Summer 14

Vagrant, chef-solo and a Django Project

Ramon Gallart

This is a novice-level tutorial-like document. Its goal is to serve as a first entry-point to create a re-usable machine to develop Django projects, so that it can be spun up in minutes by any developer team member in your team. Or whoever you want.

Table of Contents

Main Goal

This is a very straightforward step-by-step tutorial walkthrough that will guide you in the process of spinning up a development machine to work with a Django project using Virtualbox, Vagrant and Chef-solo. The Django project is the Polls app from the https://docs.djangoproject.com/en/1.6/ tutorial.

We will not explain here how to develop a Django project. Just our experience on what do you need to use Chef-solo with a Django project.

What do we need

Install Virtualbox and Vagrant if you haven't already done so.

Virtualbox allows us to install VMs to our system. Vagrant acts as a wrapper to Virtualbox (and many others) and allows us to configure the VMs using a Ruby script named Vagrantfile in an easy way. It should be easy to locate and install a version for your OS.

Install Chef-DK. This will install Chef (but not Chef-solo) along commonly used commands (knife, berks…)

Install Bundler. Bundler allows us to install Ruby gems and its dependencies on our system:

```
$ sudo gem install bundler
```

Install chef plugin, berkshelf plugin and vagrant-omnibus too for Vagrant if they are not installed. They can take a bit of a long time:

```
$ vagrant plugin install chef
$ vagrant plugin install vagrant-berkshelf --plugin-version 2.0.1
$ vagrant plugin install vagrant-omnibus
```

The chef plugin will allow us to use some Chef modules from inside the Vagrantfile.

Vagrant-berkshelf will run Berkshelf automatically each time the Vagrantfile is loaded, thus downloading all the cookbooks defined in the Berksfile.

Vagrant-omnibus, installs the chef-client inside he VM and runs it once the VM loads up.

Getting your hands dirty

Create a directory for the project:

```
$ mkdir ~/projects/django-con
$ cd django-con
```

Initialize a Vagrant machine:

```
$ vagrant init precise64
```

Resulting file (comments omitted)

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
 config.vm.box = "precise64"
 config.omnibus.chef_version = :latest
 config.berkshelf.enabled = true
end
```

Create a file named Gemfile:

```
$ cd ~/projects/django-con
$ vim Gemfile
```

And write this content in it:

```
source "https://rubygems.org"
gem 'knife-solo'
```

This will instruct bundle to install the open source tool knife-solo.

```
$ cd ~/projects/django-con
$ bundle install
```

At this point we will have the chef-solo gem installed. chef-solo will create a base infrastructure to develop our own cookbooks or download already created ones.

So, in order to continue creating our environment we need to initialize the kitchen!

```
$ cd ~/projects/django-con
$ knife solo init .
```

Once this command has ran, we have the following directory infrastructure:

```
ramonmariagallart@Olympos $ ll -a
total 32
drwxr-xr-x  14 ramonmariagallart  staff   476B 21 ago 16:36 ./
drwxr-xr-x  11 ramonmariagallart  staff   374B 21 ago 16:14 ../
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 .chef/
-rw-r--r--   1 ramonmariagallart  staff    12B 21 ago 16:36 .gitignore
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:19 .vagrant/
-rw-r--r--   1 ramonmariagallart  staff    48B 21 ago 16:32 Gemfile
-rw-r--r--   1 ramonmariagallart  staff   1,8K 21 ago 16:36 Gemfile.lock
-rw-r--r--   1 ramonmariagallart  staff   238B 21 ago 16:25 Vagrantfile
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 cookbooks/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 data_bags/
```

```
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 environments/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 nodes/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 roles/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 16:36 site-cookbooks/
```

A detailed description of every directory is out of this HOWTO scope. Just say that:

- .chef/: has the file knife.rb which will have default constants that we will use in the Vagrantfile
- cookbooks/: contains third-party cookbooks that will be downloaded by Berkshelf
- data_bags/ environments/ nodes/ roles/: JSON file definitions for each one of those
- site-cookbooks/: this will keep our own cookbooks
- .gitignore: at creation time it only has /cookbooks/ as we are not interested in keeping them in a CVS

Following we create a Berksfile. In that file we will enumerate the cookbooks we want to use on our environment. Berkshelf will take into account the different dependencies existing between the cookbooks we need and automatically downloading those that are necessary:

```
$ cd ~/projects/django-con
$ vim Berksfile
```

Arguably the best resource to find cookbooks is the Chef supermarket web (https://supermarket.getchef.com/) as we can be pretty sure that the cookbooks held in there are regularly updated and mantained. Nevertheless there are other resources where we can find very good cookbooks (if not better). One of those resources is Github (or for what matters, any git repository). Luckily enough, the Berksfile file format allows us to download cookbooks from the supermarket repository and from git repositories and even from local paths.

That said, lets write this into the file:

```
source "https://supermarket.getchef.com/"
cookbook 'apt', '~> 2.5.2'
cookbook 'nginx', '~> 2.7.4'
cookbook "postgresql", git: 'https://github.com/phlipper/chef-postgresql.git'
cookbook 'python', '~> 1.4.6'
cookbook 'supervisor', '~> 0.4.12'
```

If we start the environment up now we will see that none of these cookbooks are getting installed. This is because we have to tell Vagrant (specifically indicate to it inside the Vagrantfile) which cookbooks must be installed.

To that end we will modify our Vagrantfile until it looks something like this (in **bold** are the lines added):

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
require 'chef'
```

```
require 'json'
Chef::Config.from_file(File.join(File.dirname(__FILE__), '.chef', 'knife.rb'))
vagrant_json = JSON.parse(Pathname(__FILE__).dirname.join('nodes', (ENV['NODE'] ||
'polls.example.com.json')).read)
environment_node = (ENV['NODE'] || 'polls.example.com.json')
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
 config.vm.box = "precise64"
 config.vm.network :forwarded_port, guest: 80, host: 8080 if environment_node ==
'staging-polls.example.com.json'
 config.vm.network :forwarded_port, guest: 8000, host: 8000 if environment_node ==
'polls.example.com.json'
 config.omnibus.chef_version = :latest
 config.berkshelf.enabled = true
 config.vm.provision "chef_solo" do |chef|
   chef.roles_path = Chef::Config["role_path"]
   chef.data_bags_path = Chef::Config["data_bag_path"]
   chef.environments_path = Chef::Config["environment_path"]
   chef.environment = ENV[‘ENVIRONMENT’] || ‘development’
   chef.run_list = vagrant_json.delete('run_list')
   chef.json = vagrant_json
 end
end
```

As said before, Vagrantfile is a Ruby script and we can take advantage of that. We set up the Config object from the Chef module to include the data that is inside the file ./.chef/knife.rb.

We load too a node named ‘polls.example.com’ from which we still have not talked about. Next we forward the our host’s ports 80 and 8000 to our guest machine’s port 8080 and 8000 respectively. But we do that depending on what node will be used. By default, in the case of no defining any node, the development node is used. When spinning up the vagrant machine we can load up the development or the staging machine just switching between those two commands:

```
$ NODE=polls.example.com.json vagrant up # spins up the development machine
```

or

```
$ NODE=staging-polls.example.com.json vagrant up # spins up the staging machine
```

Finally we configure Vagrant’s chef_solo provisioner indicating where it can find different resources.

Nodes

Nodes in Chef refer to different machines in our environment. We can configure different environments too. For that HOWTO[1] we named our main development environment as 'polls.example.com' and we have to create a file named 'development.json' in the environments/ directory and another one named 'polls.example.com.json' inside our nodes/ directory:

```
$ cd ~/projects/django-con/environments/
$ vim development.json
```

This is the content for the file:

```
{
   "name": "development",
   "description": "development environment",
   "chef_type": "environment",
   "json_class": "Chef::Environment"
}
```

One of the parameters that this file may contain is "default_attributes" which we can leverage in order to modify some attributes of the downloaded cookbooks.

```
$ cd ~/projects/django-con/nodes/
$ vim polls.example.com.json
```

Write in the following content:

```
{
   "environment": "development",
   "run_list": [
       "recipe[apt]",
       "recipe[vim]",
       "recipe[git]",
       "recipe[postgresql]",
       "recipe[postgresql::server]",
       "recipe[postgresql::client]",
       "recipe[postgresql::server_dev]",
       "recipe[python]"
   ]
}
```

As we can see, inside the file we define this node to belong to the development environment and a "run_list". This list contains the recipes we want to run for this node. **It is important to mention that recipes are executed in the same order found in the list**.

So the next step is to provision our environment with those recipes:

```
$ vagrant up  # if it is not already up
$ vagrant provision  # so chef will run and install all the recipes
```

[1] On the repo you can find another node file named staging-polls.example.com.json. It uses more recipes than polls.example.com.json and it is included as a demonstration on how to have more than one node for us to test different environments.

We can make a minimal check in order to know if everything went OK. The first thing to look at is the output produced by the vagrant provision command. No errors? Cool! Another thing we can try is vagrant ssh into the VM and execute psql. It should return an error saying there is no “vagrant” role. That’s fine for now, we have not created any user for Postgres, but it got installed! You can check further doing `sudo -u postgres psql`. You should see something like that:

```
psql (9.3.5)
Type "help" for help.
postgres=#
```

Hooray! The server got installed too!

Anyway, this will just install the software that we need in order to run our application. The next step is to create our own cookbook that will configure all of this software so that it will run as we intend it to do it.

Creating our own cookbook for fun and profit

The goal creating our own cookbook is to manage configuration for our environment the way we want it to be. This can be achieved mainly in two different ways:

- Create a new cookbook and use recipes, resources and attributes from other already downloaded cookboks
- Create a new cookbook and set your own scripts to do the things you need

The postgresql cookbook

Anyway, we need to create a new cookbook. So, to create it:

```
$ cd ~/projects/django-con/site-cookbooks/
$ berks cookbook poll-app
```

This will create poll_app cookbook with this directory structure:

```
ramonmariagallart@Olympos $ ll -a poll-app/
total 88
drwxr-xr-x  22 ramonmariagallart  staff   748B 21 ago 18:21 ./
drwxr-xr-x   4 ramonmariagallart  staff   136B 21 ago 18:21 ../
drwxr-xr-x  10 ramonmariagallart  staff   340B 21 ago 18:21 .git/
-rw-r--r--   1 ramonmariagallart  staff   155B 21 ago 18:21 .gitignore
-rw-r--r--   1 ramonmariagallart  staff   173B 21 ago 18:21 .kitchen.yml
-rw-r--r--   1 ramonmariagallart  staff    51B 21 ago 18:21 Berksfile
-rw-r--r--   1 ramonmariagallart  staff    99B 21 ago 18:21 CHANGELOG.md
-rw-r--r--   1 ramonmariagallart  staff   449B 21 ago 18:21 Gemfile
-rw-r--r--   1 ramonmariagallart  staff    72B 21 ago 18:21 LICENSE
-rw-r--r--   1 ramonmariagallart  staff   850B 21 ago 18:21 README.md
-rw-r--r--   1 ramonmariagallart  staff   241B 21 ago 18:21 Thorfile
-rw-r--r--   1 ramonmariagallart  staff   3,3K 21 ago 18:21 Vagrantfile
drwxr-xr-x   2 ramonmariagallart  staff    68B 21 ago 18:21 attributes/
-rw-r--r--   1 ramonmariagallart  staff   960B 21 ago 18:21 chefignore
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 18:21 files/
drwxr-xr-x   2 ramonmariagallart  staff    68B 21 ago 18:21 libraries/
-rw-r--r--   1 ramonmariagallart  staff   248B 21 ago 18:21 metadata.rb
drwxr-xr-x   2 ramonmariagallart  staff    68B 21 ago 18:21 providers/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 18:21 recipes/
drwxr-xr-x   2 ramonmariagallart  staff    68B 21 ago 18:21 resources/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 18:21 templates/
drwxr-xr-x   3 ramonmariagallart  staff   102B 21 ago 18:21 test/
```

The first thing we have to do is update the metadata.rb file which contains main information about the cookbook.

```
$ cd cd ~/projects/django-con/site-cookbooks/poll-app/
$ vim metadata.rb
```

And we can leave it like this:

```
name             'poll-app'
```

```
maintainer       'Ramon Maria Gallart'
maintainer_email 'rgallart@ramagaes.com'
license          'MIT'
description      'Installs/Configures poll-app'
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version          '0.1.0'
```

Self-explanatory. For the long-description we will insert the contents of the README.md file contained in the same directory as metadata.rb.

OK. So now it's time to use some of the other downloaded recipes the way we want. For example, we would like to create a user and a database for our PostgreSQL. How can we do that?

First of all, we do have to add our recently created cookbook to our run_list for the node we want it to be ran:

```
# nodes/polls.example.com.json
…
       "recipe[python]",
       "recipe[supervisor]",
       "recipe[poll-app]"
   ]
}
```

But this is the first step of two. We have to modify our Berksfile like this:

```
# ./Berksfile
…
postgresql.git'
cookbook 'python', '~> 1.4.6'
cookbook 'supervisor', '~> 0.4.12'
cookbook 'poll-app', path: "./site-cookbooks/poll-app"
```

Doing that we ensure ourselves that Chef will be able to find our cookbook.

After that two-step procedure, if we want our cookbook to use some of the other already downloaded cookbooks we have to tell it we want to do so. First, in the metadata.rb file in our cookbook we include a line like this:

```
# metadata.rb
...
depends “postgres”
```

And then, in our recipe.rb file:

```
# recipes/default.rb
include_recipe “postgres”
# CREATE POSTGRESQL USER
pg_user "polluser" do
 privileges superuser: false, createdb: false, login: true
 password "polluserpwd"
end
# CREATE POSTGRES DB
pg_database "polldb" do
```

```
 owner "polluser"
 encoding "UTF-8"
 template "template0"
 locale "en_US.UTF-8"
end
```

With that script we will create a polluser on our database with login permissions and a polldb database whose owner is the polluser previously created.

Run vagrant provision or vagrant up --provision followed by vagrant ssh and check that the user and the database have been correctly created:

```
$ psql -d polldb -Upolluser -W
Password for user polluser:
psql (9.3.5)
Type "help" for help.
polldb=>
```

Cool! We can see that the user has been created as well as the database!

The python cookbook

So, what's our next step? If we want to develop a Django app it is common to use tools like pip and virtualenv. The previous python cookbook allows us to install and use those two pieces of software. Lets create a new cookbook to manage that:

```
$ cd ~/projects/django-con/site-cookbooks/
$ berks cookbook poll-app-python
```

In the recipes/metadata.rb we will write the following code:

```
package "python-psycopg2"
include_recipe "python"
# CREATE A VIRTUALENV
python_virtualenv "/home/vagrant/polls_ve" do
 owner "vagrant"
 group "vagrant"
 action :create
end
# INSTALLING PYTHON STUFF
bash "install_requirements.txt" do
 user "vagrant"
 cwd "/vagrant/"
 code <<-EOH
 . /home/vagrant/polls_ve/bin/activate && pip install -r requirements/requirements.txt
 EOH
end
```

What we are instructing chef-solo to do here is that we want to install "python-psycopg2" if it is not already. After that we state that we will need recipes inside the "python" cookbook.

So, with that we create a virtualenv using the python_virtualenv LWRP from the python cookbook. The virtualenv gets created at /home/vagrant/polls_ve.

Next we use the bash resource provided by chef to install the requirements needed by the app. The bash resource allows us to run bash scripts from our recipes. To do that we need to state under which user the script will be ran, what's the working directory and the command to run.

As the poll-app cookbook we have to declare our new recipe to the Berksfile and add it to the polls.example.com.json node file:

```
# nodes/polls.example.com.json
…
        "recipe[python]",
        "recipe[supervisor]",
        "recipe[poll-app]",
        "recipe[poll-app-python]"
    ]
}
```

But, remember, this is the first step of two. We have to modify our Berksfile like this:

```
# ./Berksfile
…
postgresql.git'
cookbook 'python', '~> 1.4.6'
cookbook 'supervisor', '~> 0.4.12'
cookbook 'poll-app', path: "./site-cookbooks/poll-app"
cookbook 'poll-app', path: "./site-cookbooks/poll-app-python"
```

Lets check if that works! As before, run first vagrant provision or vagrant up --provision and then vagrant ssh.

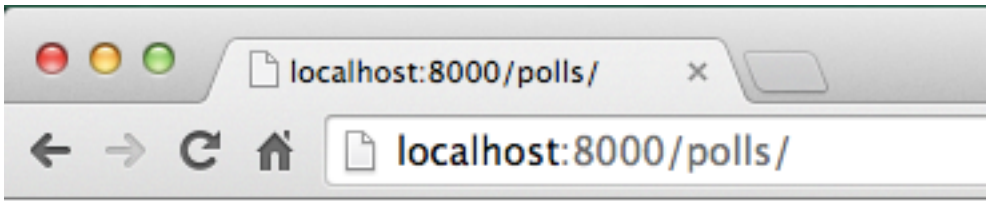
```
vagrant@precise64:~$ pwd
/home/vagrant
vagrant@precise64:~$ . polls_ve/bin/activate
(polls_ve)vagrant@precise64:~$ cd /vagrant/src/
(polls_ve)vagrant@precise64:/vagrant/src$ ./manage.py shell
Python 2.7.3 (default, Feb 27 2014, 19:58:35)
Type "copyright", "credits" or "license" for more information.
IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
In [1]: import django
In [2]: django.VERSION
Out[2]: (1, 6, 6, 'final', 0)
In [3]: quit
(polls_ve)vagrant@precise64:/vagrant/src$
```

Great! All the requirements have been installed and seems like we are now ready to work with the app running it from the VM and editing from our preferred editor in our host machine!

Lets start the runserver:

```
(polls_ve)vagrant@precise64:/vagrant/src$ ./manage.py runserver 0.0.0.0:8000
Validating models...
0 errors found
September 01, 2014 - 11:54:26
Django version 1.6.6, using settings 'mysite.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
[01/Sep/2014 11:54:46] "GET /polls HTTP/1.1" 301 0
[01/Sep/2014 11:54:46] "GET /polls/ HTTP/1.1" 200 173
[01/Sep/2014 11:54:46] "GET /static/polls/style.css HTTP/1.1" 200 27
```

And, effectively, we can see the iconic Polls app on our browsers:



- What's up?

Things left behind

Data bags

Resources and providers

Cookbook testing

Resources

Info and tutorials about chef and chef-solo

http://chef.leopard.in.ua/

http://blog.smalleycreative.com/tutorials/setup-a-django-vm-with-vagrant-virtualbox-and-chef/

http://tumblr.nrako.com/post/22320729770/vagrant-chef-librarian

http://stackoverflow.com/questions/11325479/how-to-control-the-version-of-chef-that-vagrant-uses-to-provision-vms

http://www.getchef.com/blog/2013/12/03/doing-wrapper-cookbooks-right/

Tools

Vagrant

Virtualbox

ChefDK

Git

Berkshelf

Chef Supermarket