef
 - Describe some of Chef's tools
 - Apply Chef's primitives to solve your problems
 - Use Chef to manage EC2 instances
 - Use Chef to manage an entire application topology in AWS
 - Know where to go for more info

Tool Survey

- chef-apply
- chef
- chef-client in local mode
- knife in local mode
- Chef Provisioning
- Chef Provisioning AWS Driver

Vocabulary

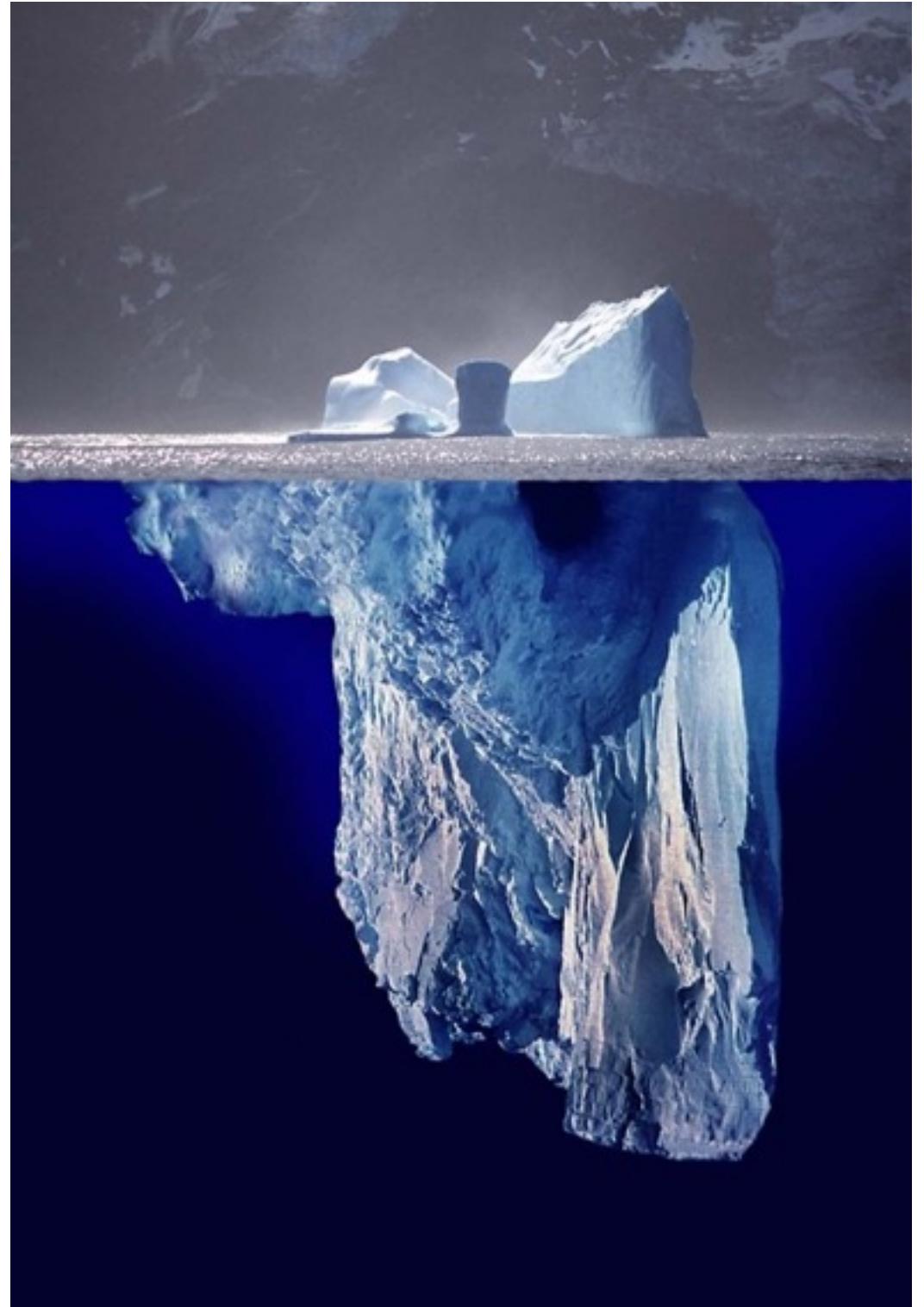
- Resources
- Recipes
- Cookbooks
- Nodes

Resources

- Package
- File
- Template
- Service
- Machine
- Load Balancer

But wait...

- There is so much more!
- How much time do we have left, I could go on for days!



Continued Learning

- The LearnChef Site
 - Guided Tutorials
 - Chef Fundamentals intro
<http://learnchef.com>
- How-To's, Conference Talks, Webinars, more
<http://youtube.com/user/getchef>
- Attend a Chef Fundamentals Class

Further Resources

- <http://chef.io>
- <http://docs.chef.io>
- <http://supermarket.chef.io>
- <http://lists.opscode.com>
- <https://www.chef.io/training>
- irc.freenode.net #chef
- Twitter @chef #getchef, @learnchef #learnchef

Chef Fundamentals (Online)

- 3 Day Chef Fundamentals (Windows/Linux)
 - July 6-8, 2015, 11am-5:30pm Eastern
- 3 Day Chef Fundamentals (Linux)
 - July 8-10, 2015, 11am-5:30pm Eastern
- 3 Day Chef Fundamentals (Windows/Linux)
 - July 14-16, 2015, 11am-5:30pm Eastern

25% OFF, use discount code: LOFT

Food Fight Show

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



Please give us your feedback!

- 3 Question Survey
 - <http://bit.ly/ratethisclass>
- AWS Services survey
<http://bit.ly/chef-aws-feedback>
- Slide Deck
 - <http://bit.ly/chefloft21>



Time to hack





Questions?

Thank you!

CHEF

@gmiranda23

gmiranda@chef.io