**More than one website**

**Heads up!** To follow along with the examples in this chapter you'll need the Hosted Chef account and set up from the last chapter.

This chapter extends our cookbook to work with multiple web sites. It introduces slightly more advanced Chef concepts like data bags and search.

First we'll do a bit of house keeping and learn a new Chef concept. Our **phpapp** recipe was a bit verbose. It would be nice if we could group all the resources required to setup a WordPress site into one call and put that call elsewhere like a function or method. Well we can do exactly that with [definitions](http://docs.opscode.com/essentials_cookbook_definitions.html).

**Definitions**

Definitions are good for abstracting parts of your recipe into common code that can be used in other recipes or even cookbooks. Let's move our WordPress setup code into a definition.

Create *cookbooks/phpapp/definitions/wordpress\_site.rb*.

define :wordpress\_site, :path => "/var/www/phpapp", :database => "phpapp", :db\_username => "phpapp", :db\_password => "phpapp", :template => "site.conf.erb" do

wordpress\_latest = Chef::Config[:file\_cache\_path] + '/wordpress-latest.tar.gz'

remote\_file wordpress\_latest do

source 'http://wordpress.org/latest.tar.gz'

mode 0644

end

directory params[:path] do

owner 'root'

group 'root'

mode 0755

action :create

recursive true

end

execute 'untar-wordpress' do

cwd params[:path]

command 'tar --strip-components 1 -xzf ' + wordpress\_latest

creates params[:path] + '/wp-settings.php'

end

wp\_secrets = Chef::Config[:file\_cache\_path] + '/wp-secrets.php'

remote\_file wp\_secrets do

source 'https://api.wordpress.org/secret-key/1.1/salt/'

action :create\_if\_missing

mode 0644

end

salt\_data = ''

ruby\_block 'fetch-salt-data' do

block do

salt\_data = File.read(wp\_secrets)

end

action :create

end

template params[:path] + '/wp-config.php' do

source 'wp-config.php.erb'

mode 0755

owner 'root'

group 'root'

variables(

:database => params[:database],

:user => params[:db\_username],

:password => params[:db\_password],

:wp\_secrets => salt\_data)

end

# Due to a Chef quirk we can't pass our params to another definition

docroot = params[:path]

server\_name = params[:name]

web\_app server\_name do

template "site.conf.erb"

docroot docroot

server\_name server\_name

end

end

You might recogonise the above as slightly modified code from our recipe from the first chapter. You'll see we've replaced various node attributes with **params["param"]**. Those parameters are defined and named in the **define** line.

We haven't included the database and database access resources in our definition. That's because setting up a database might not necessarily be the job of the Chef recipe creating the website configuration.

Add the code above and save the file. We'll now modify our recipe to use the code.

Open *cookbooks/phpapp/recipes/default.rb*.

#

# Cookbook Name:: phpapp

# Recipe:: default

#

# Copyright 2013, YOUR\_COMPANY\_NAME

#

# All rights reserved - Do Not Redistribute

#

include\_recipe "apache2"

include\_recipe "mysql::client"

include\_recipe "mysql::server"

include\_recipe "php"

include\_recipe "php::module\_mysql"

include\_recipe "apache2::mod\_php5"

include\_recipe "mysql::ruby"

apache\_site "default" do

enable false

end

mysql\_database node['phpapp']['database'] do

connection ({:host => 'localhost', :username => 'root', :password => node['mysql']['server\_root\_password']})

action :create

end

mysql\_database\_user node['phpapp']['db\_username'] do

connection ({:host => 'localhost', :username => 'root', :password => node['mysql']['server\_root\_password']})

password node['phpapp']['db\_password']

database\_name node['phpapp']['database']

privileges [:select,:update,:insert,:create,:delete]

action :grant

end

Remove the code until your recipe looks like above or just replace it all with what's above.

Add the following code.

wordpress\_site node["phpapp"]["server\_name"] do

path "/var/www/phpapp"

database node["phpapp"]["database"]

db\_username node["phpapp"]["db\_username"]

db\_password node["phpapp"]["db\_password"]

template "site.conf.erb"

end

Now save the file. Lets upload our edited cookbook to the Chef Server.

$ knife cookbook upload phpapp

Uploading phpapp [0.1.0]

Uploaded 1 cookbook.

Let's create a new node. Use knife to bootstrap a node or create a cloud instance. A couple of example commands are listed below but refer to the previous chapter for more details.

* [Bootstrap via SSH](http://gettingstartedwithchef.com/more-than-one-website.html#cmd1-bootstrap)
* [Brightbox](http://gettingstartedwithchef.com/more-than-one-website.html#cmd1-brightbox)
* [Rackspace](http://gettingstartedwithchef.com/more-than-one-website.html#cmd1-rackspace)
* [Amazon](http://gettingstartedwithchef.com/more-than-one-website.html#cmd1-amazon)

$ knife bootstrap --run-list "role[phpapp]" --sudo hostname

Replace hostname with the name or IP of the machine.

Once **chef-client** on the node has completed, confirm your cookbook still works by visiting the IP or hostname of the machine. You should see the WordPress installation page.

**More than one WordPress site**

Our **phpapp** cookbook is all very well but what if we want host more than one WordPress site on our web server? What if we want to add new sites in the future without changing our cookbook? We need somewhere to store this data somewhere and [data bags](http://docs.opscode.com/essentials_data_bags.html) are ideal.

**Data bags**

A *data bag* is a collection of bits of JSON called *data bag items*, indexed by an ID, that Chef allows us to use and search in our recipes. Let's use knife to create our data bag.

Return to your chef-repo from the second chapter.

Type the following on your workstation to create a data bag called **wp-sites**.

$ knife data bag create wp-sites

Created data\_bag[wp-sites]

We'll call our site hosts **website1.example.com** and **website2.example.com**. Imaginative I know. Let's create our first website as the data bag item **website1**.

$ knife data bag create wp-sites website1

You may receive the following error if you do not have an EDITOR environment variable set.

ERROR: RuntimeError: Please set EDITOR environment variable

You can resolve this quickly with the **--editor** option. e.g. **--editor vi** or **--editor nano** etc.

Now we will add our per site configuration as JSON attributes. As well as a different hostname for each site, we want to be able to configure a separate database name, user, password for each of the sites on our server so we'll include these details in our data bag.

{

"id": "website1",

"host": "website1.example.com",

"database": "website1",

"db\_username": "website1",

"db\_password": "212b09752d173876a84d374333ae1ffe"

}

Add the code in green. Save the file. We now have our first data bag item. Let's create another.

$ knife data bag create wp-sites website2

Our second website will be similar to the first!

{

"id": "website2",

"host": "website2.example.com",

"database": "website2",

"db\_username": "website2",

"db\_password": "e50586910465ad767b36d11ec3fe323c"

}

Add the code in green. Save the file. We now have two data bags! Now time for something cool!

**Chef Shell**

As well as cookbooks, recipes and knife, we can also use [chef-shell](http://docs.opscode.com/chef_shell.html) to interact with Chef. Let's use it to take a look at our new data bag.

$ chef-shell -c .chef/knife.rb

loading configuration: none (standalone session)

Session type: standalone

Loading...done.

This is the chef-shell.

Chef Version: 11.x.x

http://www.opscode.com/chef

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

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

Ohai2u andy@computer!

chef >

Those familiar with Python or the IRb (Interactive Ruby Shell) already know how you use **chef-shell**. We can just type cookbook code in and see what happens! Useful for experimenting or debugging.

Let's get our data bag from the Chef Server. From inside chef-shell.

chef > databags('wp-sites').list

=> [data\_bag\_item["wp-sites", "website2", {"id"=>"website2", "host"=>"website2.example.com", "database"=>"website2", "db\_username"=>"website2", "db\_password"=>"e50586910465ad767b36d11ec3fe323c"}], data\_bag\_item["wp-sites", "website1", {"id"=>"website1", "host"=>"website1.example.com", "database"=>"website1", "db\_username"=>"website1", "db\_password"=>"212b09752d173876a84d374333ae1ffe"}]]

We can see our entire data bag is returned. Let's find our which data bag item contains the host **"website2.example.com"**.

chef > databags('wp-sites').search "host:website2.example.com"

=> [data\_bag\_item["wp-sites", "website2", {"id"=>"website2", "host"=>"website2.example.com", "database"=>"website2", "db\_username"=>"website2", "db\_password"=>"e50586910465ad767b36d11ec3fe323c"}]]

You can see one data bag item is returned.

We'll come back to chef-shell later. Press Ctrl-D to quit.

**Using data bags in your recipes**

Load *cookbooks/phpapp/recipes/default.rb* in your text editor. Locate the **wordpress\_site** definition we created earlier. Replace it and the **mysql\_database** and **mysql\_database\_user** resource calls with the following code.

sites = data\_bag("wp-sites")

sites.each do |site|

opts = data\_bag\_item("wp-sites", site)

mysql\_database opts["database"] do

connection ({:host => 'localhost', :username => 'root', :password => node['mysql']['server\_root\_password']})

action :create

end

mysql\_database\_user opts["db\_username"] do

connection ({:host => 'localhost', :username => 'root', :password => node['mysql']['server\_root\_password']})

password opts["db\_password"]

database\_name opts["database"]

privileges [:select,:update,:insert,:create,:delete]

action :grant

end

wordpress\_site opts["host"] do

path "/var/www/" + opts["host"]

database opts["database"]

db\_username opts["db\_username"]

db\_password opts["db\_password"]

template "site.conf.erb"

end

end

We'll explain the key parts of that now.

First we retrieve a list of items in our data bag.

sites = data\_bag("wp-sites")

We then loop through that list of items (which is called iterating).

sites.each do |site|

We get the contents of the data bag we are looking at and store it in **opts**.

opts = data\_bag\_item("wp-sites", site)

We then pass the values in opts to our resources and our **wordpress\_site** definition.

Save the file and let's try this out. Upload our cookbook to Hosted Chef with knife.

$ knife cookbook upload phpapp

Uploading phpapp [0.1.0]

Uploaded 1 cookbook.

SSH to the box we created earlier on. Run chef-client as root.

# chef-client

..

Chef Client finished, 10 resources updated

We now have two websites.

To test if that's worked, create two hosts file entries. One for website1.example.com and one for website2.example.com.

[Rackspace have a handy guide](http://www.rackspace.com/knowledge_center/article/how-do-i-modify-my-hosts-file) if you don't know how to do it for your workstation.

Visit [website1.example.com](http://website1.example.com) and [website2.example.com](http://website2.example.com). Fill in the WordPress installation page for each one (giving them a different Site Title) then visit the two different sites. You'll see they are different sites. So that means we can now run multiple websites on our Chef configured server.

Now adding a new site to our web server is easy. Let's create website3. All we need to do is create a new data bag item. Back on our workstation run the following command.

$ knife data bag create wp-sites website3

Add the following code in green then save the file.

{

"id": "website3",

"host": "website3.example.com",

"database": "website3",

"db\_username": "website3",

"db\_password": "b96b1ae6deb7da0e042de904cf0ba70d"

}

SSH back to the box running our web server. Re-run chef-client.

# chef-client

..

Chef Client finished, 14 resources updated

Create a hosts entry for website3.example.com on your workstation. Visit [website3.example.com](http://website3.example.com). You'll get another WordPress installation page. You can create as many WordPress sites as you like!

In the next chapter we'll discuss extending our cookbook to handle running our application on multiple nodes.