13: Search

Search
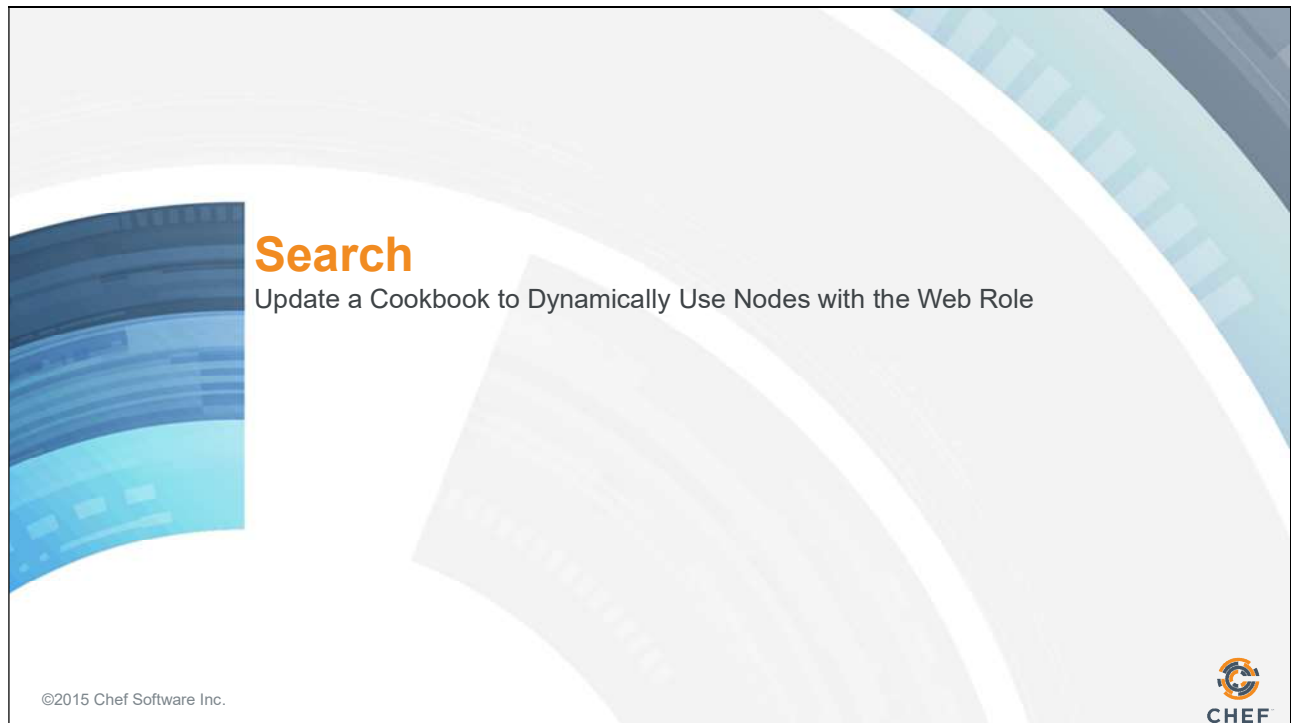Update a Cookbook to Dynamically Use Nodes with the Web Role

CHEF

Slide 2

Objectives

After completing this module, you should be able to

➢ Describe the query syntax used in search

➢ Build a search into your recipe code

➢ Create a Ruby Array and Ruby Hash

➢ Update the myhaproxy wrapper cookbook (for the load balancer) to dynamically use nodes with the web role

13- 2

In this module you will learn how to describe the query syntax used in search, build a search into your recipe code, create a ruby array and ruby hash, and update the myhaproxy wrapper cookbook to dynamically use nodes with the web role

Slide 3

So far we have seen how Chef is able to manage the policy of the nodes within our infrastructure.

We have two web servers and one load balancer. As more customers come to our website we can continue scale up to meet that demand.

Slide 4

To add new servers as load balancer members, we would need to bootstrap a new web server and then update our myhaproxy cookbook to include that new web server. But that seems dramatically inefficient to have to update a cookbook recipe.

A more ideal solution would be for the recipe to instead discover all of the web servers within our organization and automatically add them to a list of available members for our load balancer.

Slide 5

The Chef Server and Search

Chef Server maintains a representation of all the nodes within our infrastructure that can be searched on.

Search is a service discovery tool that allows us to query the Chef Server.

Chef Server

Web Nodes	Load Balancers
Node 1	Node 2
Node 3	Node 5
Node 8	Node 6
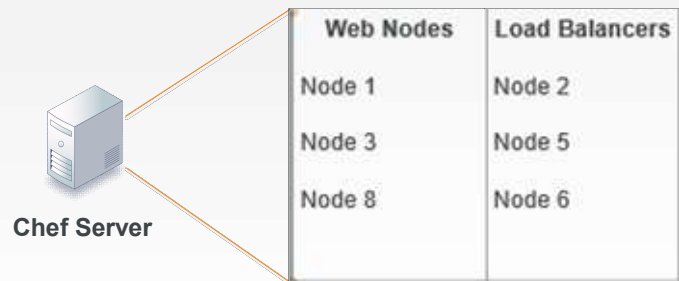
https://docs.chef.io/chef_search.html

https://docs.chef.io/chef_search.html#search-indexes

13- 5

CHEF

The Chef Server maintains a representation of all the nodes within an infrastructure and provides a way for us to discover these systems through Search.

Search is a service discovery tool that allows us to query the Chef Server across a few indexes. One such index is on our nodes.

Slide 6

The Chef Server and Search

We can ask the Chef Server to
return all the nodes or a subset
of nodes based on the query
syntax that we provide it
through `knife search` or within
our recipes through `search`.

Chef Server

Web Nodes	Load Balancers
Node 1	Node 2
Node 3	Node 5
Node 8	Node 6

13- 6
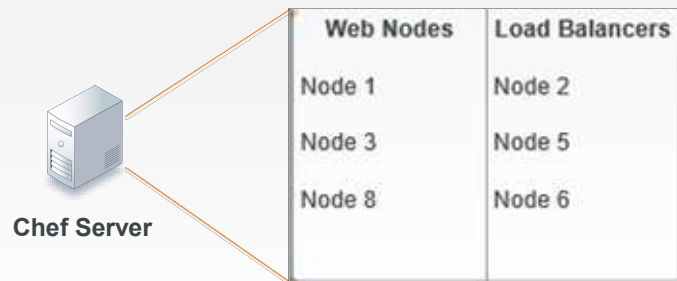
We can ask the Chef Server to return back to us all the nodes or a subset of nodes based
on the query syntax that we provide it through the knife command `knife search` or within
our recipes through the `search` method.

Search Criteria

The search criteria that we have been using up to this point is "*:*"

Querying and returning every node is not what we need to solve our current problem.

Scenario: We want only to return a subset of our nodes--only the nodes that are webservers.

We have been using a form of the search criteria already when we have employed the `knife ssh` command. The search criteria that we have been using up to this point is "*:*" which we explained matched every node within our infrastructure.

Querying and returning every node is not exactly what we need to solve our current problem. Scenario: We want only to return a subset of our nodes--only the nodes that are webservers.

Let's examine the search criteria more so we can understand how it works and how we can use it to find a subset of the nodes--only the nodes that are webservers.

Slide 8

Search Syntax

A search query is comprised of two parts: the key and the search pattern. A search query has the following syntax:

key:search_pattern

...where key is a field name that is found in the JSON description of an indexable object on the Chef server and search_pattern defines what will be searched for,

CHEF

A search query is comprised of two parts: the key and the search pattern. A search query has the following syntax:

key:search_pattern

...where key is a field name that is found in the JSON description of an indexable object on the Chef server (a role, node, client, environment, or data bag) and search_pattern defines what will be searched for.

Slide 9

Search Syntax within a Recipe

```
all_web_nodes = search('node','role:web')
```

creates and names a variable

assigns the value of the
operation on the right
into the variable on the left

invokes the search method

the index or items to search

the search criteria - key:value

CHEF

Search within a recipe is done through a `search` method that is available within the recipe.

The `search` method accepts two arguments. The first argument is a string or variable that contains the index or item to search on the Chef Server. These are: nodes; roles; and environments. The second argument is a string or variable that contains the search criteria to scope the results. This is using the notation 'key:value'.

The result of the search method is stored in a local variable that is named 'all_web_nodes'. Variables within Ruby are created immediately when you assign them.

Search Syntax within a Recipe

```
all_web_nodes = search('node','role:web')
```

Search the Chef Server for all node objects that have the role equal to 'web' and store the results into a local variable named ''all_web_nodes'.

13-10

This example syntax could be translated to mean: Search the Chef Server for all node objects that have the role equal to 'web' and store the results into a local variable named 'all_web_nodes'.

Slide 11

Hard Coding Example

`~/chef-repo/cookbooks/myhaproxy/recipes/default.rb`

```
node.default['haproxy']['members'] = [{
    'hostname'  => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',
    'ipaddress' => '52.8.71.11',
    'port' => 80,
    'ssl_port' => 80
  },
  {
    'hostname'  => 'ec2-54-176-64-173.us-west-1.compute.amazonaws.com',
    'ipaddress' => '54.175.46.48',
    'port' => 80,
    'ssl_port' => 80
  }
]
include_recipe 'haproxy::default'
```

©2015 Chef Software Inc.   13-11

Previously, we had been hard coding the hostname and ipaddress values in our wrapped haproxy recipe. We can request these values from the Chef Server through the `knife node show` command. The hostname and ipaddress values are captured by Ohai and sent to the Chef Server. On the Chef Server we can query those values when we ask about a specific attribute about the node. We do that by providing the `-a` flag with the name of the attribute. Because the nodes that we manage are hosted in the cloud, these attributes are stored under a parent attribute named 'cloud'.

Slide 12

GE: Dynamic Web Load Balancer

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!

Objective:

❑ Update the myhaproxy cookbook to dynamically use nodes with the web role

13-12

In this section we'll update the load balancer's myhaproxy cookbook to dynamically use nodes with the web role.

Slide 13

GE: Showing node1 Cloud Attributes

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

```
node1:
  cloud:
    local_hostname:  ip-10-198-51-26.us-west-1.compute.internal
    local_ipv4:      10.198.51.26
    private_ips:     10.198.51.26
    provider:        ec2
    public_hostname: ec2-204-236-155-223.us-west-1.compute.amazonaws.com
    public_ips:      204.236.155.223
    public_ipv4:     204.236.155.223
```

©2015 Chef Software Inc.   13-13

Here we are asking for all the 'cloud' attributes for 'node1'.

Slide 14

GE: Showing node3 Cloud Attributes

```
$ knife node show node3 -a cloud
```

```
node3:
  cloud:
    local_hostname:  ip-10-197-105-148.us-west-1.compute.internal
    local_ipv4:      10.197.105.148
    private_ips:     10.197.105.148
    provider:        ec2
    public_hostname: ec2-54-176-64-173.us-west-1.compute.amazonaws.com
    public_ips:      54.176.64.173
    public_ipv4:     54.176.64.173
```

CHEF

Here we are asking for all the 'cloud' attributes for 'node3'.

Slide 15

GE: Remove the Hard-coded Members

`~/chef-repo/cookbooks/myhaproxy/recipes/default.rb`

```
node.default['haproxy']['members'] = [{
    'hostname'  => 'ec2-52-8-71-11.us-west-1.compute.amazonaws.com',
    'ipaddress' => '52.8.71.11',
    'port' => 80,
    'ssl_port' => 80
  },
  {
    'hostname'  => 'ec2-54-176-64-173.us-west-1.compute.amazonaws.com',
    'ipaddress' => '54.175.46.48',
    'port' => 80,
    'ssl_port' => 80
  }
]
include_recipe 'haproxy::default'
```

CHEF

Edit the 'myhaproxy' cookbook's default recipe and remove the current default recipe where you hard-coded the members.

Slide 16

GE: Use Search to Identify the Members

`~/chef-repo/cookbooks/myhaproxy/recipes/default.rb`

```
all_web_nodes = search('node','role:web')

#TODO: Convert all found nodes into hashes with ipaddress,
#      hostname, port, ssl_port
#TODO: Assign all the hashes to the node's haproxy members
#      attribute.

include_recipe 'haproxy::default'
```

CHEF

Replace it with an updated recipe that searches for all nodes that have the 'web' role defined.

The search method's first parameter is asking the Chef Server to look at all the nodes within our organization.

The search method's second parameter is asking the Chef Server to only return the nodes that have been assigned the role web.

All of those nodes are stored in a local variable named `all_web_nodes`. This is an array of node objects. It may contain zero or more nodes that match the search criteria.

Creating an Array to Store the Converted Members

📄 `~/chef-repo/cookbooks/myhaproxy/recipes/default.rb`

```ruby
all_web_nodes = search('node','role:web')

members = []

#TODO: Convert all found nodes into hashes with ipaddress,
#      hostname, port, ssl_port

node.default['haproxy']['members'] = members

include_recipe 'haproxy::default'
```

13-17

Unfortunately we cannot simply assign our array of web nodes into the haproxy's members attributes because it needs a hash that contains the keys 'hostname', 'ipaddress', 'port', and 'ssl_port'. We will need to convert each of the web node objects into a structure that the haproxy member's attribute expects.

First we create an empty array and assign that empty array into a local variable named `members`. `members` is an array that we will populated with the hashes we will create later; until then we will write a TODO for us. Then we will assign that array into the `node.default['haproxy']['members']`.

Slide 18

Populating the Members with Each New Member

`~/chef-repo/cookbooks/myhaproxy/recipes/default.rb`

```ruby
all_web_nodes = search('node','role:web')

members = []

all_web_nodes.each do |web_node|
  member = {}
  # TODO: Populate the hash with hostname, ipaddress, port, and
  #       ssl_port
  members.push(member)
end

node.default['haproxy']['members'] = members

include_recipe 'haproxy::default'
```

CHEF

So we need to loop through the array of all the web nodes stored in `all_web_nodes`. We do that through a method available on every array object named 'each'. With the each method a block of code is provided -- you see it here from the first 'do' right after the each to the 'end' later in the file.

A block of code is an operation that you want perform on every item in the array. In our case we want to take each of the node objects and convert them into a hash object.

So every member of the array is visited and every member of the array runs through the block of code.

Populating the Hash with Node Details

~/chef-repo/cookbooks/myhaproxy/recipes/default.rb

```ruby
# ... BEFORE THE LOOP IN THE RECIPE ...

all_web_nodes.each do |web_node|
  member = {
    'hostname' => web_node['cloud']['public_hostname'],
    'ipaddress' => web_node['cloud']['public_ipv4'],
    'port' => 80,
    'ssl_port' => 80
  }
  members.push(member)
end

# ... AFTER THE LOOP IN THE RECIPE ...
```

©2015 Chef Software Inc.   13-19   CHEF

Between the pipes we see a local variable that we are defining that exists only in the block `web_node`. This local variable, `web_node`, is a name we came up with to refer to each node in our array of `all_web_nodes`. Each web node in the array is sent through the block. When inside the block of code it is referred to as `web_node`. Inside the block the first thing that is created is another local variable named `member` which is assigned a hash that contains the web_node's hostname and the web_node's ipaddress. Then the local variable `member`, which contains that hash is pushed into the array of members. This adds the member to the end of the array. When we are done looping through every web node the `members` array contains a list of all these hash objects.

Slide 20

The Final Recipe

`~/chef-repo/cookbooks/myhaproxy/recipes/default.rb`

```ruby
all_web_nodes = search('node','role:web')

members = []

all_web_nodes.each do |web_node|
  member = {
    'hostname' => web_node['cloud']['public_hostname'],
    'ipaddress' => web_node['cloud']['public_ipv4'],
    'port' => 80,
    'ssl_port' => 80
  }
  members.push(member)
end

node.default['haproxy']['members'] = members

include_recipe 'haproxy::default'
```

CHEF

This is the complete recipe source code.

A completed example can be found at:

https://raw.githubusercontent.com/chef-training/chef-essentials-repo/myhaproxy-complete/cookbooks/myhaproxy/recipes/default.rb

Slide 21

Dynamic Web Load Balancer

Every time we create a web node we need to update our load balancer (myhaproxy) cookbook. That doesn't feel right!

Objective:

✓ Update the myhaproxy cookbook to dynamically use nodes with the web role

13-21

The default recipe of the myhaproxy recipe is now dynamic. Every time a load balancer checks in with the Chef Server, when you run `chef-client`, it will ask the Chef Server if there are any new nodes that are web servers.

As you add nodes, your load balancer will dynamically grow to accommodate them, returning them as node objects, which are then converted to hashes, and then assigned as members.

As you remove nodes, your load balancer will dynamically shrink to accommodate them, returning a smaller set of node objects, which are then converted to hashes, and then assigned as members.

Slide 22

Lab: Upload the Cookbook

❑ Update the major version of the myhaproxy
cookbook
❑ Upload the cookbook
❑ Run chef-client on the load balancer node
❑ Verify that the load balancer node relays
requests to both web nodes

As a lab exercise:

* Update the major version of the cookbook

* Update the cookbook to the Chef Server

* Run `chef-client` on the load balancer node

* Verify that the load balancer node still relays requests to both of our web servers

Instructor Note: Allow 8 minutes to complete this exercise.

Slide 23

Lab: Update the Version Number

`~/chef-repo/cookbooks/myhaproxy/metadata.rb`

```
name              'myhaproxy'
maintainer        'The Authors'
maintainer_email  'you@example.com'
license           'all_rights'
description        'Installs/Configures myhaproxy'
long_description  'Installs/Configures myhaproxy'
version           '1.0.0'

depends 'haproxy', '~> 1.6.6'
```

First we update the version to the next major release. We set the version number to 1.0.0.

Lab: CD and Install Dependencies

```
$ cd ~/chef-repo/cookbooks/myhaproxy
$ berks install
```

```
Resolving cookbook dependencies...
Fetching 'myhaproxy' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Using build-essential (2.2.3)
Using cpu (0.2.0)
Using haproxy (1.6.6)
Using myhaproxy (1.0.0) from source at .
```

©2015 Chef Software Inc.   13-24

CHEF

Change into the cookbook's directory and then install any new dependencies that your cookbook may need at version 1.0.0.

We have no new dependencies but this is required by berkshelf whenever you update the version of the cookbook.

Slide 25

Lab: Upload the Cookbook

```
$ berks upload

Uploaded build-essential (2.2.3) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded cpu (0.2.0) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded haproxy (1.6.6) to:
'https://api.opscode.com:443/organizations/steveessentials2'
Uploaded myhaproxy (1.0.0) to:
'https://api.opscode.com:443/organizations/steveessentials2'
PS C:\Users\sdelfante\chef-repo\cookbooks\myhaproxy>
```

CHEF

Upload the cookbook using the `berks upload` command.

Instructor Note: During the course the learner may find they have a mistake with the cookbook and need to re-upload the cookbook. Berkshelf will 'freeze' the versions of the cookbooks that you upload. This is to prevent you from accidently overriding cookbooks that you may have already created. It is a best practice to not re-upload a cookbook again with new changes if they share the same version. During this course, however, it is important that the version numbers be aligned to make future sections work correctly so it OK to do in the training environment. To re-upload a cookbook with Berkshelf replacing the existing cookbook can be done with `berks upload --force`

Slide 26

Lab: Run the 'knife ssh' Command

```
$ knife ssh "role:load_balancer" -x USER -P PWD "sudo
chef-client"
```

```
ec2-54-210-192-12.compute-1.amazonaws.com Starting Chef Client, version 12.3.0
ec2-54-210-192-12.compute-1.amazonaws.com resolving cookbooks for run list:
["myhaproxy"]
ec2-54-210-192-12.compute-1.amazonaws.com Synchronizing Cookbooks:
ec2-54-210-192-12.compute-1.amazonaws.com    - build-essential
ec2-54-210-192-12.compute-1.amazonaws.com    - cpu
ec2-54-210-192-12.compute-1.amazonaws.com    - haproxy
ec2-54-210-192-12.compute-1.amazonaws.com    - myhaproxy
ec2-54-210-192-12.compute-1.amazonaws.com Compiling Cookbooks...
ec2-54-210-192-12.compute-1.amazonaws.com Converging 9 resources
ec2-54-210-192-12.compute-1.amazonaws.com Recipe: haproxy::install_package
ec2-54-210-192-12.compute-1.amazonaws.com    * yum_package[haproxy] action
```

CHEF

Use `knife ssh` and ask only the nodes with the role load_balancer' to run `sudo chef-client`. This is more efficient than targeting all of the nodes as we did before and more accurate than targeting the node2 "name:node2".

This ensures that all nodes that are also load balancers check in with the Chef Server-- similar to how we are targeting only the web server nodes in the recipe.

Slide 27
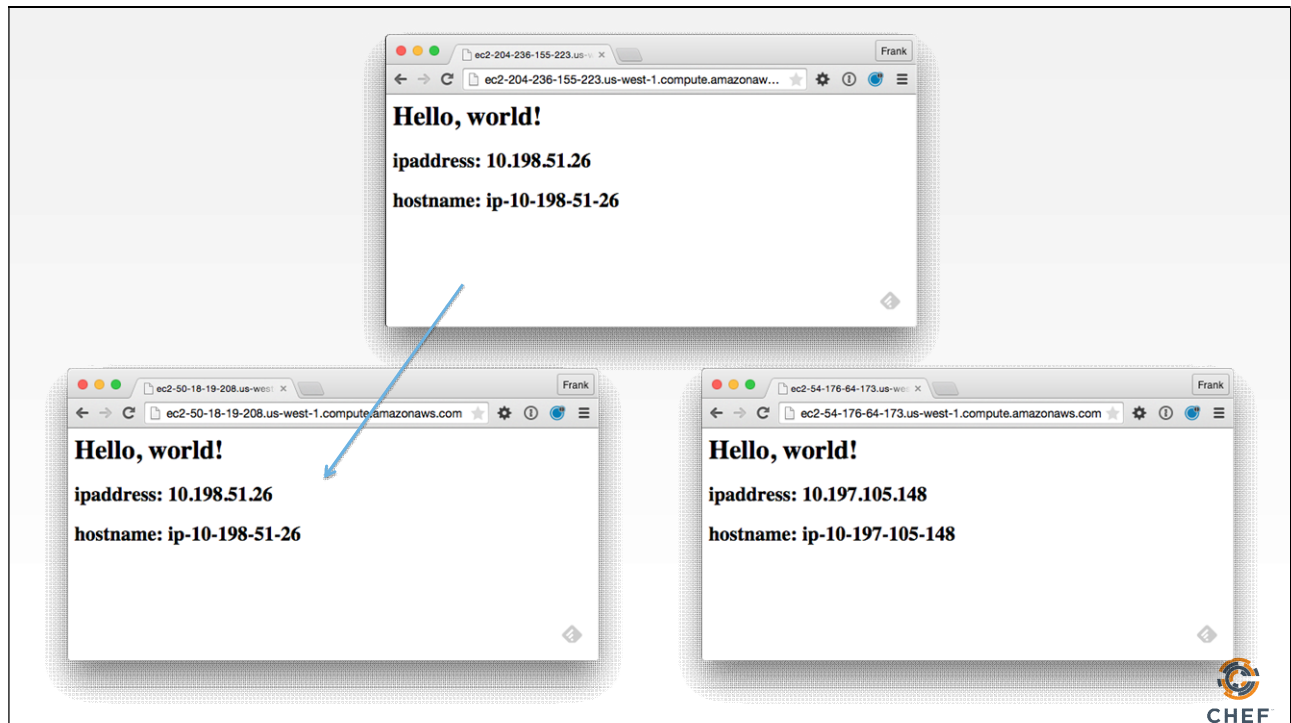
Nothing should change externally. You may see some differences in the logs as the load_balancer configuration file might change the order of the two entries but the end results is that our load balancer node is still delivering traffic to our two web server nodes.

Slide 29
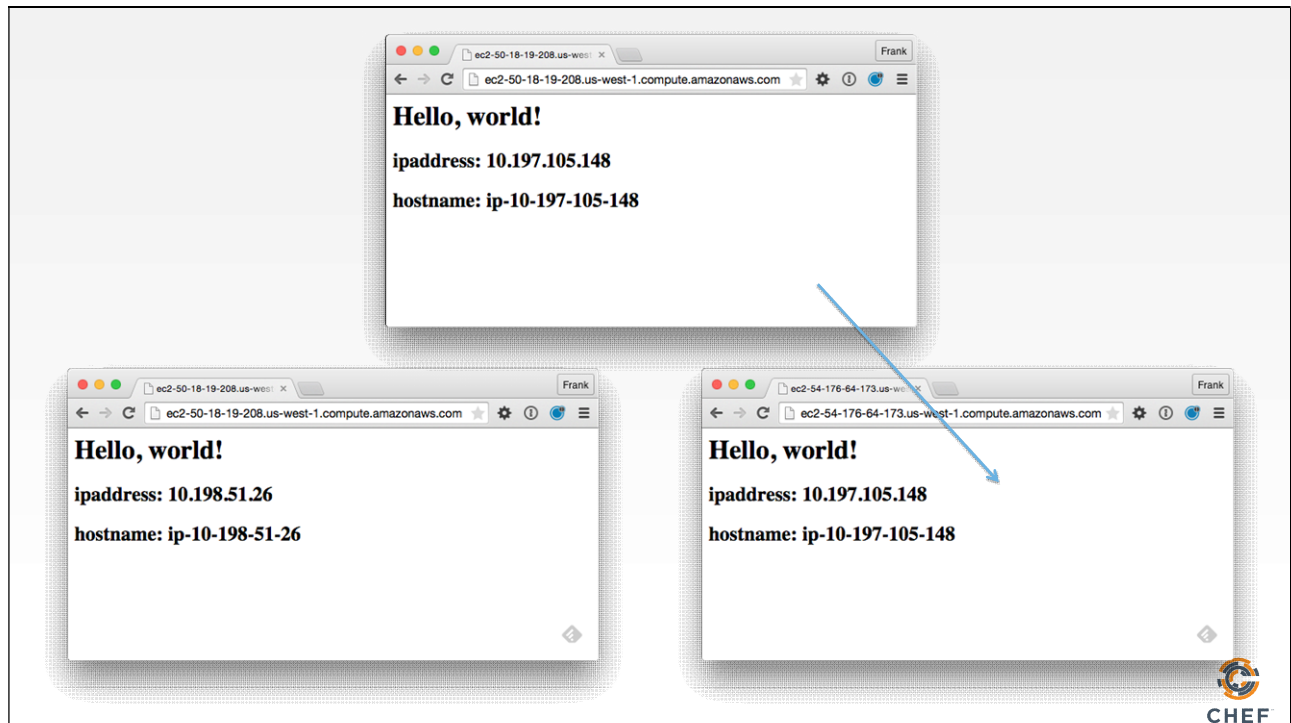
DISCUSSION

Discussion

What happens when new web nodes are added to the organization? Removed?

What happens if you were to terminate a web node instance without removing it from the Chef Server?

CHEF

Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

"Terminate" here means to turn off the machine or have the cloud provider disable the machine so that it is no longer online and network addressable.

Slide 30

Slide 31