/Chef/cooking up servers

Download the files! > All the files you need for this tutorial can be found at http://netm.ag/chef-237

Cloud hosting has made running multi-server sites affordable for small businesses and teams. **Gavin Montague** explains how to make managing these servers easier with Chef

Knowledge needed: Basic Unix
Requires: Chef, cloud server, SSH client
Project time: 2 hours

One of the first problems any successful web application meets is scaling its servers to match user demand. Where your site once happily ran on a single £100-a-year host, you soon find yourself juggling 10, 20 and, hopefully, hundreds of servers. At the extreme end of the scale, consider Instagram: zero to one hundred million users in two years. That must have been fun for their sysadmins.

Scaling an application is a unique challenge, but a big part of any solution is the management, maintenance and tuning of the server infrastructure underlying everything. Nearly all sysadmins have built up toolbelts of shell scripts, notes detailing gotchas and half-remembered configuration files that they rely on. "Run this script to set up a new user, remember that version 1.2.3 isn't compatible with this version of Python; remember to run script-001 before script-01. Wait... have I already run script-01? Why is there a comment that says: 'Watch out for errors'?" That way lies madness. Thankfully Chef is here to help.

Introducing Chef

An open source product by Opscode (www.opscode.com), Chef is a class of software normally referred to as a systems integration framework. It provides a toolkit for abstracting the work of hand-crafting servers into a more mechanical, controllable and engineered process.

The central premise of Chef is that the act of doing something on a server can be turned into programmatic expressions. But why would we want to do this? Well, firstly the abstraction enables us to hide differences between operating systems with an interface of intent rather than action. Ubuntu, Red Hat, CentOS and Windows all have slightly different ways of managing long-running processes. Were I to try and restart Apache across a mixed cluster of boxes, I'd need to remember the exact invocation for each box. With Chef I can declare:

```
service "apache2" { action :restart }
```

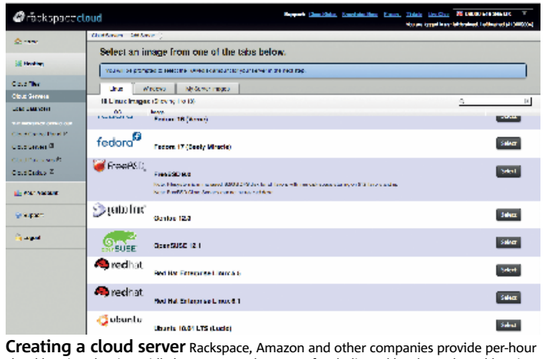
...and trust that Chef knows how to achieve the desired action across all systems. Similarly, for installing software Chef not only deals with operating system differences, but with the states of each server. Whether I have one server or one hundred, Chef can make sure that they all have PHP 6.0 installed, limited to 48MB of memory and with PostgreSQL bindings enabled. It doesn't matter if some of the boxes already had PHP installed, if some are on older versions or if they're all entirely clean – Chef works out the changes required to converge the box into the desired state.

> The act of doing something on a server can be turned into programmatic expressions

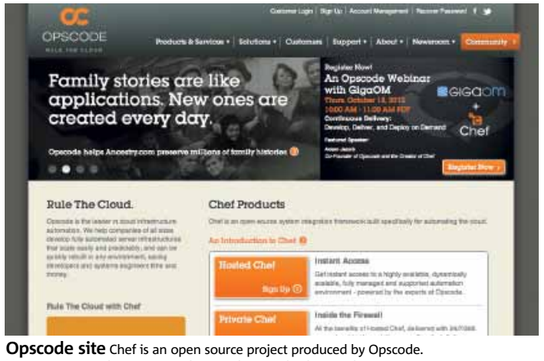
Developers can build up complex roles for servers by combining many recipes and attributes. Rather than being carefully crafted, hard-to-reproduce one-offs, these managed servers become throw-away commodities that can be built from scratch in minutes as opposed to hours or even days.

Away from the data centre, a growing number of developers use tools like Chef to manage their office computers. There's something quite awesome about being able to sit down at a fresh-out-of-the-box laptop, run a handful of commands and come back 30 minutes later to a just-the-way-you-like-it set-up with all your programs, preferences and shortcuts magically in place.

Chef's major strength is its ability to orchestrate massive armies of servers, but it's equally suitable for managing single boxes. In this tutorial we're going to walk through setting up a Chef workstation and node on a remote server before deploying some software to it.



Creating a cloud server Rackspace, Amazon and other companies provide per-hour cloud hosting that is rapidly become a replacement for dedicated hardware-based hosting



Opscode site Chef is an open source project produced by Opscode. Visit www.opscode.com to read more about its background

Glossary: Chef terms

Chef introduces quite a few terms early on, and getting your head around what they mean and how they fit together is the first step towards becoming confident in using it. Here's a quick guide to the most commonly used jargon: we won't use it all in this tutorial, but if you're looking to expand upon it by reading the Chef documentation (http://wiki.opscode.com) this glossary will come in handy.

- **Node** – any computer (or other hardware) managed by Chef.
- **Workstation** – the machine from which you build Chef cookbooks and issue commands to the Chef Server
- **Chef Server** – The central hub through which your commands are read by nodes. Also provides a user-friendly web app that replicates some of the functions of Knife. Opscode provides a free, hosted Chef server, which entails considerably less work than setting up your own.
- **Chef Client** – the program that runs on a node. The client carries out the instructions of the Chef Server and delivers feedback on how the run went and the current state of the node.
- **Knife** – The command line tool through which your workstation talks to the Chef Server. We'll use it here to download/upload cookbooks and manipulate the run_list, but it can also be used to search and edit nodes, create cookbooks and even provision new servers with common cloud hosting providers.
- **Cookbook** – a collection of recipes and supporting files that are packed together into a single unit. For example we might talk about the User cookbook for managing accounts on a node, or the MySQL cookbook for installing and manipulating a database programme.
- **Recipe** – An executable instruction within a cookbook. The workman of Chef: it consists of instructions written in Ruby on how to install, uninstall, enable, disable, create whatever resource it's managing.
- **Attribute** – A variable, or constant, on a node. The name of the operating system running on a node is a constant attribute we can't manipulate, but we could search on. Contrast this with an attribute required by a cookbook – we can change this and have the alteration applied to the node on the next run.

Set up a cloud server

Chef is capable of managing Windows boxes, but when most people talk about cloud hosting, what they mean are Linux cloud servers. What distinguishes cloud hosting from other types is usually its on-demand nature: cloud servers are available almost instantly and can be spun up and down as need dictates, often via an API layer that enables customers to automate yet another layer of their infrastructure. Chef includes tools for automatically provisioning servers with both EC2 and Rackspace, but in order to do this you need a workstation, which we've yet to set up. To make things a bit easier to follow here, we'll spin up a server manually before deploying Chef onto it.

In the instructions below, I'm setting up a Rackspace cloud account with a server, which will cost about £0.02 per hour to run. If you already have an Amazon EC2 account, or even a local VMware/VirtualBox set-up, that's fine to use too, and I'll assume you can jump forward to the next section. If you're not familiar with the process of setting up a server, read on.

Visit www.rackspace.com and open a **Cloud Servers** account, selecting the unmanaged option. You'll need to provide credit card details here, but we'll only be running a small server for a few hours, so the total cost should be less than 20 pence. Once your account is created and approved, sign into the **Cloud Control** panel and **Add Server** from within the **Cloud Server** section of the control panel. Select **Ubuntu 12.04 LTS (Precise Pangolin)** as your **Distro**, followed by **256MB** as the server's memory size. Give your server a name and press **Create**. You'll be billed two pence per hour while the server is running, so remember to come back to the panel and press **Destroy** when you're done.

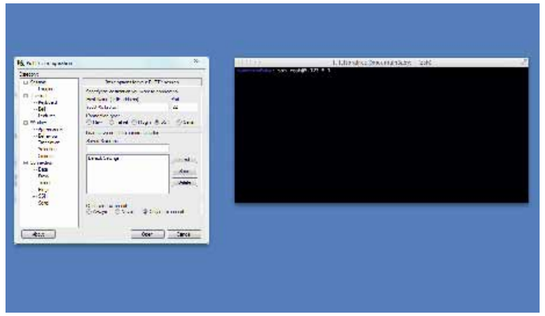
While your server is being provisioned, you'll be told the root password and the IP address. You'll need these details to sign in to your server, so make a note of them and go make a cup of tea. When the server becomes active you'll be able to connect to it via SSH and control it via the command line interface.

If you're on OS X or Linux you'll use your Terminal emulator with the IP address from earlier to connect as root, entering the password when prompted.

```
$ ssh root@<ip-address-here>
```

Unfortunately, Windows doesn't come with an SSH client, so you'll need to download PuTTY (http://netm.ag/putty-237) and use that instead. The default settings PuTTY provides will be fine. Just enter the username **root** along with the server IP address and password you recorded above.

Normally, your local machine would serve as the workstation: the machine from which you issue instructions to your empire of client servers (or 'nodes' in Chef parlance: see Glossary above). But to make things a bit simpler for Windows users, we'll use our newly provisioned server as both workstation and node. From here on, all the commands are to be executed on the remote server.



Connecting to your server To connect to our new remote server we'll use PuTTY on Windows (left) or a Terminal Emulator on Linux/OS X (right)

Installing Chef

Installing Chef is a trivial operation with the Opscode Omnibus installer (www.opscode.com/chef/install). We can jump straight to the final command, which you'll enter at the prompt:

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

The install will only take a few minutes, after which you can enter:

```
$ chef-client -v
```

This will return the version number and confirm that the install was successful. To get set up as a workstation just run the following commands.

```
$ apt-get install git;
$ git clone git://github.com/opscode/chef-repo.git workstation;
$ cd workstation && mkdir .chef;
$ export EDITOR=nano;
```

At this point it's worth discussing the architecture of Chef. Under normal circumstances, our workstation would be our office computer, and on that we would write cookbooks and issue instructions. However, your workstation might not be accessible to your nodes because of local network conditions – or simply because it's not always turned on. We need a second server through which our nodes can communicate with our workstation. This is the Chef Server.

It's possible to download the Chef Server application for free and set it up yourself, but that's beyond the scope of this tutorial.

>>

Using the command line

Familiarity with the Unix command line interface (CLI) is essential knowledge for web developers. The web runs on Unix, and Unix is best understood, controlled and developed from the CLI. Sure, it's possible to build and maintain websites without understanding it, but only in the same way that it's possible to drive on the motorway without leaving first gear.

From the perspective of a sysadmin maintaining dozens of servers, the CLI interface is the only way to go. When controlling servers, the command line is far superior to GUIs. The everything-is-text Unix philosophy means that aggregation of data from multiple sources is far easier than trying to follow what's going on in individual GUIs. CLI interfaces take far less bandwidth and are therefore ideal for managing servers located thousands of miles away. These days a sysadmin needs only a decent 3G connection to control servers from a CLI on their phone, where a GUI display would just not be practical.

There's a huge number of tools for controlling almost every facet of a computer, and almost no limit to what you can do on the command line.

The CLI can seem a strange, retro world, but it's a relatively small set of concepts and simple commands that build into a powerful set of tools. If you're running Windows, I'd suggest downloading Ubuntu and running it from a DVD or pen drive to get a feel for what it can do. If you're not familiar with the commands in this tutorial, I highly recommend this: (http://linuxcommand.org/learning_the_shell.php). Below, I've outlined some of basic commands we will be using in this tutorial:

ssh <servername> – A means of connecting securely between machines
cd <target_dir> – Change Directory: the command line equivalent of opening a folder in your GUI
pwd – Print Working Directory. If you ever lose your place in the file tree, pwd will show you where you currently sit
nano <filename> – A simple, but powerful, text editor. We'll use it in this tutorial to edit files on our remote server

>> Luckily, Opscode provides a hosted Chef Server product that's ideal for us. Sign up for a free account at www.opscode.com/hosted-chef. Make a note of your account name because you'll use it later.

Once you've created your account, you'll have access to the Chef web app. This provides a more user-friendly way of browsing and editing your nodes. It's looking pretty sparse just now, because we've not registered any nodes or uploaded any cookbooks. At present, we just need to download some credentials and a configuration file that will allow our Cloud server to talk to our new Chef Server.

First, visit the **Organizations** page and select **join**, then choose **Regenerate validation key** and then **Generate knife config**, saving both files to your local machine. Next, click on your username and then choose **view profile** to see your community Chef page. From here, select **get private key** and save the file.

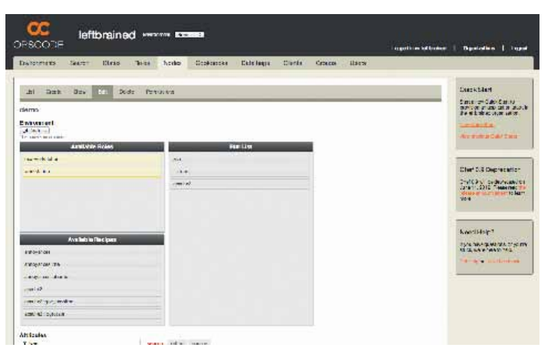
These three files all need to get into the .chef (note the leading dot) directory on your remote workstation. If you are on OS X or Linux, simply copy the files into place by SCP-ing from your local machine.

```
$ scp <path_to_file> root@<ip_address>:workstation/.chef/
```

If you're on Windows, it'll be easier for you to just copy the contents of each file onto your clipboard from Notepad, and then remotely create a new file with the same name and paste the contents in. For example, my Opscode account is called **net-demo**, so I can run the following command to open a text editor in the correct place:

```
$ nano ~/workstation/.chef/net-demo-validator.pem
```

I would then paste in the contents of contents of my local net-demo-validator.pem file and save from nano with Ctrl-X. Repeat this for the validator key, account key and knife.rb.



The Chef web app The Chef web app, available at http://manage.opscode.com, provides a user-friendly interface for seeing your nodes and making alterations to them

The final step of the set-up is to register our Cloud Server with our Chef Server as a node.

```
$ knife bootstrap localhost -n demo
```

With this done you should now see a node called **demo** appear in your Chef web app, Alternatively, issue the command below to see a list of all the nodes your Chef Server knows about:

```
$ knife node list
```

Let's get cooking!

The basic unit of Chef is the recipe. A recipe instructs a node to take an action: install software, create a user, run a command and so forth. Recipes are gathered together into collections called cookbooks. For example, the apache2 cookbook

The basic unit of Chef is the recipe. A recipe instructs a node to take an action

contains recipes for installing Apache, creating virtual hosts and stopping or starting the web server.

The first thing we need to be doing here is download some cookbooks to our workstation...

```
$ knife cookbook site install fortune;
$ knife cookbook site install apache2;
```

This will bring down the two cookbooks from the Opscode community site, which is a repository of shared resources from Chef's users (http://community.opscode.com). Although you can explore the cookbooks on the workstation, you may find it easier to visit the community in your browser and manually download copies to your local machine.

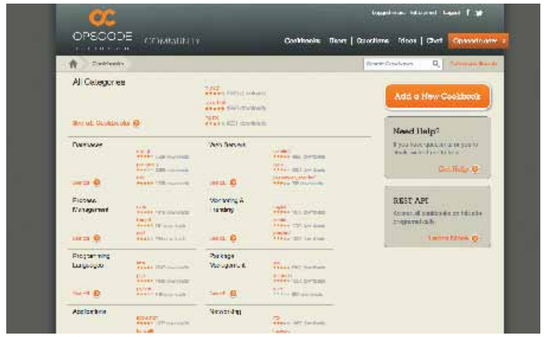
Next, we will be needing to let our Chef Server know all about our newly acquired cookbooks.

```
$ knife cookbook upload –all
```

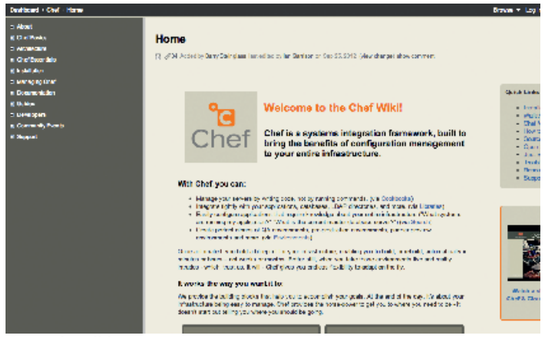
We can now use Chef to install the fortune program onto our server to provide us with quips and quotes.

```
$ knife node run_list add <node-name> “recipe[fortune]”
```

The **run_list** is a node's to-do list of operations to be carried out the next time it checks in. In this case run the fortune recipe, which will install the program.



Community site Chef cookbooks are easily shared. The http://community.opscode.com repository serves as a central database of user-submitted cookbooks



Opscode wiki Information on Chef can be found on the Opscode wiki at http://wiki.opscode.com. It's very developer-focused, but contains detailed notes on how Chef works

We can confirm what's planned for our node by using Knife.

```
$ knife node show <node-name>
```

Alternatively, if you visit the Chef Server in your browser and drill down to your node, you'll see that both the available cookbooks and current **run_list** are available as a drag-and-drop interface.

```
$ chef-client
```

The Chef Client will take a few minutes to complete. In this time our node contacts the Chef Server and syncs its attributes, run_lists, cookbooks and so on for any changes that have occurred since the last time it checked in. Once this is done, the node will work out what the end state of its **run_list** should be (in other words, fortune should be installed) and take whatever steps are required to bring itself into line. In a production set-up, Chef Client is normally set to run automatically on a regular schedule or is triggered with a tool such as mcollective. All being well...

```
$ fortune
```

...will print out some wise words. That's a very simple example of installing a single package with Chef.

Using attributes

Let's try something a bit more complex. We will now set up an Apache server.

```
$ knife node run_list add “recipe[apache2]”
$ chef-client.
```

On this run you should see Apache being installed and restarted. Also, notice what didn't happen: the node checked to see if fortune was already installed and just skipped it.

In your browser, visit your server's IP and you should now see the default Apache page: Chef has set up the web server for us.

Let's mix things up a bit. Imagine our Apache server being actually one of a dozen in our site's set-up, and word coming down that only chumps are running their websites on port 80 these days. The real hotness is on 8080! In a manually managed world, a poor sysadmin would have to make a list of all their web servers, shell onto each one, edit the Apache config, restart the server, log out, repeat. With Chef we can remove the grunt work.

Download the apache2 cookbook to your local machine and look in file attributes/default.rb. Most cookbooks provide a set of default values that get injected into the recipe at runtime. These variables can be overridden dynamically to customise behaviour. For example, we might have our firewalls open port 25 on our mail server, but 3306 on our database.

To change the port information, go back to the command line and use Knife to edit your node's attributes.

```
$ knife node edit <name>
```

Knife will pull down a JSON representation of your node's attributes and open it up in the editor. If you've done any JavaScript before, the formatting should be familiar to you. At the top of the file we have some information about the node itself, and at the bottom we have a copy of the **run_list** (which we could edit from here). In the middle of the file you'll find a pre-seeded section of Apache settings. Let's add in our new port number at the bottom of the block.

```
...
  “default_site_enabled” : false,
  “listen_ports”: [ “8080” ]
}
...
```

Save the file with Ctrl-X and Knife will push your changes back to the server. If you have any syntax errors in your file, Knife will noisily complain: JSON is quite strict in its layout. If you're not comfortable with this amount of CLI editing, the Chef Server provides a friendlier editor under **Node > Edit**, which will help you construct the JSON.

Rerun Chef Client and try visiting your IP again in the browser. Nothing? Try adding :8080 to the URL. You should now be looking at the default page of your Apache server. This time around Chef knew it didn't need to install Apache, but it did update our configuration with the new information and restart the web server.

In a production set-up, Chef provides a level of grouping above the node called a role, which is essentially a collection of attributes and recipes that can be applied to a whole batch of nodes – example roles might be WebServer, MailServer, LampServer and so forth. With our servers all grouped under one role, the process of updating any number of servers is reduced to editing that role and triggering Chef Client.

Conclusion

This article is merely a quick introduction to Chef, and we've hardly even scratched the surface of all the many and wondrous things that it can do. With the set-up and key concepts under your belt, it becomes possible to start learning about how to automate your site's infrastructure.

If you'd like to learn more about Chef, I highly recommend spending a little time with the Opscode Wiki (http://wiki.opscode.com), Joshua Timberman's Blog (http://jtimberman.housepub.org) and the DevOps Weekly Newsletter (www.devopsweekly.com). ●

About the author

Name Gavin Montague
URL www.leftbrained.co.uk
Twitter @gavinmontague
Areas of expertise Ruby, usability
Clients? itison
If you could travel back in time, what would you do?
xxxxxxxxxxxxx