Chef Fundamentals

training@getchef.com
Copyright (C) 2014 Chef Software, Inc.

Nathen Harvey

- Community Director
- Co-host of the Food Fight Show Podcast

- @nathenharvey

Webinar Objectives and Style

CHEF™
GETCHEF.COM

Multi-week Webinar Series

- After completing of this webinar series you will be able to
  - Automate common infrastructure tasks with Chef
  - Describe Chef's architecture
  - Describe Chef's various tools
  - Apply Chef's primitives to solve your problems

How to learn Chef

- You bring the domain expertise about your business and infrastructure
- Chef provides a framework for automating your infrastructure
- Our job is to work together to teach you how to model and automate your infrastructure with Chef

Chef is a Language

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

Agenda

CHEF™
GETCHEF.COM

Topics

- Overview of Chef
- Workstation Setup
- Node Setup
- Chef Resources and Recipes
- Introducing the Node object
- Setting Node attributes
- Roles
- Community Cookbooks

Overview of Chef

CHEF™
GETCHEF.COM

Lesson Objectives

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

Complexity

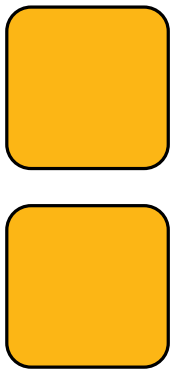
Items of Manipulation (Resources)

- Networking
- Files
- Directories
- Symlinks
- Mounts
- Registry Keys
- Powershell Scripts
- Users
- Groups
- Packages
- Services
- Filesystems

A tale of growth...

Application

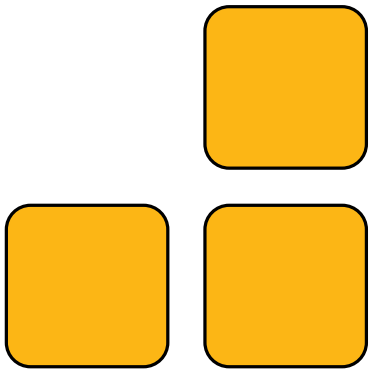
Add a database



Application

Application Database

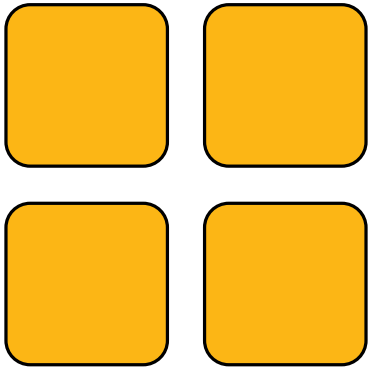
Make database redundant



Application

App Databases

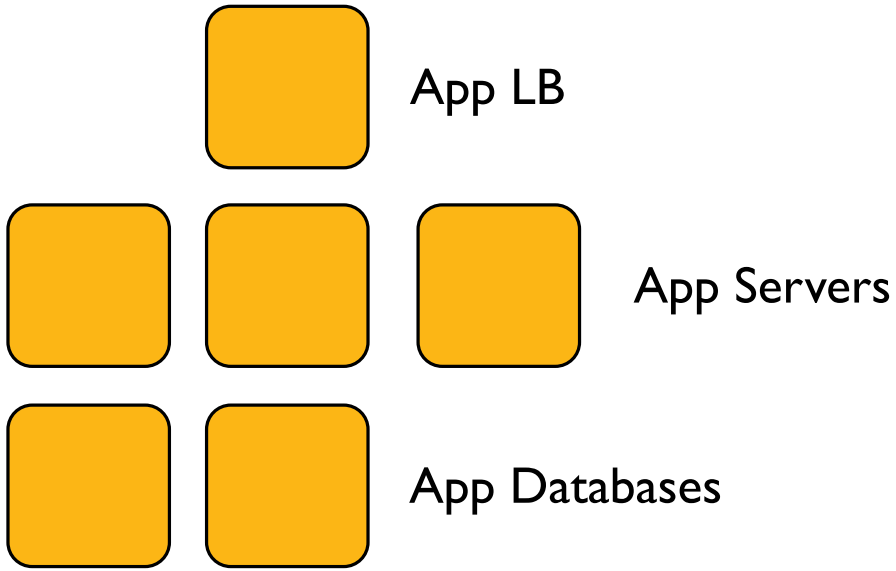
Application server redundancy



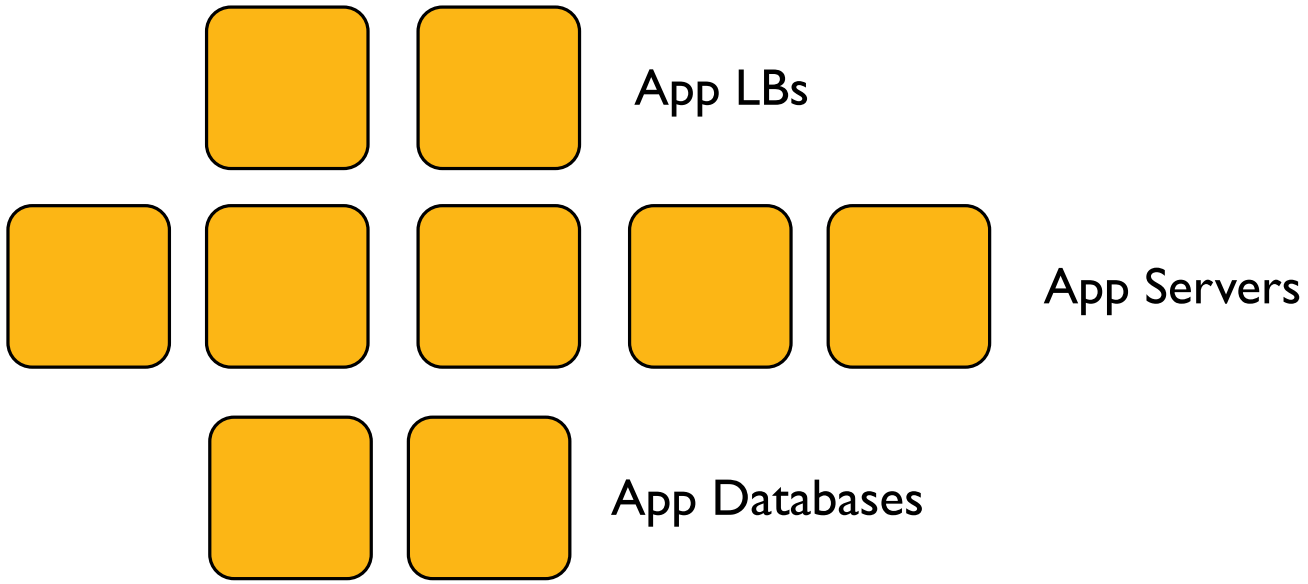
App Servers

App Databases

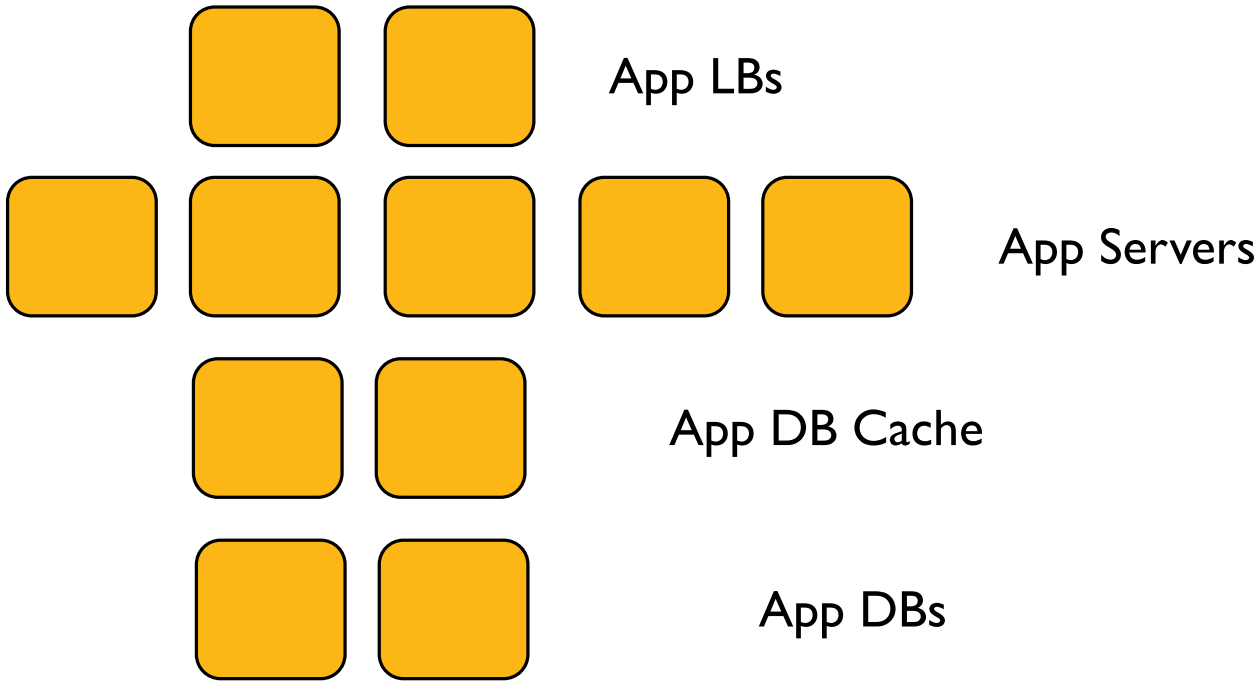
Add a load balancer



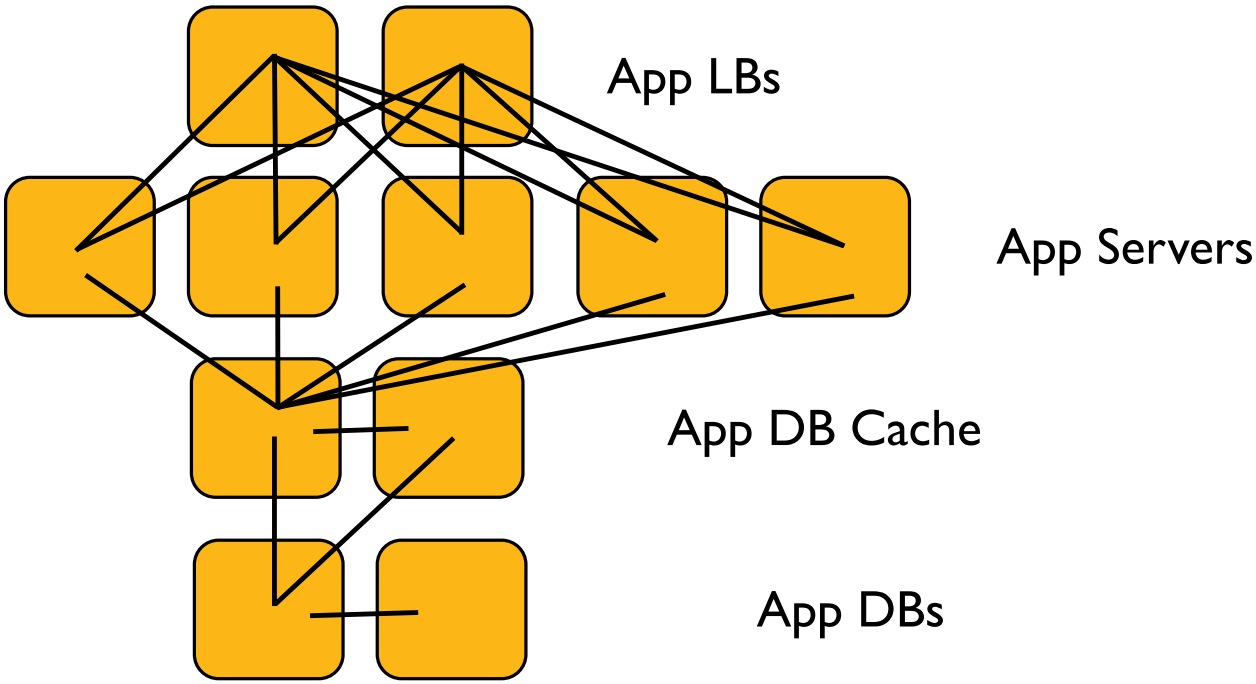
Webscale!



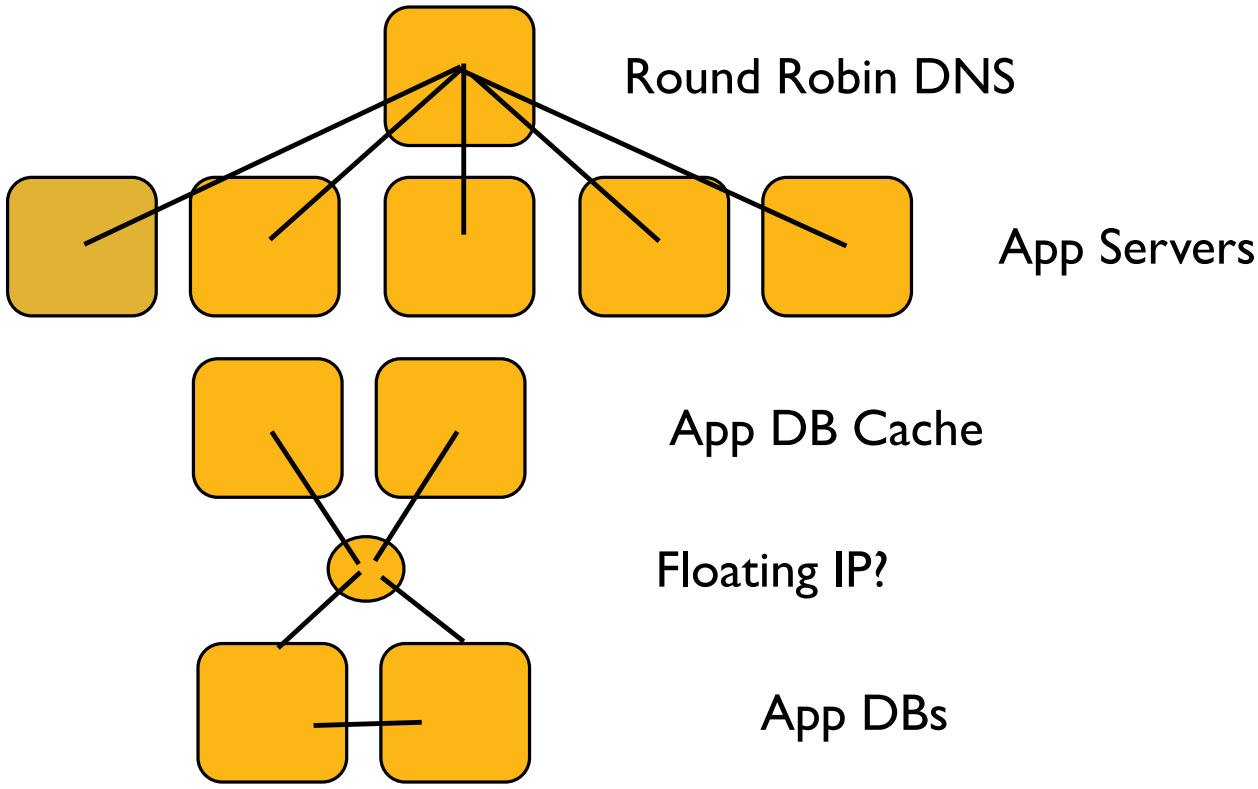
Now we need a caching layer



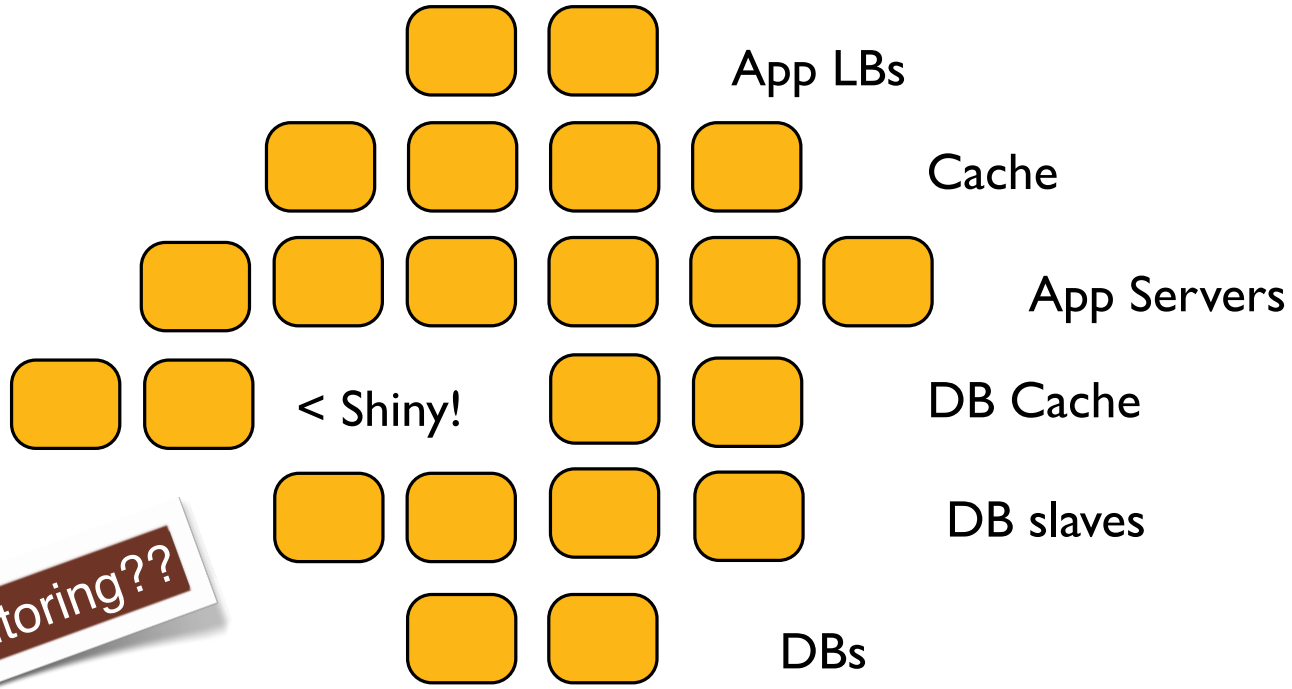
Infrastructure has a Topology



Your Infrastructure is a Snowflake

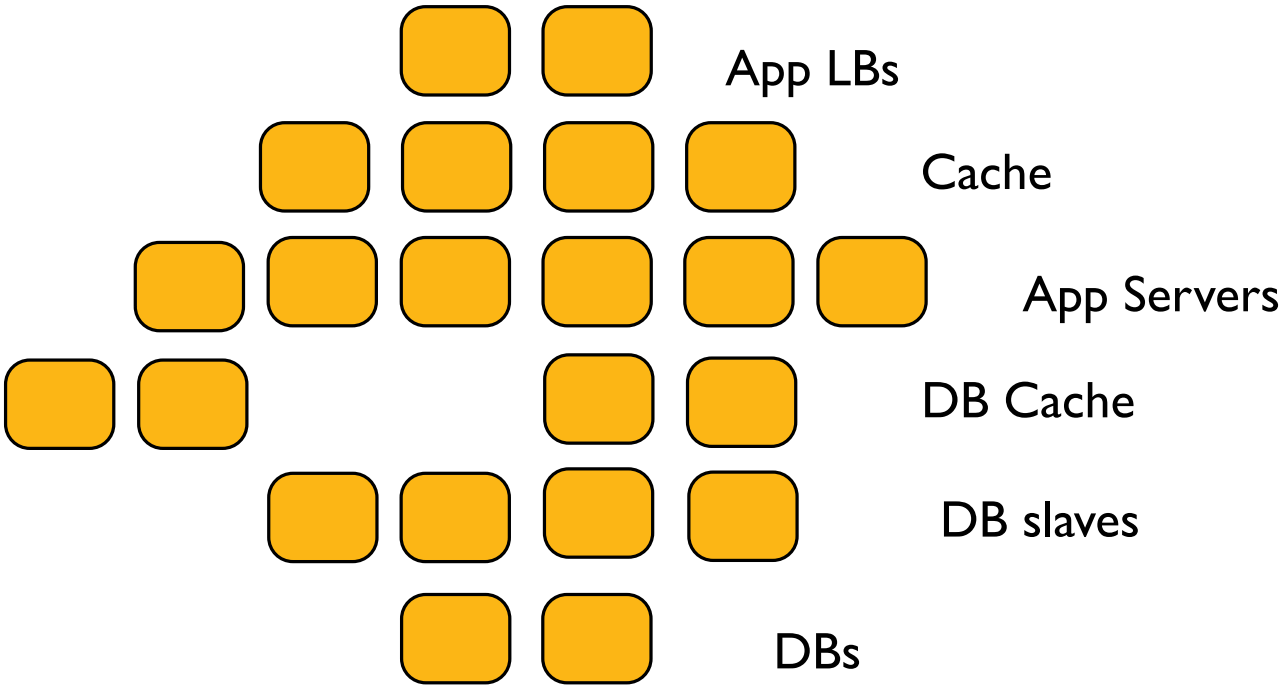


Complexity Increases Quickly

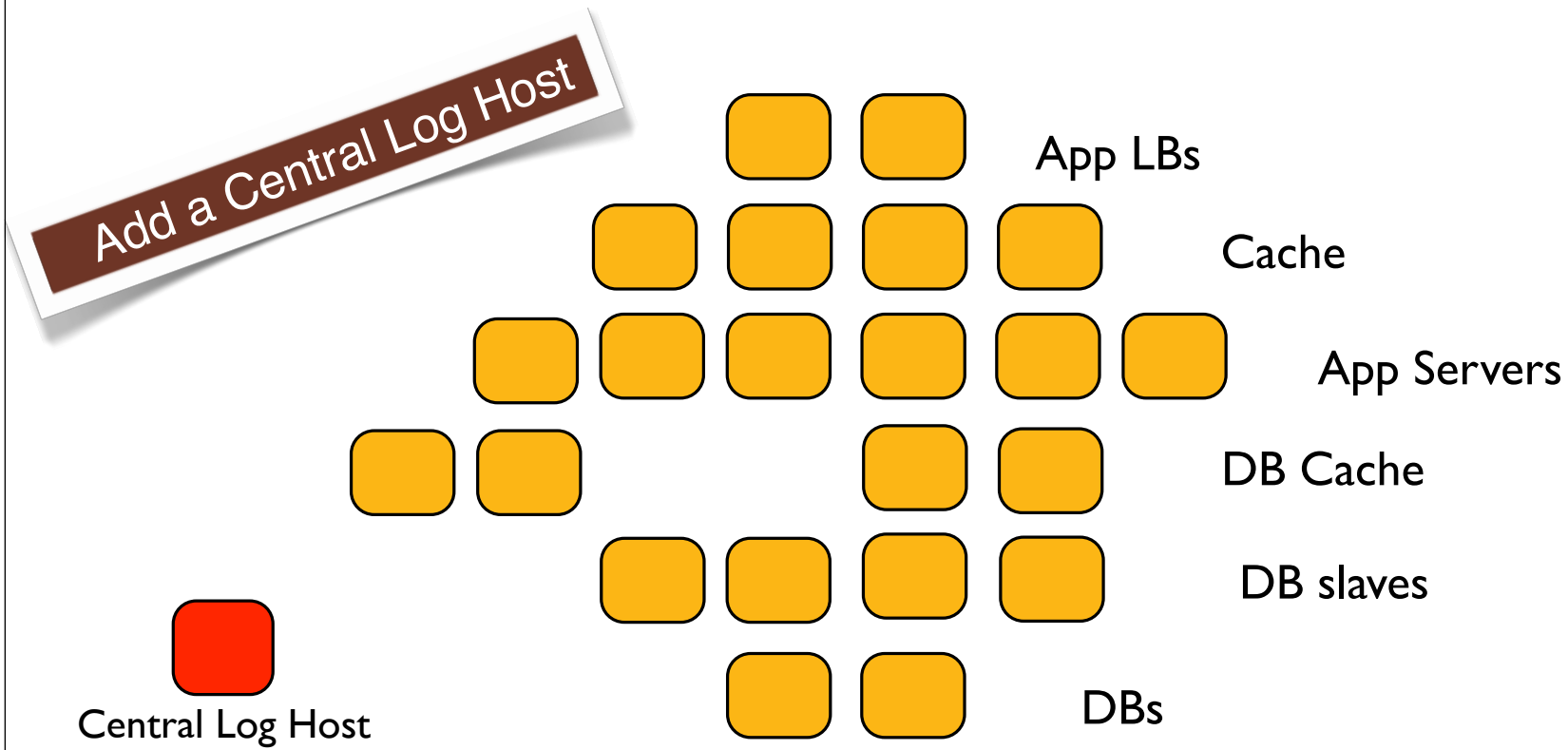


Are we monitoring??

...and change happens!



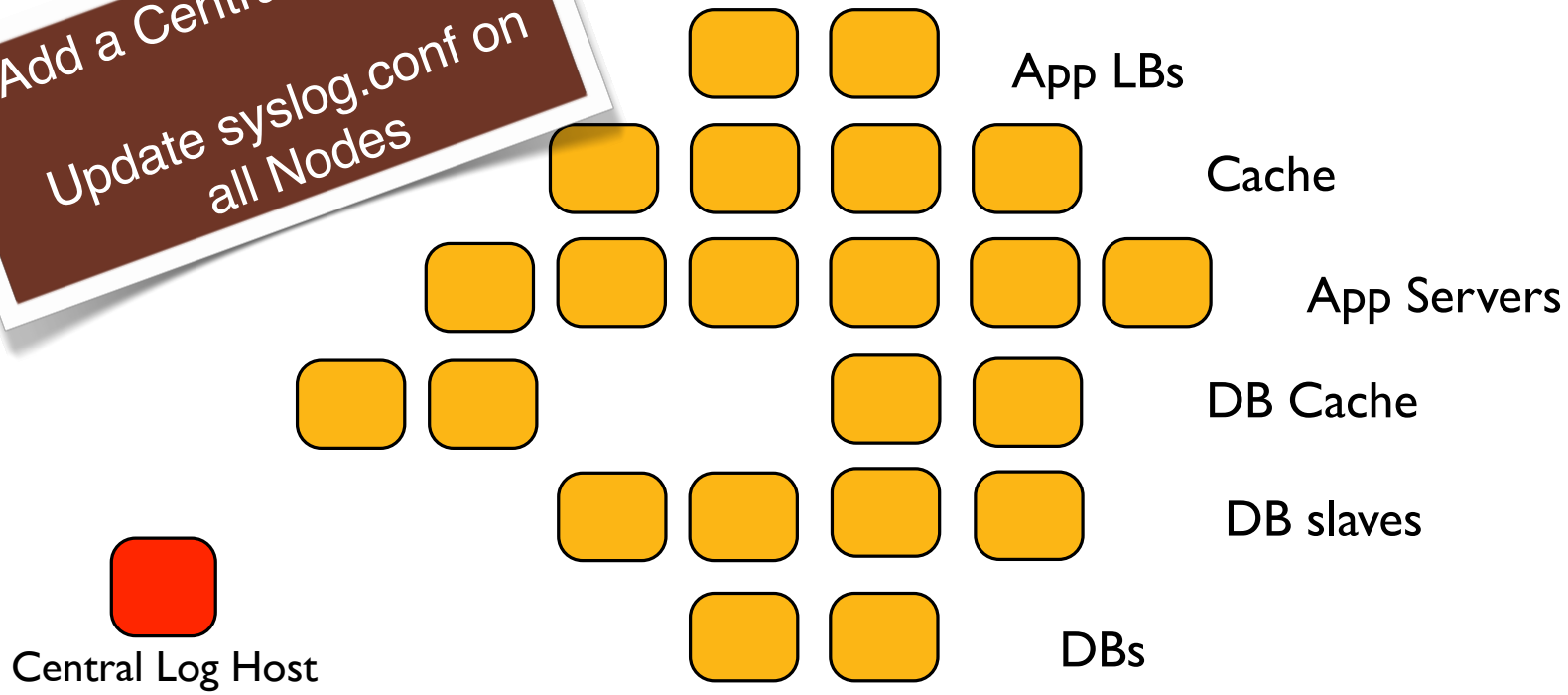
...and change happens!



...and change happens!

Add a Central Log Host

Update syslog.conf on all Nodes



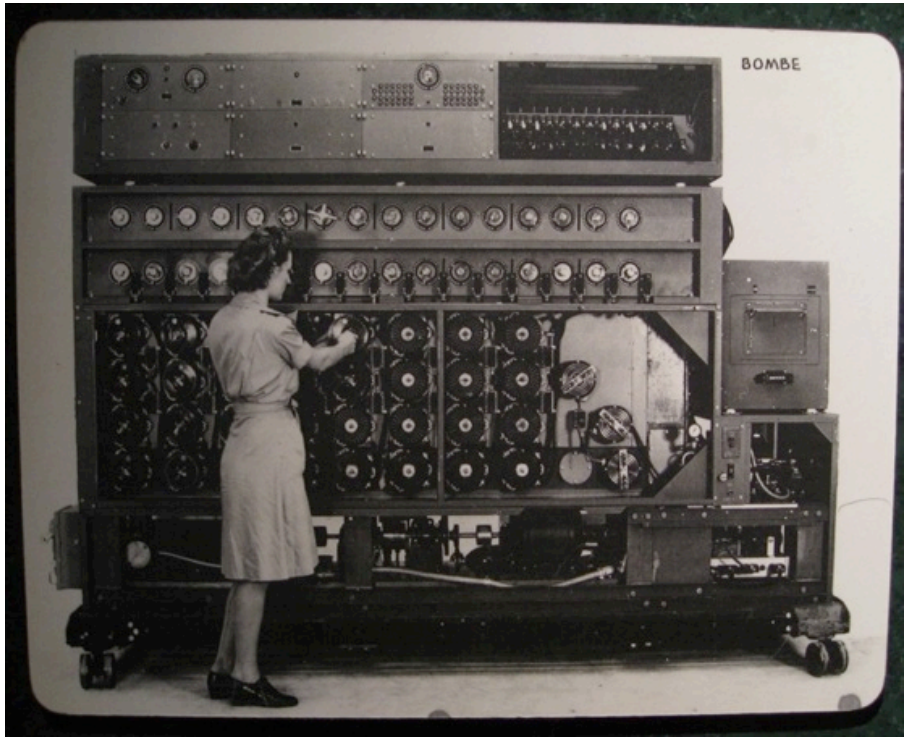
Chef Solves This Problem

CHEF™

GETCHEF.COM

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

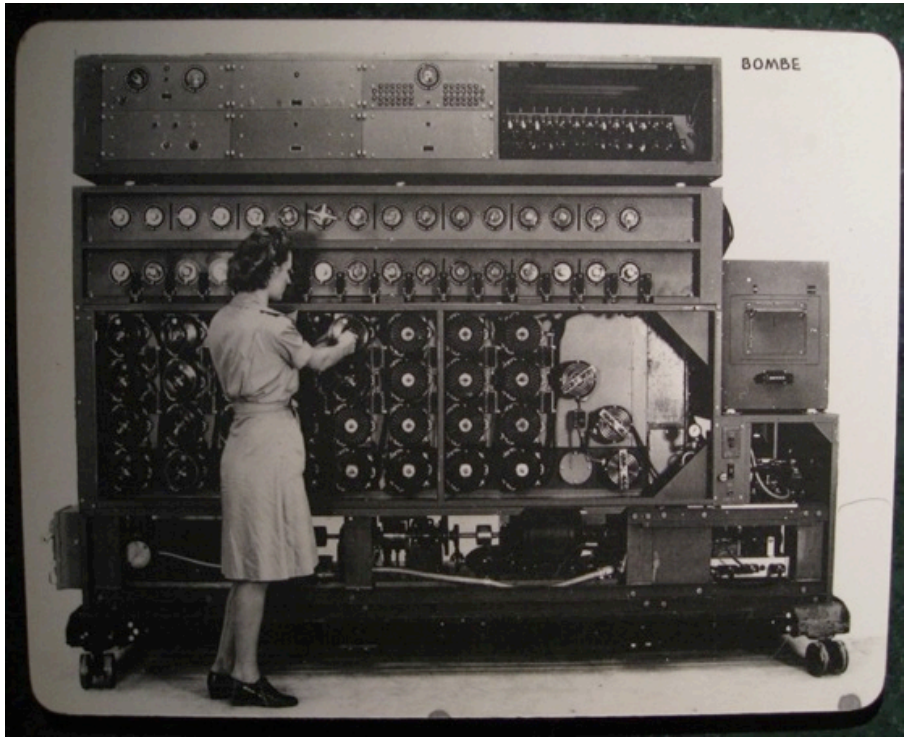
Chef is Infrastructure as Code



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

- Programmatically provision and configure components

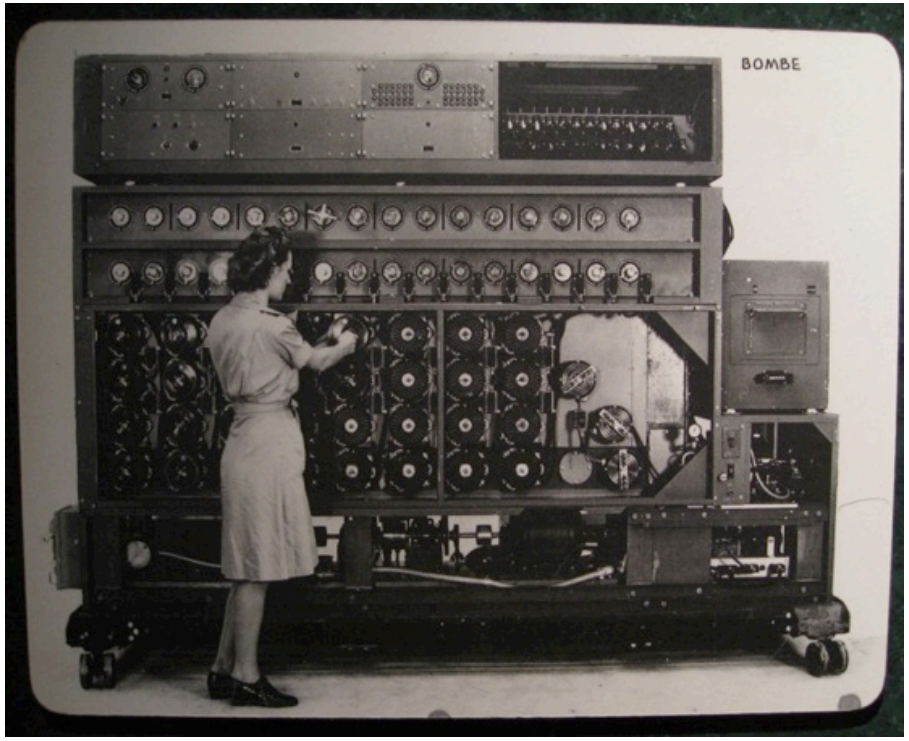
Chef is Infrastructure as Code



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

- Treat like any other code base

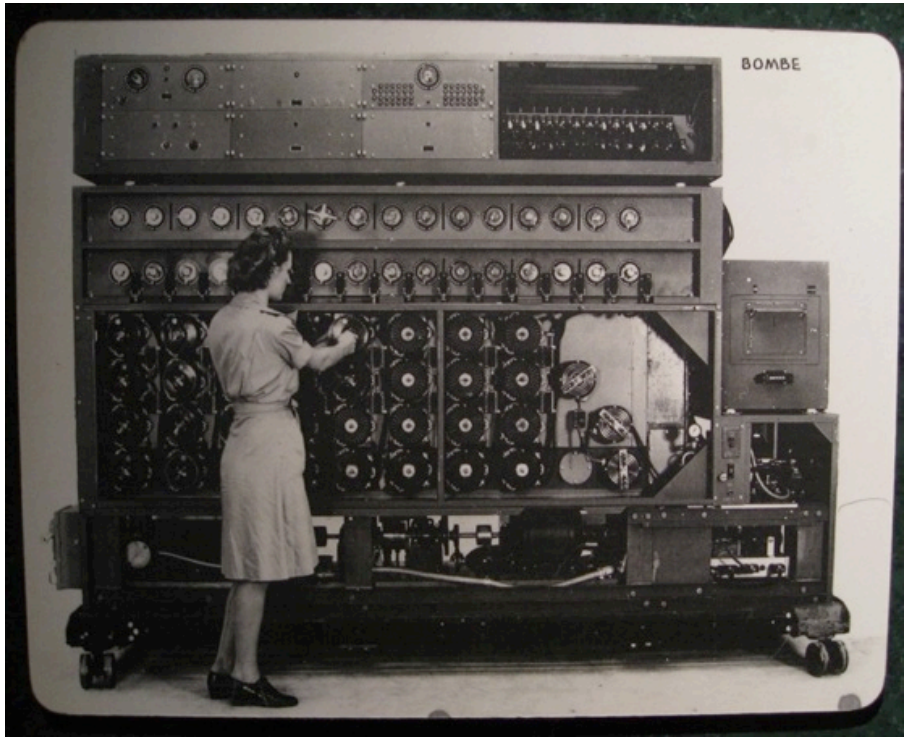
Chef is Infrastructure as Code



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

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

Chef is Infrastructure as Code



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

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

Configuration Code

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

Declarative Interface to Resources

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

Managing Complexity

- Resources
- Recipes
- Nodes
- Search

Resources

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

Resources in Recipes

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

Recipes

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

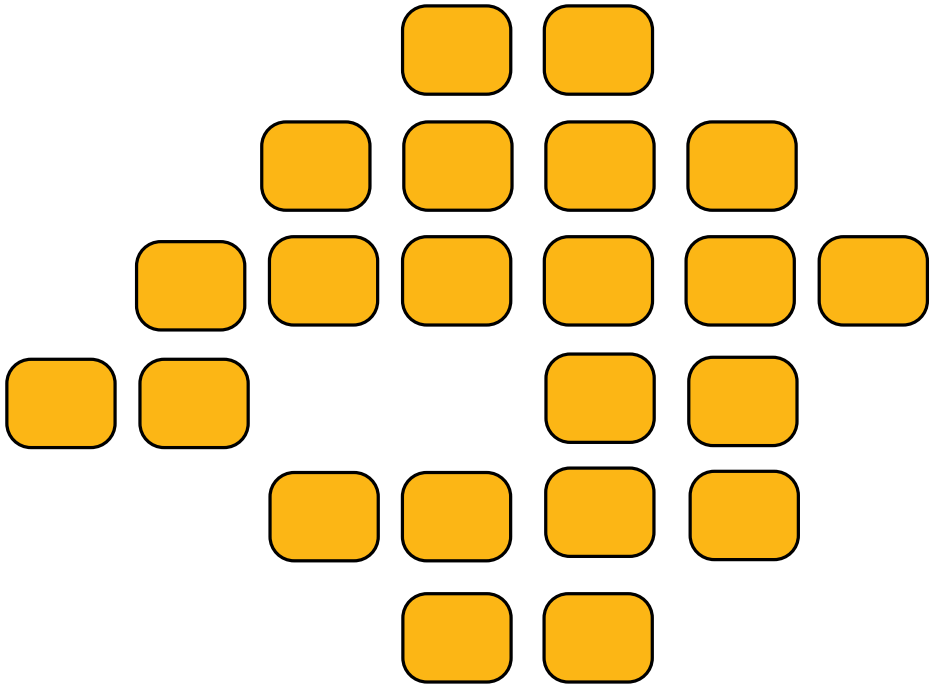
Example Recipe

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

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

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

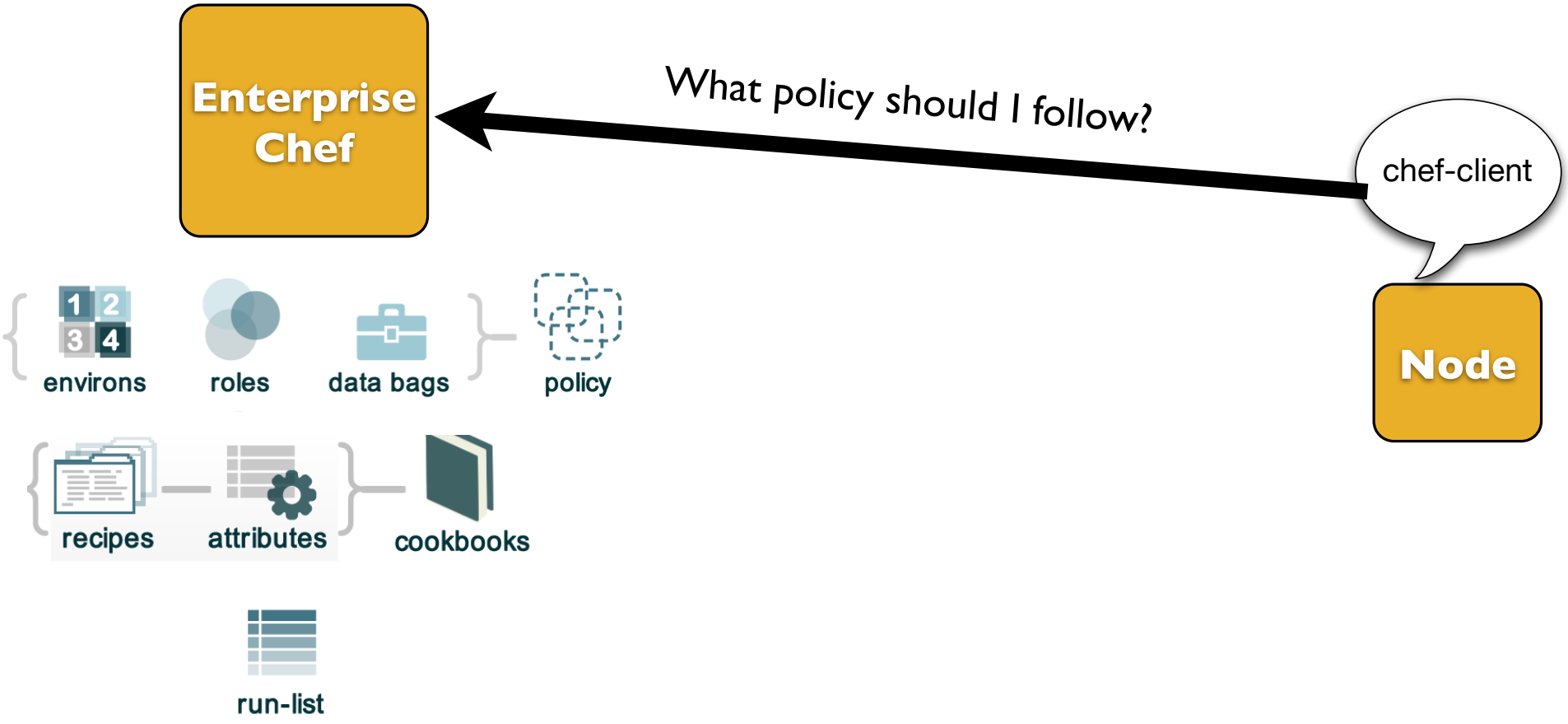
Nodes



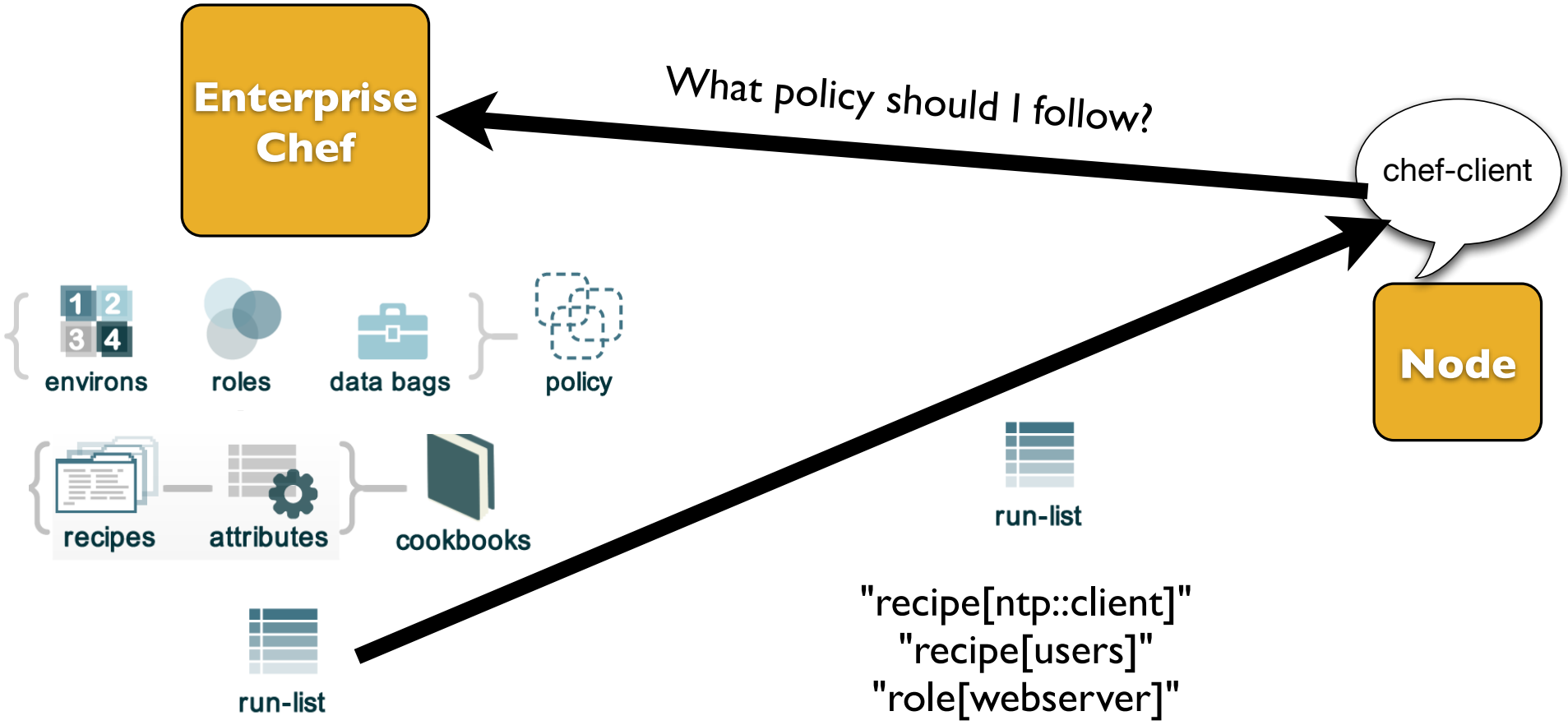
Nodes Adhere to Policy

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

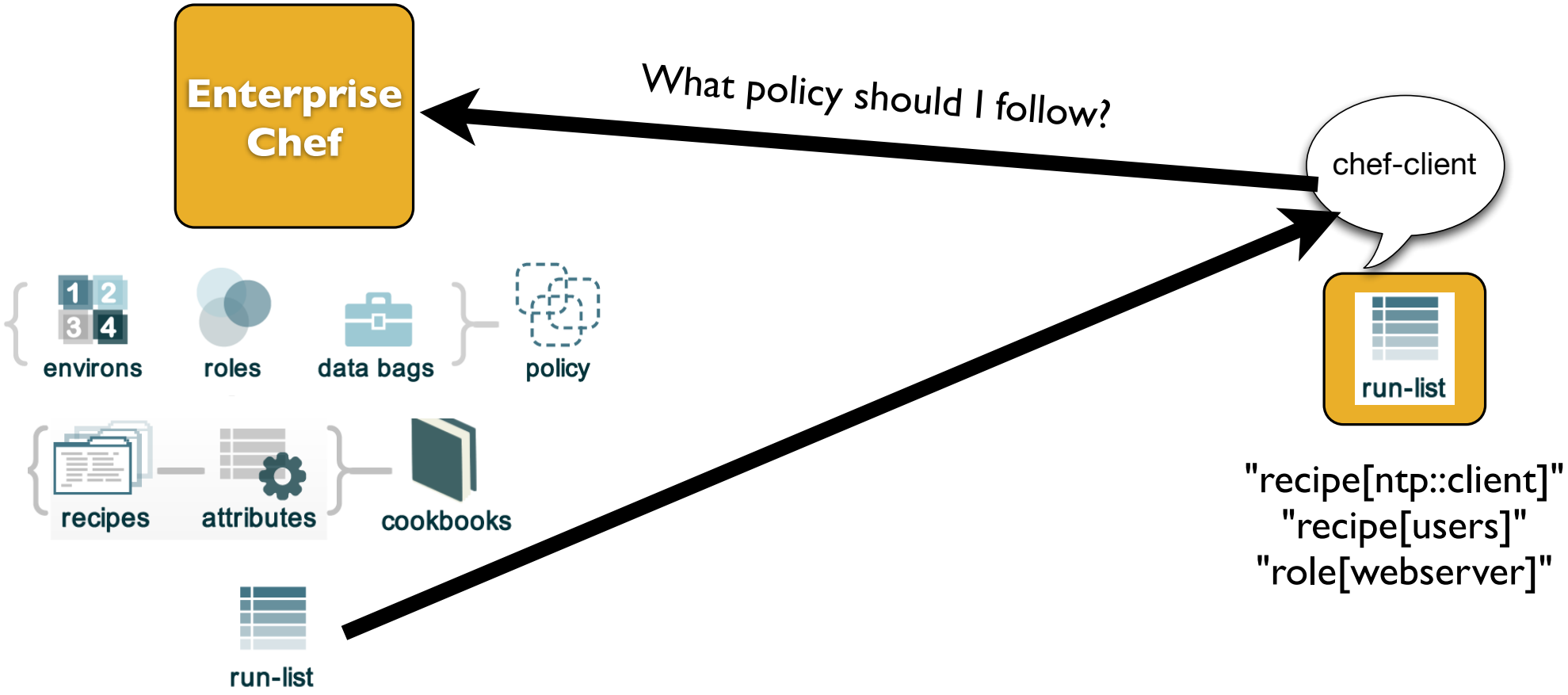
Run List


Enterprise Chef
What policy should I follow?
chef-client
Node
environs
roles
data bags
policy
recipes
attributes
cookbooks
run-list

Run List


Enterprise Chef
What policy should I follow?
chef-client
Node
environs
roles
data bags
policy
recipes
attributes
cookbooks
run-list
run-list
"recipe[ntp::client]"
"recipe[users]"
"role[webserver]"

Run List


Enterprise Chef
What policy should I follow?
chef-client
environs
roles
data bags
policy
run-list
recipes
attributes
cookbooks
"recipe[ntp::client]"
"recipe[users]"
"role[webserver]"
run-list

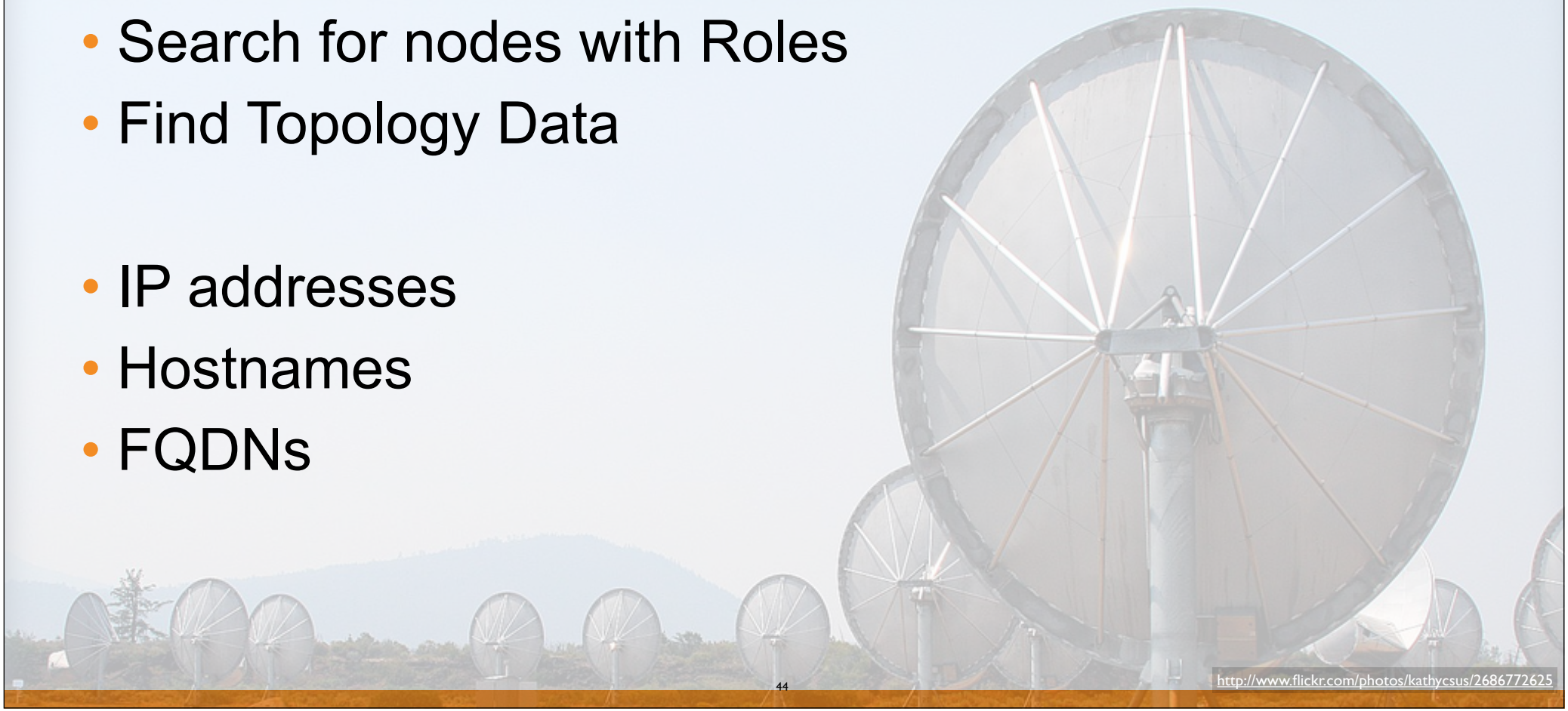
Run List Specifies Policy

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

Search

- Search for nodes with Roles
- Find Topology Data

- IP addresses
- Hostnames
- FQDNs

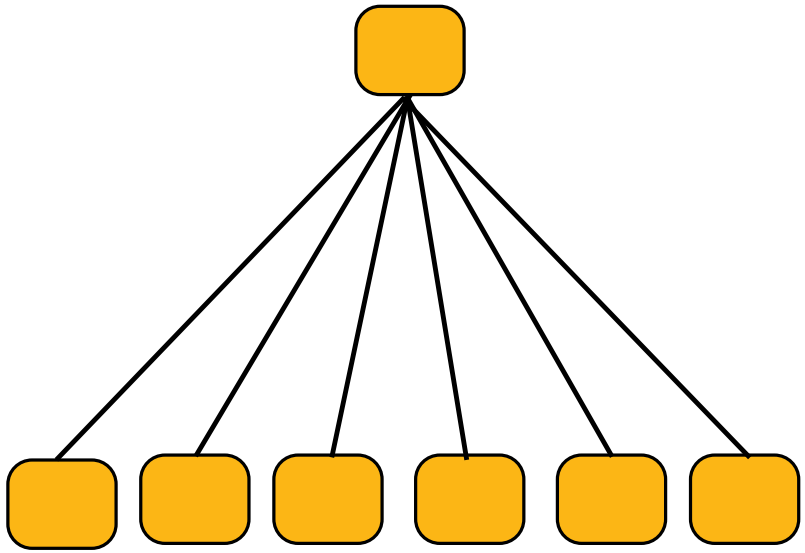


http://www.flickr.com/photos/kathycsus/2686772625

Search for Nodes

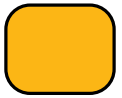
```
pool_members = search("node","role:webserver")

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

HAProxy Configuration

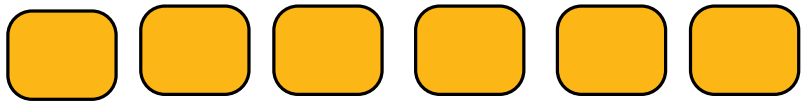


HAProxy Load Balancer

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



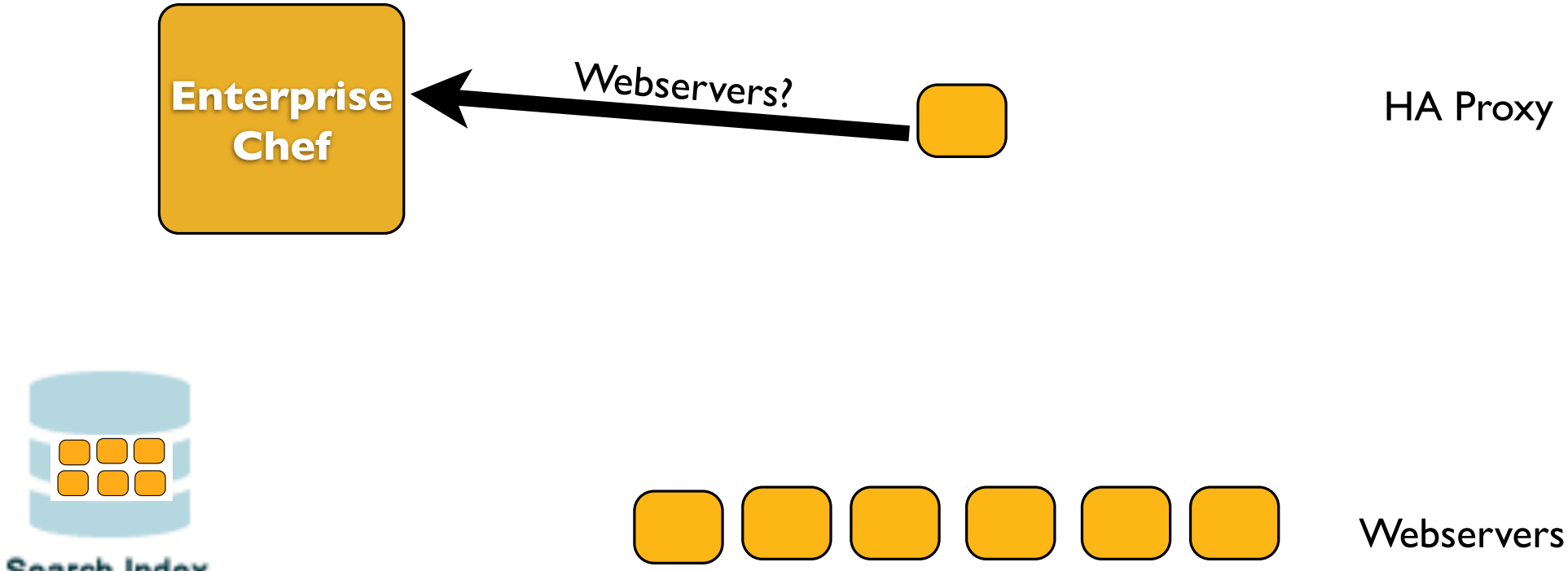
HA Proxy



Webservers

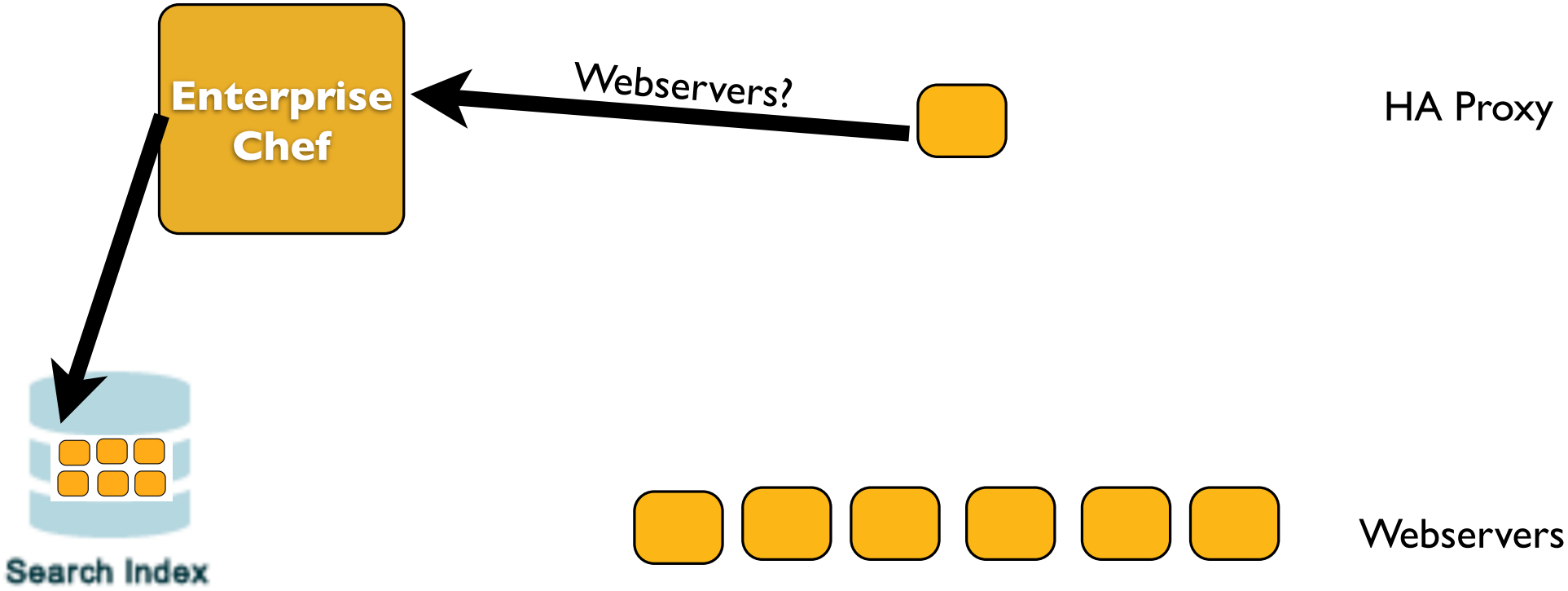
HAProxy Load Balancer

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



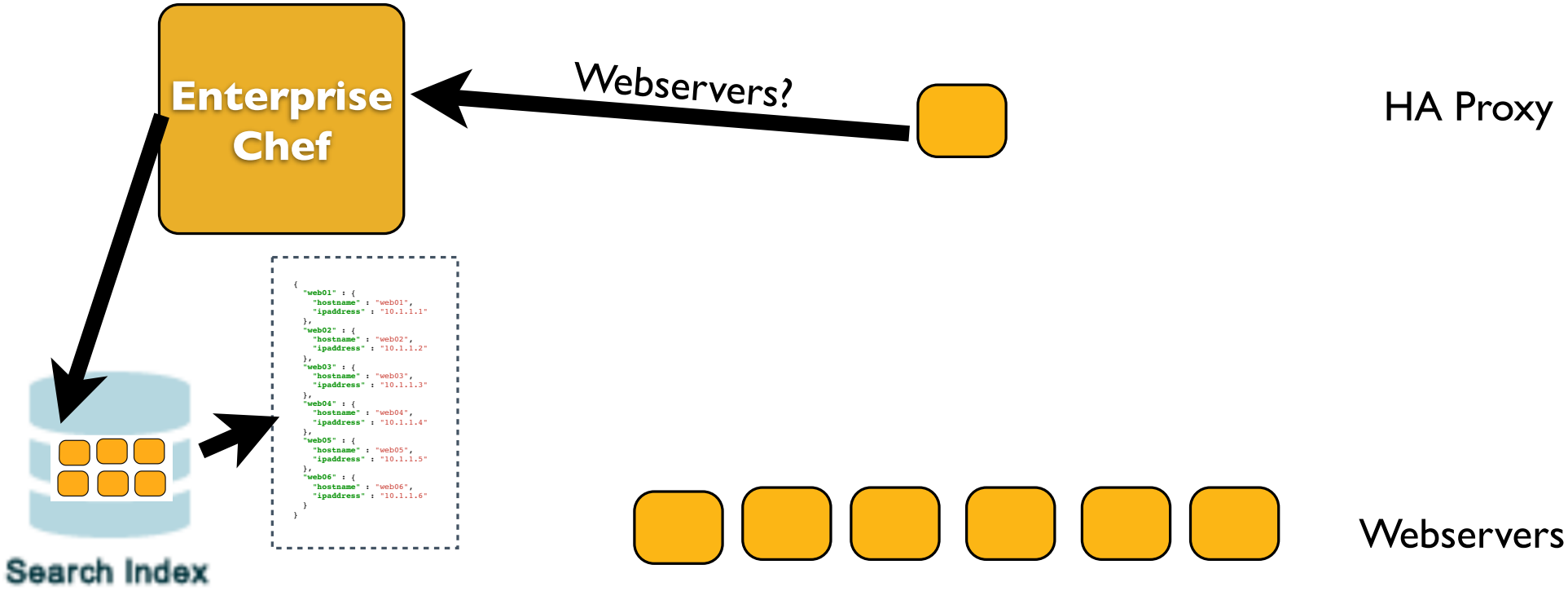
HAProxy Load Balancer

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



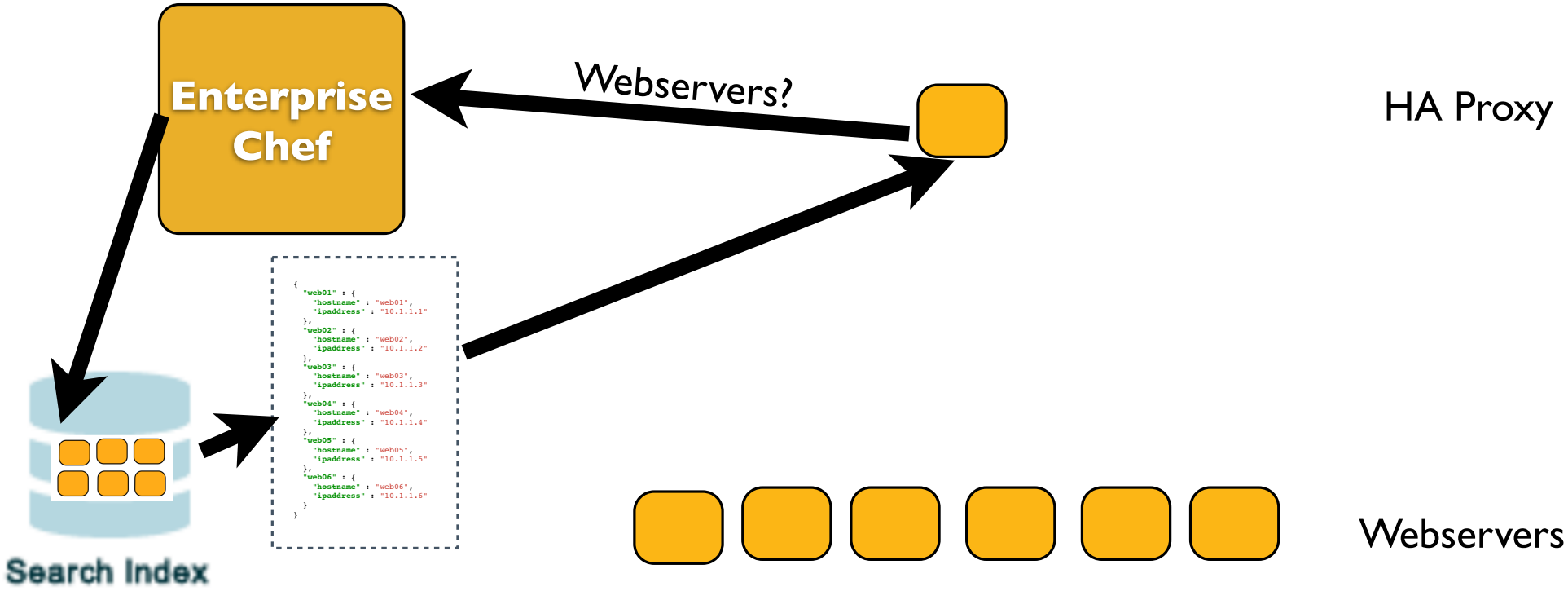
HAProxy Load Balancer

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



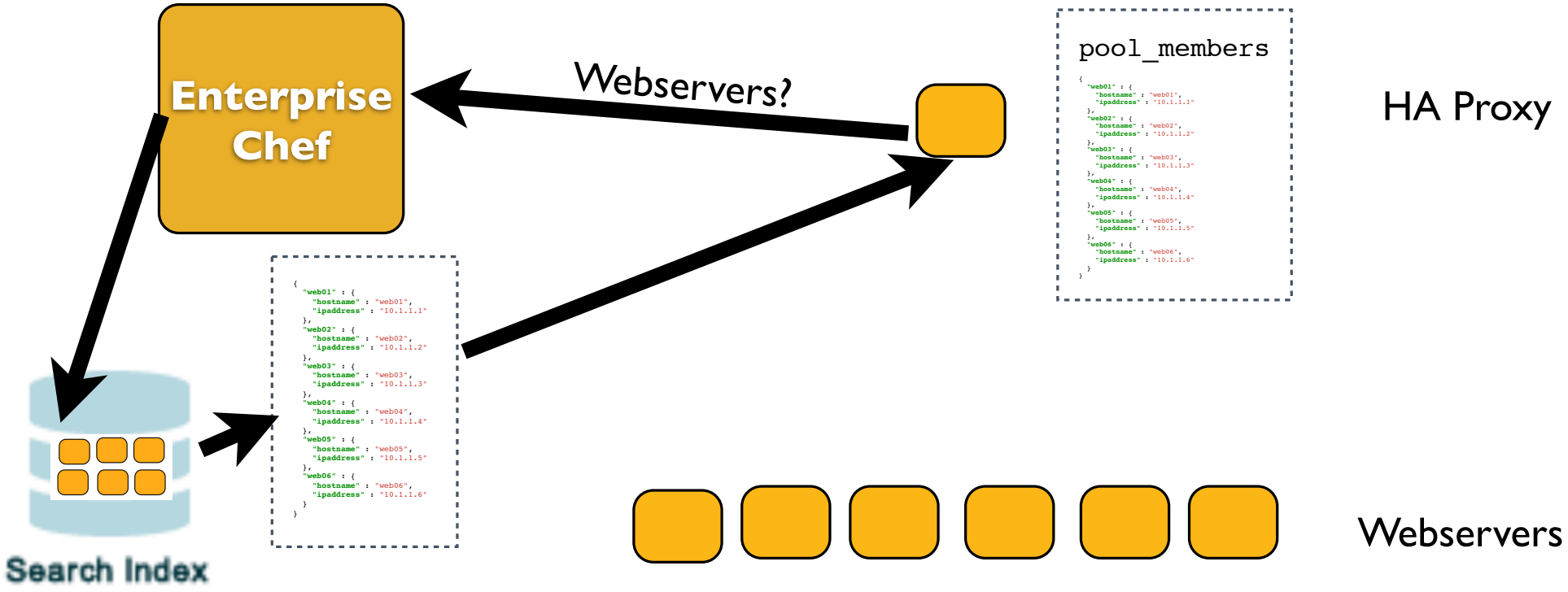
HAProxy Load Balancer

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



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 -%>
```

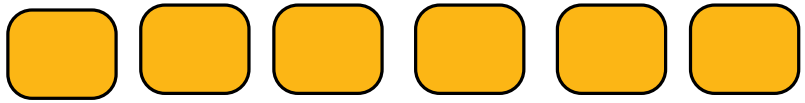
HAProxy Configuration

```
<% @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"
  },
  "web05" : {
    "hostname" : "web05",
    "ipaddress" : "10.1.1.5"
  },
  "web06" : {
    "hostname" : "web06",
    "ipaddress" : "10.1.1.6"
  }
}
```


HA Proxy



Webservers

HAProxy Configuration

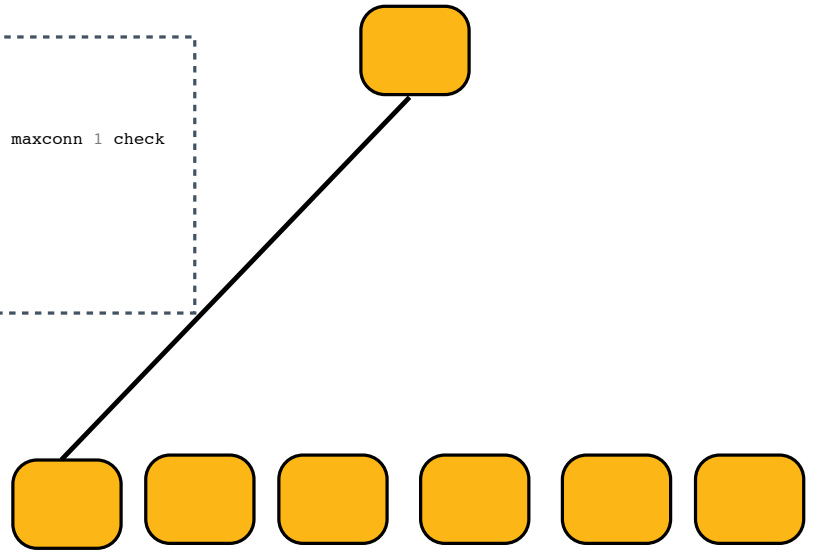
```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>:> weight 1 maxconn 1 check
<% end -%>
```

pool_members


```
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  },
  "web05" : {
    "hostname" : "web05",
    "ipaddress" : "10.1.1.5"
  },
  "web06" : {
    "hostname" : "web06",
    "ipaddress" : "10.1.1.6"
  }
}
```


haproxy.cfg

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

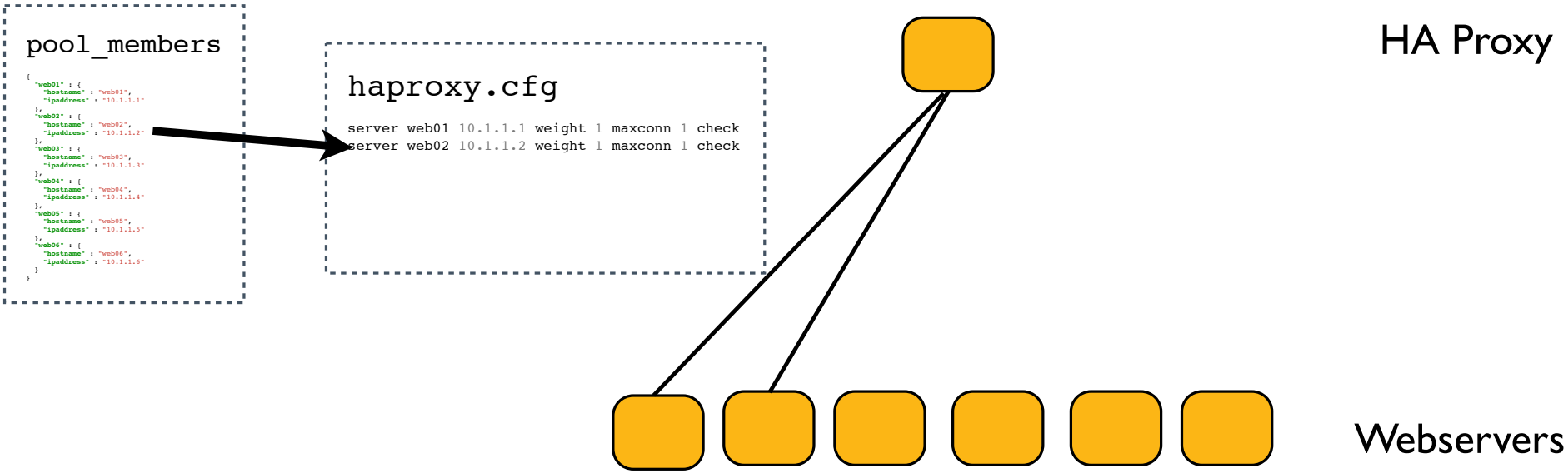


HA Proxy

Webservers

HAProxy Configuration

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



HAProxy Configuration

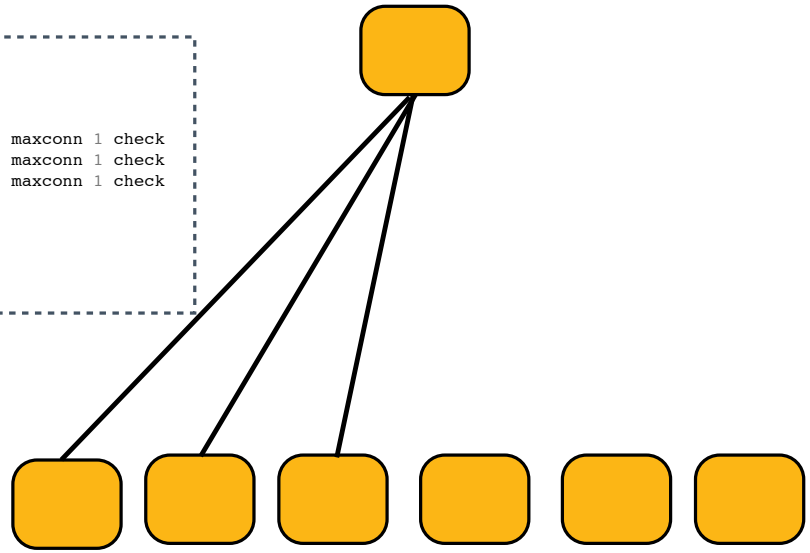
```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>:> weight 1 maxconn 1 check
<% end -%>
```

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"
  },
  "web05" : {
    "hostname" : "web05",
    "ipaddress" : "10.1.1.5"
  },
  "web06" : {
    "hostname" : "web06",
    "ipaddress" : "10.1.1.6"
  }
}
```


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
```



HA Proxy

Webservers

HAProxy Configuration

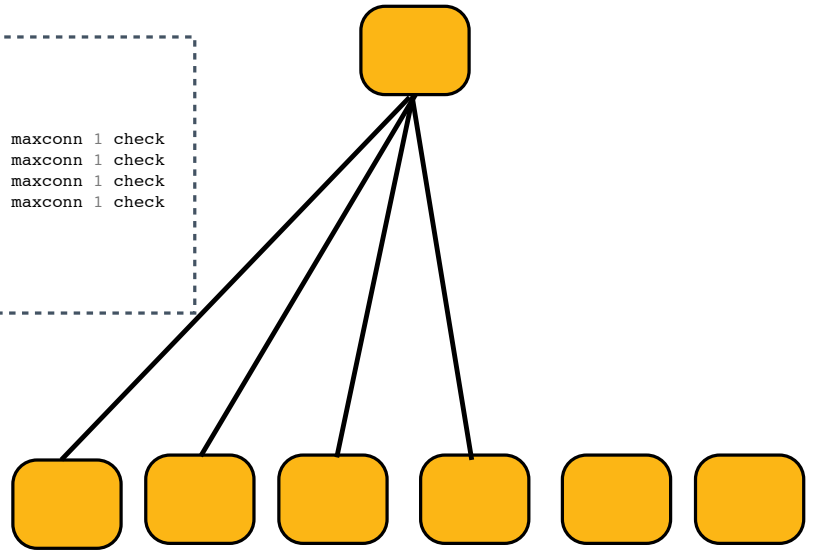
```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>:> weight 1 maxconn 1 check
<% end -%>
```

pool_members


```
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  },
  "web05" : {
    "hostname" : "web05",
    "ipaddress" : "10.1.1.5"
  },
  "web06" : {
    "hostname" : "web06",
    "ipaddress" : "10.1.1.6"
  }
}
```


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
```



HA Proxy

Webservers

HAProxy Configuration

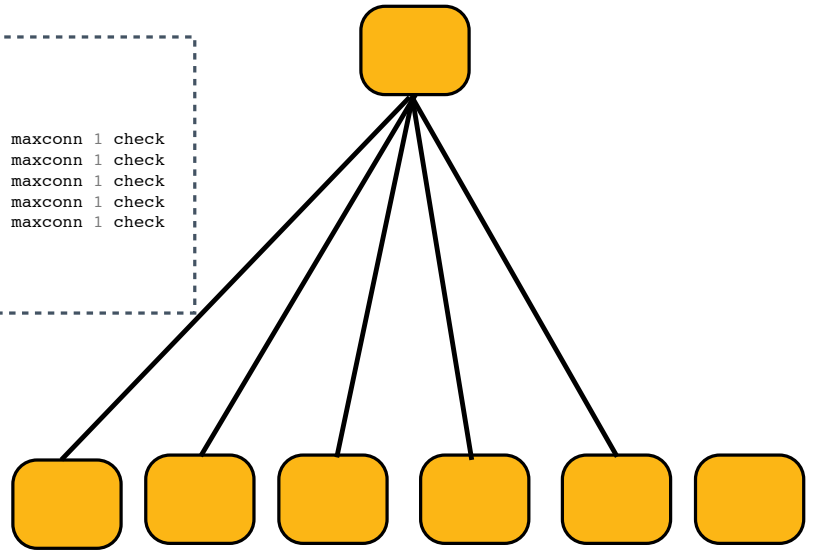
```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>:> weight 1 maxconn 1 check
<% end -%>
```

pool_members


```
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  },
  "web05" : {
    "hostname" : "web05",
    "ipaddress" : "10.1.1.5"
  },
  "web06" : {
    "hostname" : "web06",
    "ipaddress" : "10.1.1.6"
  }
}
```


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
server web05 10.1.1.5 weight 1 maxconn 1 check
```



HA Proxy

Webservers

HAProxy Configuration

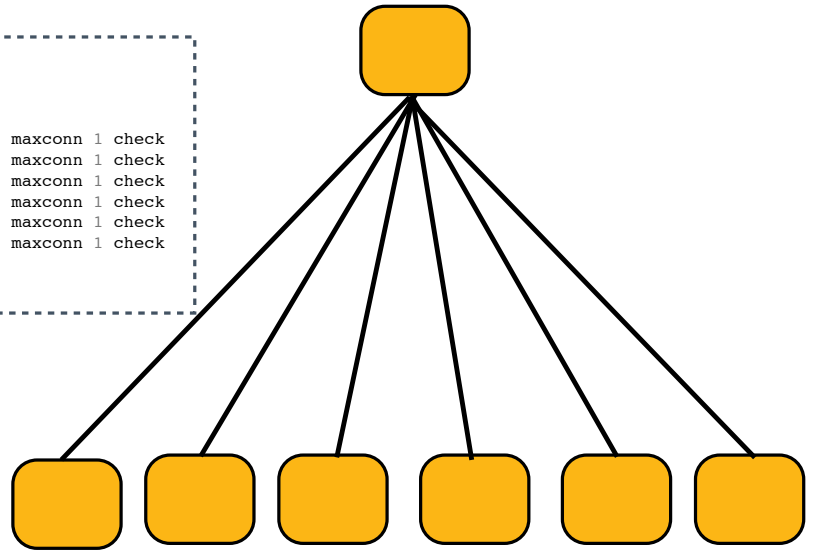
```
<% @pool_members.each do |member| -%>
server <%= member[:hostname] %> <%= member[:ipaddress] %>:> weight 1 maxconn 1 check
<% end -%>
```

pool_members


```
{
  "web01" : {
    "hostname" : "web01",
    "ipaddress" : "10.1.1.1"
  },
  "web02" : {
    "hostname" : "web02",
    "ipaddress" : "10.1.1.2"
  },
  "web03" : {
    "hostname" : "web03",
    "ipaddress" : "10.1.1.3"
  },
  "web04" : {
    "hostname" : "web04",
    "ipaddress" : "10.1.1.4"
  },
  "web05" : {
    "hostname" : "web05",
    "ipaddress" : "10.1.1.5"
  },
  "web06" : {
    "hostname" : "web06",
    "ipaddress" : "10.1.1.6"
  }
}
```


haproxy.cfg

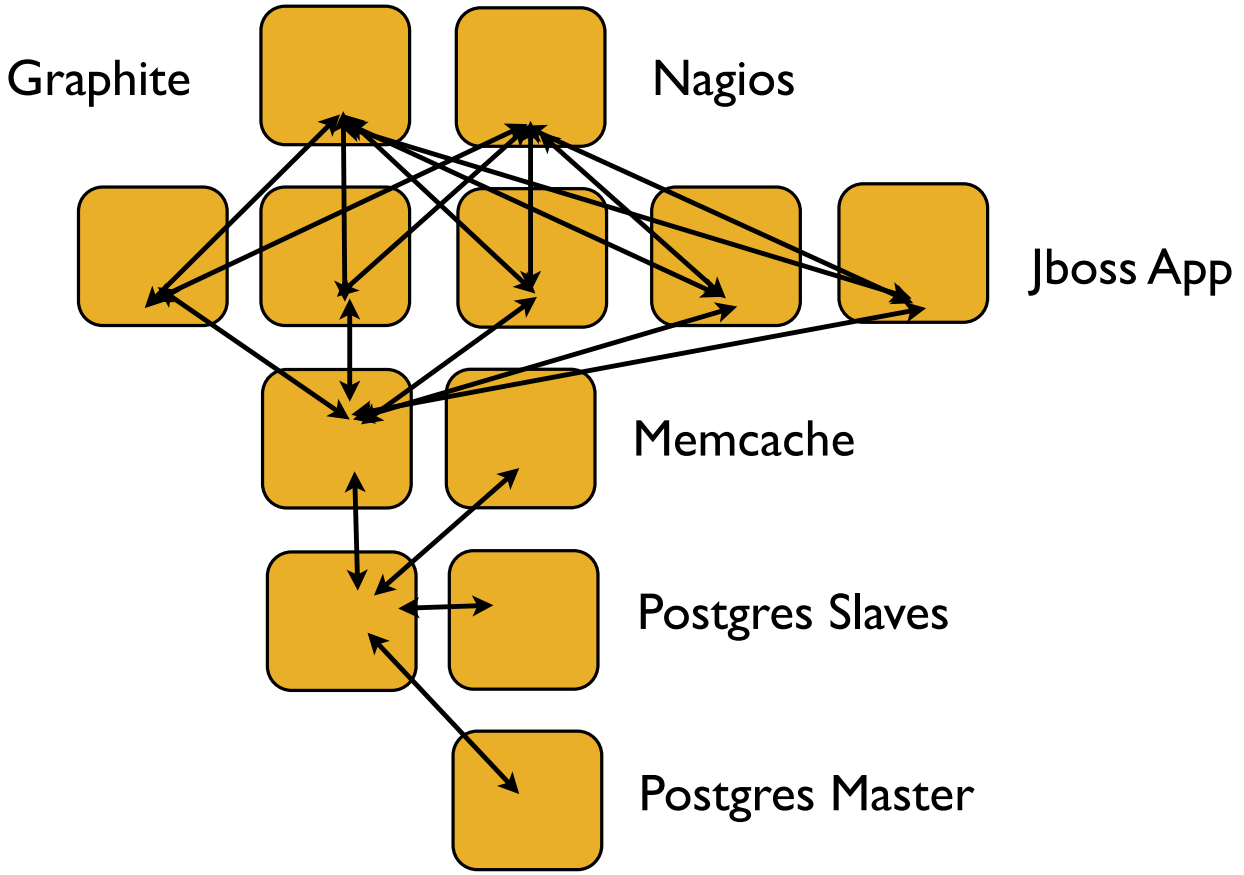
```
server web01 10.1.1.1 weight 1 maxconn 1 check
server web02 10.1.1.2 weight 1 maxconn 1 check
server web03 10.1.1.3 weight 1 maxconn 1 check
server web04 10.1.1.4 weight 1 maxconn 1 check
server web05 10.1.1.5 weight 1 maxconn 1 check
server web06 10.1.1.1 weight 1 maxconn 1 check
```



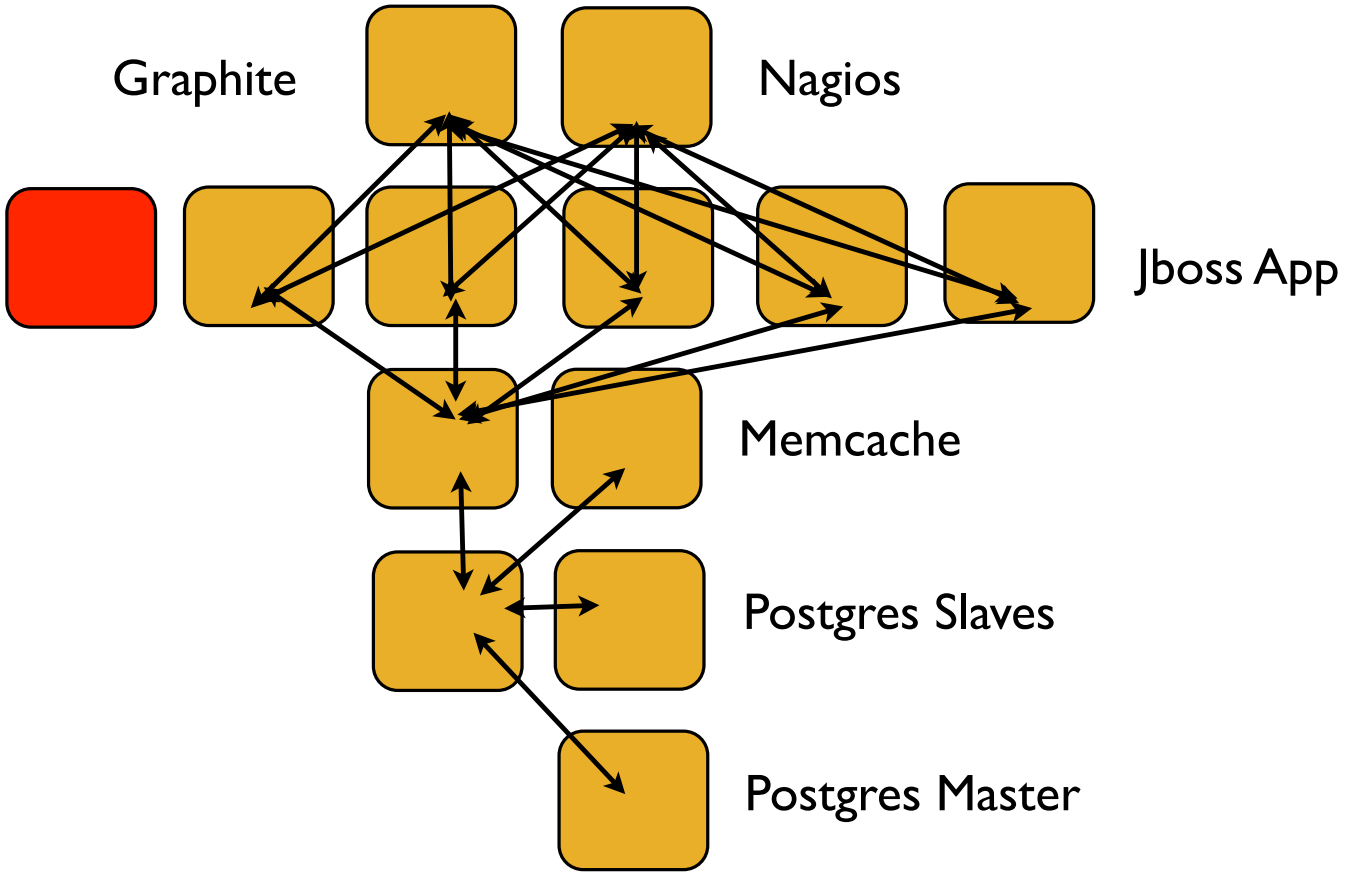
HA Proxy

Webservers

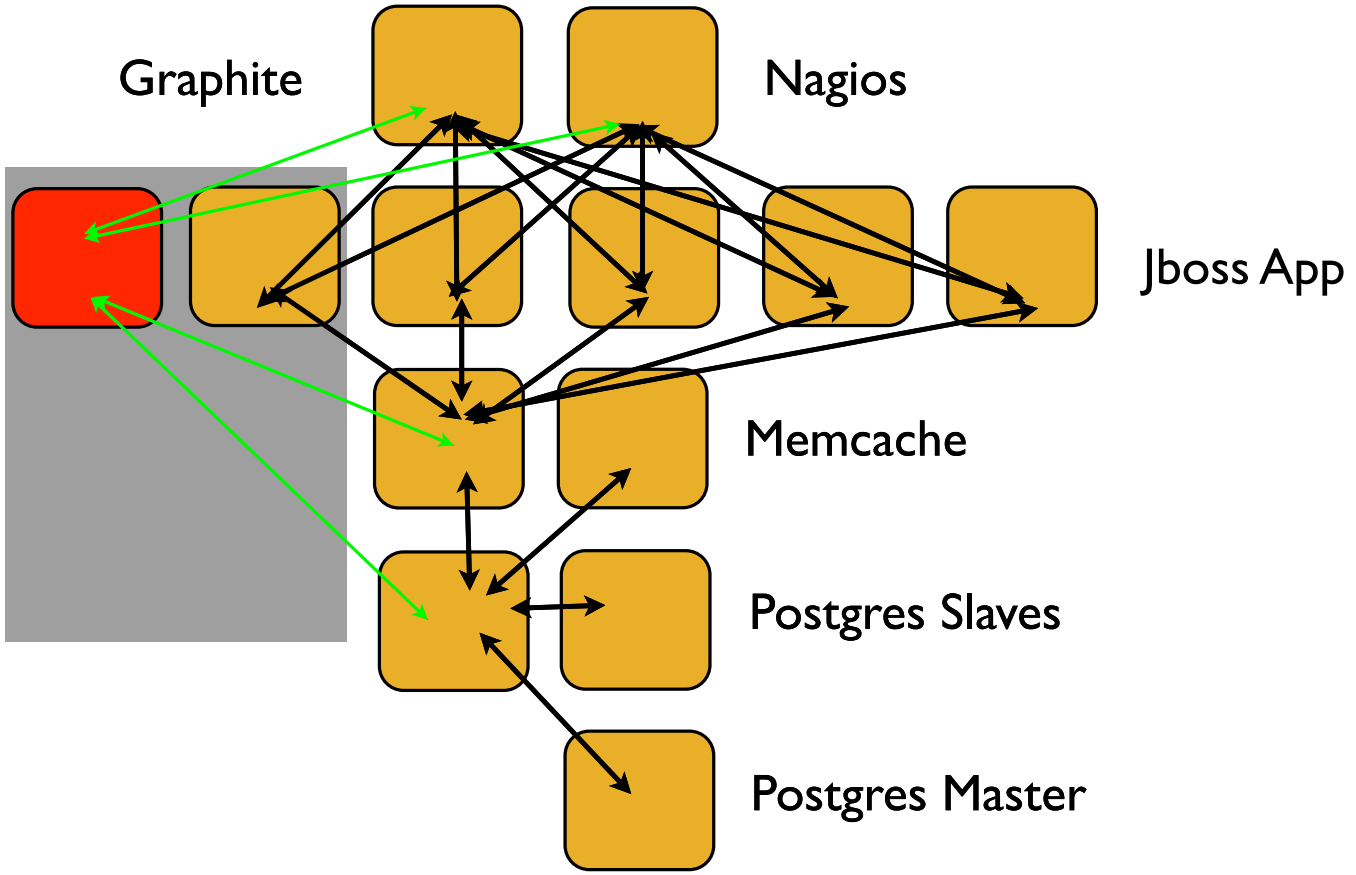
So when this...


Graphite
Nagios
Jboss App
Memcache
Postgres Slaves
Postgres Master

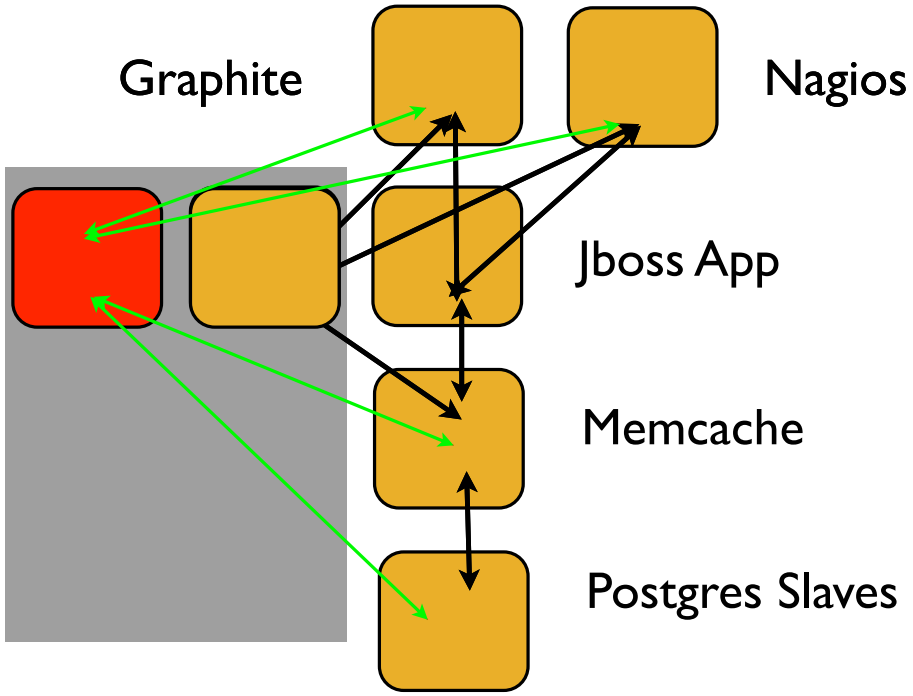
...becomes this



...this can happen automatically



Count the Resources



- 12+ resource changes for 1 node addition

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

Manage Complexity

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

Configuration Drift

- Configuration Drift happens when:
  - Your infrastructure requirements change
  - The configuration of a server falls out of policy

- Chef makes it easy to manage
  - Model the new requirements in your Chef configuration files
  - Run the chef-client to enforce your policies

Recap

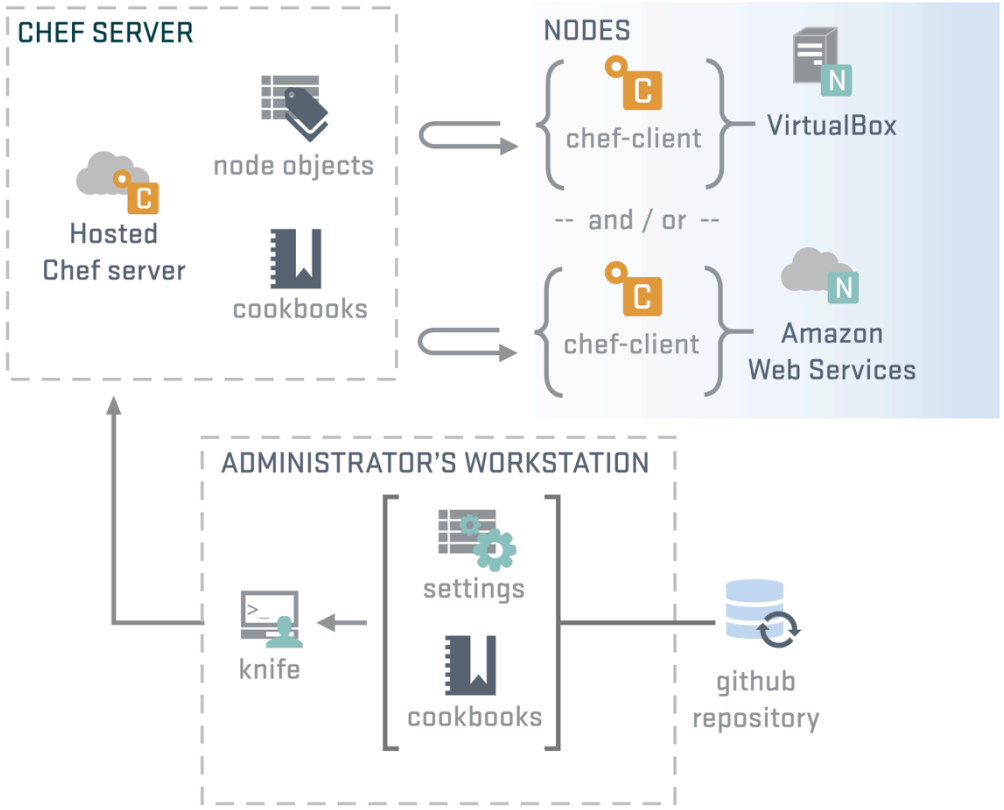
- In this section, we have
  - Described how Chef thinks about Infrastructure Automation
  - Defined the following terms:
    - Resource
    - Recipe
    - Node
    - Run List
    - Search

What Questions Do You Have?

Nathen Harvey
Community Director
nharvey@getchef.com
@nathenharvey

CHEF™
GETCHEF.COM

Chef Infrastructure

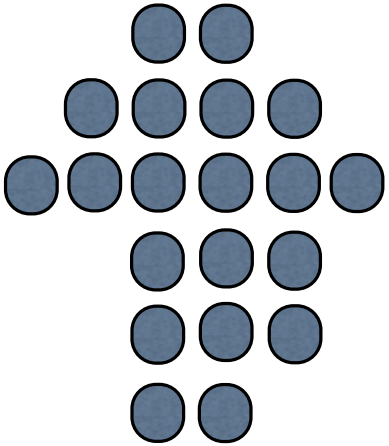


Sign-up for Hosted Chef

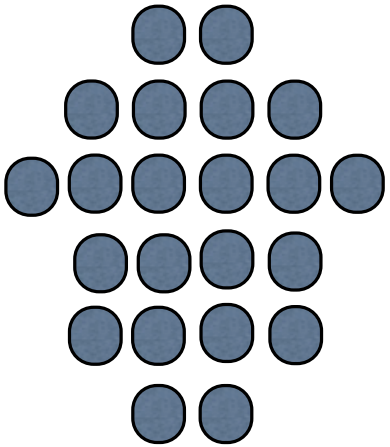
- http://getchef.com
- Click “Get Chef”
- Select “Hosted Chef”
- Complete the registration form
- Create an Organization

Organizations

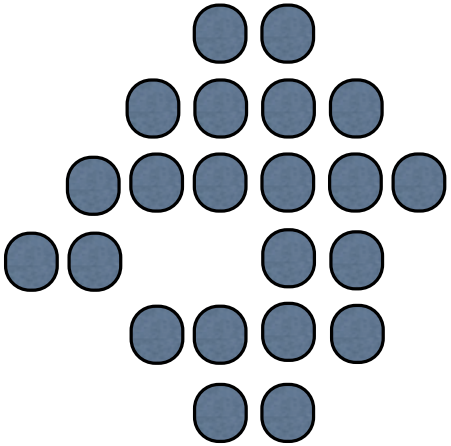
My Infrastructure



Your Infrastructure



Their Infrastructure



Organizations

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

Configure Workstation

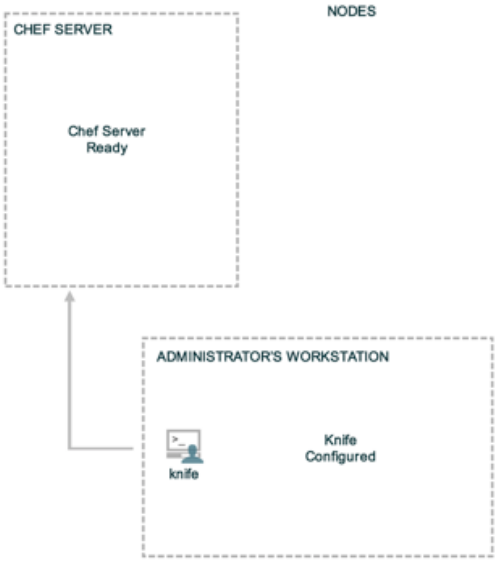
- Download and extract Chef starter kit
- Install chef-client
  - http://getchef.com/chef/install

Verify Knife

```
$ knife --version
Chef: 11.12.4
```

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

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



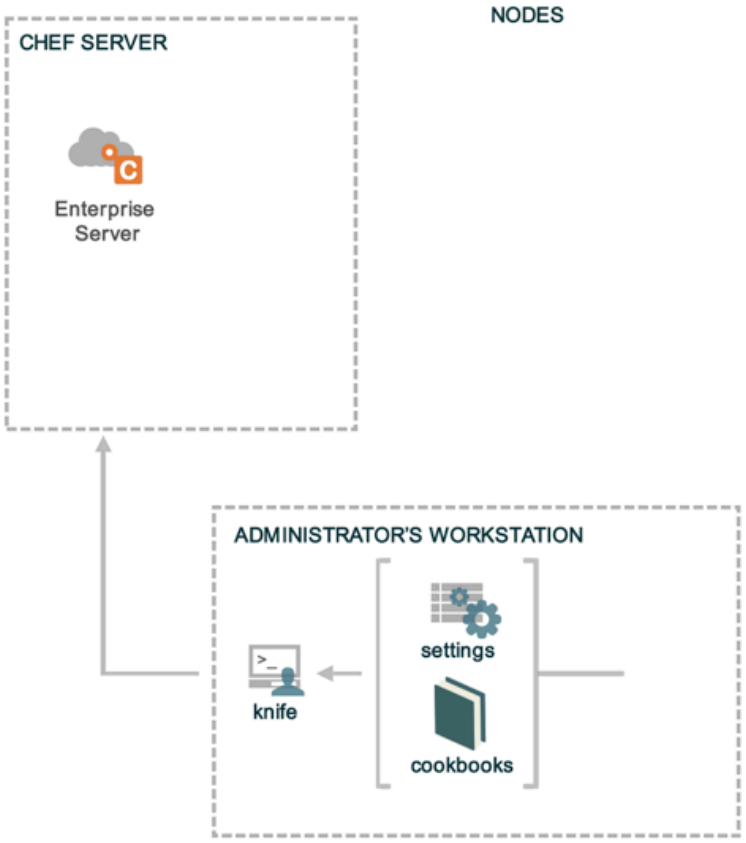
knife.rb

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

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

knife client list

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



Chef Fundamentals Webinar Series

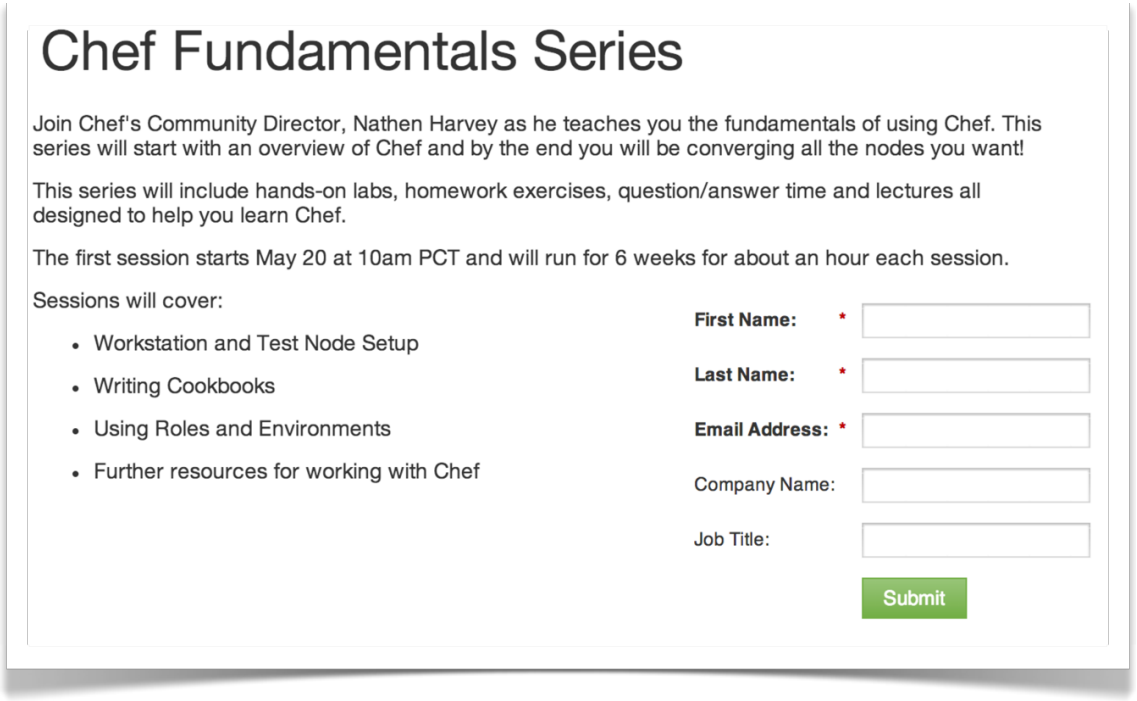
Six Week Series

- May 20 - Overview of Chef
- May 27 - Node Setup, Chef Resources & Recipes
- June 3 - Working with the Node object
- June 10 - Common configuration data with Databags
- June 17 - Using Roles and Environments
- June 24 - Community Cookbooks and Further Resources

- * Topics subject to change, schedule unlikely to change

Sign-up for Webinar

- http://pages.getchef.com/cheffundamentalsseries.html



Chef Fundamentals Series

Join Chef's Community Director, Nathen Harvey as he teaches you the fundamentals of using Chef. This series will start with an overview of Chef and by the end you will be converging all the nodes you want!

This series will include hands-on labs, homework exercises, question/answer time and lectures all designed to help you learn Chef.

The first session starts May 20 at 10am PCT and will run for 6 weeks for about an hour each session.

Sessions will cover:

- Workstation and Test Node Setup
- Writing Cookbooks
- Using Roles and Environments
- Further resources for working with Chef

First Name: *

Last Name: *

Email Address: *

Company Name:

Job Title:

Submit

Additional Resources

- Chef Fundamentals Webinar Series
- https://www.youtube.com/watch?v=S5lHUpzoCYo&list=PL11cZfNdwNyPnZA9D1MbVqldGuOWqbumZ

- Discussion group for webinar participants
- https://groups.google.com/d/forum/learnchef-fundamentals-webinar

Additional Resources

- Learn Chef
- http://learnchef.com

- Documentation
- http://docs.opscode.com