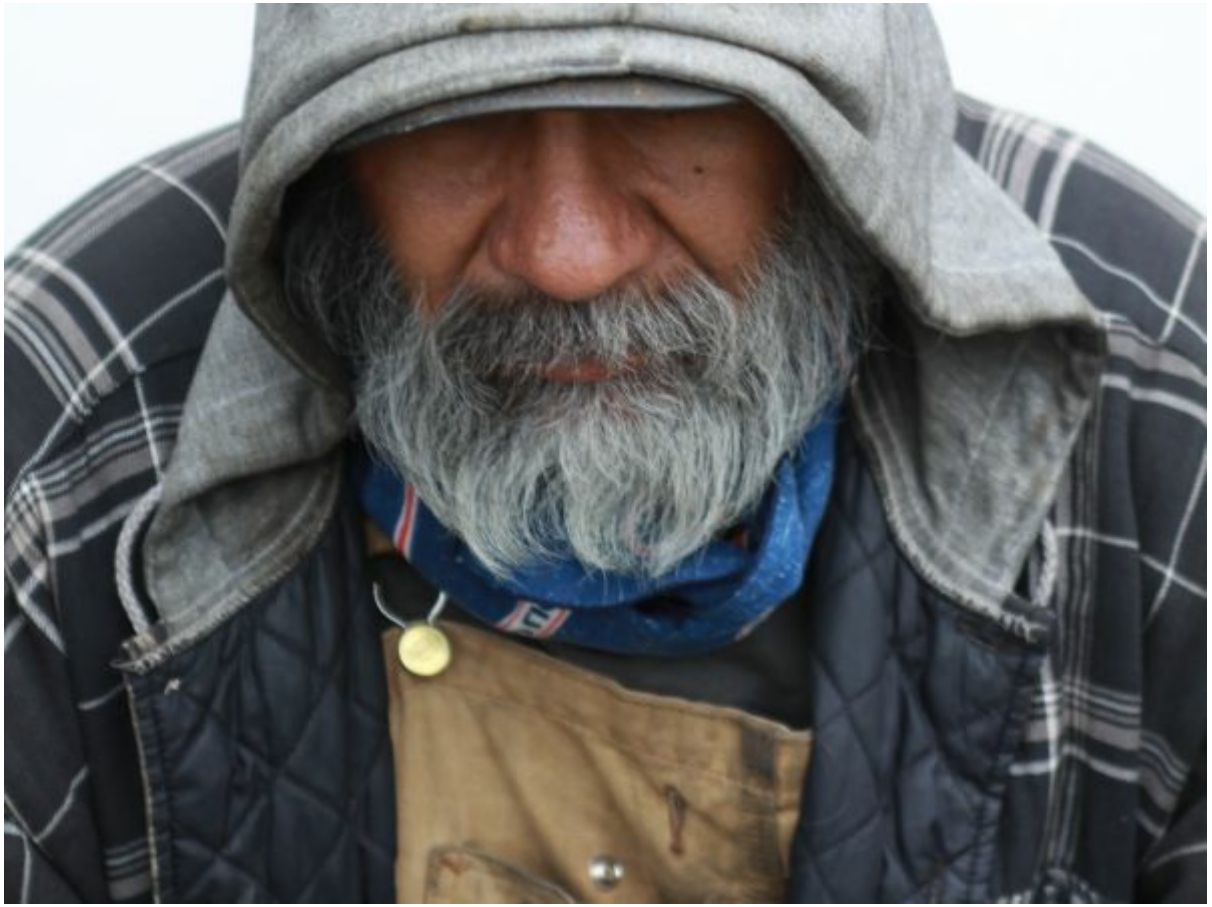
Boosting developer's productivity: Using Vagrant and Chef

- Author: **Alexander Shvets**
- Year: **2013**

Why do we need virtualiation in development?

- We want to have **same environment for all developers**, no matter on what platform they are working now.
- We are **working on multiple projects** on same workstation. As a result, suddenly your computer has "hidden", hard-to-discover inter-project dependencies or different versions of same library.
- To overcome **It "works on my machine!"** syndrome - development environment is different from production environment.
- Sometimes required software is **not available** on developer's platform. Example: 64-bit instant client for oracle was broken for almost two years on OS/X >= 10.7.
- **Development for PAAS**, such as Heroku, Engine Yard etc. You can find/build virtualization that is pretty close to your platform.

Vagrant



What is it?

- It is ruby gem.
- it helps to provide easy to configure, reproducible, and portable work environments.

Installation

- Install VirtualBox. Download it from dedicated web site and install it as native program. You can use it in UI mode, but it’s not required.
- Install Vagrant gem:

```
$ gem install vagrant
```

- or put it into **Gemfile**:

```
gem "vagrant"
```

- Run bundler:

```
bundle install
```

Playing with vagrant

- Install linux image

```
$ vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

- Initialize

```
$ vagrant init precise64
```

- Run or reload

```
$ vagrant up
$ vagrant reload
$ vagrant provision
```

Playing with vagrant (continued)

- Suspend/destroy:

```
$ vagrant suspend
$ vagrant destroy
```

- or remove linux image

```
$ vagrant box remove precise64
```

Access VM

```
$ vagrant ssh

vagrant> ls /vagrant
```

- You will see all project's files

```
drwxr-xr-x  48 alex  staff   1.6K Apr 21 15:29 ./
drwxrwxrwx  24 alex  staff   816B Apr 21 15:31 ../
-rw-rw-rw-   1 alex  staff   101B Apr 19 10:11 Berksfile
-rw-rw-rw-   1 alex  staff   512B Apr 19 09:57 Berksfile.lock
-rw-rw-rw-   1 alex  staff   188B Apr 17 11:40 Cheffile
-rw-rw-rw-   1 alex  staff   715B Apr 16 11:39 Cheffile.lock
-rw-rw-rw-   1 alex  staff   4.4K Apr 21 14:20 Gemfile
-rw-r--r--   1 alex  staff   9.7K Apr 21 14:20 Gemfile.lock
-rw-r--r--   1 alex  staff   106B Mar 23 13:14 Procfile
-rw-r--r--   1 alex  staff   297B Mar 23 13:14 README.md
-rw-r--r--   1 alex  staff   5.8K Apr 20 11:54 Rakefile
-rw-rw-rw-   1 alex  staff   5.1K Apr 17 15:19 Vagrantfile
drwxr-xr-x   9 alex  staff   306B Mar 24 13:37 app/
-rwxrwxrwx@  1 alex  staff   3.1K Apr 18 09:56 bootstrap.sh*
drwxr-xr-x   8 alex  staff   272B Apr 20 07:54 public/
```

Available provisioners for Vagrant

- Shell
- Chef Server
- Chef Solo
- Puppet Server/Standalone

Example 1

- **Vagrantfile** with **bash** shell provisioning:

```
Vagrant::Config.run do |config|
  config.vm.box = "precise64"

  config.vm.network :hostonly, "22.22.22.22"

  config.vm.provision :shell, :path => "bootstrap.sh"
end
```

Example 2

- **Vagrantfile** with **chef** provisioning

```
Vagrant::Config.run do |config|
  config.vm.box = "precise64"
  config.vm.network :hostonly, "22.22.22.22"

  config.vm.provision :chef_solo do |chef|
    chef.cookbooks_path = ["vendored_cookbooks", "cookbooks"]

    chef.add_recipe 'application'

    chef.json.merge!({
      application: {app_name: "triton"},

      rvm: {ruby: {version: '1.9.3', implementation: 'ruby',
            patch_level: 'p392'}},
      postgresql: { username: "postgres", password: "postgres"},
      mysql: { server_debian_password: "root",
               server_root_password: "root",
               server_repl_password: "root"}
    })
  end
end
```

Example 3

- **Vagrantfile** with multiple environments:

```
Vagrant::Config.run do |config|
  config.vm.define :node1 do |node_config|
    node_config.vm.box = "lucid32"
  end
  config.vm.define :node2 do |node_config|
    node_config.vm.box = "precise32"
  end
end
```

- Start first node in one tab:

```
vagrant up node1
vagrant ssh node1
```

- and second node in another tab:

```
vagrant up node2
vagrant ssh node2
```

Packaging

- You can create new package based on your vagrant setup.
- This new package can be used by other developers for faster installation.
- It will have already preinstalled and configured ruby/rvm/mysql/postgres etc.:

```
vagrant package --vagrantfile Vagrantfile --output proteus.box
```

What's next?

- Once you install Vagrant with some provision (script, chef-solo or puppet), you can use it in same way as your worksation:

```
vagrant ssh

cd /vagrant

rake db:dev:reset

rspec

ASSET_HOST=http://22.22.22.22:3000 rails s
```

- and then access it from the browser within host computer:

```
open http://22.22.22.22:3000/triton
```

Example of shell script for provisioning:

```
# Install core packages

apt-get update
sudo apt-get install -y curl
sudo apt-get install -y g++
sudo apt-get install -y subversion

# Install RVM

\curl -L https://get.rvm.io | bash -s stable --ruby=$RUBY_VERSION
source /usr/local/rvm/scripts/rvm
chown -R vagrant /usr/local/rvm
/usr/local/rvm/bin/rvm autolibs enable

# Install databases

sudo apt-get install -y postgresql-client
sudo apt-get install -y postgresql
sudo apt-get install -y mysql-client
apt-get -y install mysql-server

cd $APP_HOME
bundle install --without=production
rake db:migrate
```

Example of chef script for provisioning

```
# cookbooks/application/recipes/default.rb

include_recipe "apt"

package "curl"
package "g++"
package "subversion"

include_recipe "rvm"

bash "Registering RVM" do
  code "source /etc/profile.d/rvm.sh"
  not_if "test -e /usr/local/rvm/bin/rvm"
end

bash "Configuring RVM" do
  user "root"

  code <<-CODE
    chown -R vagrant /usr/local/rvm
    /usr/local/rvm/bin/rvm autolibs enable
  CODE
end
include_recipe "rvm::install"
```

Example of chef script for provisioning (continued)

```
include_recipe "postgresql"
include_recipe "postgresql::server"
include_recipe "mysql"
include_recipe "mysql::server"

bash "Installing bundle" do
  user 'vagrant'
  cwd app_home

  code <<-CODE
source /etc/profile.d/rvm.sh
rvm use #{ruby_version}@#{gem_name} --create
bundle install --without=production
 CODE
end

bash "Project db bootstrap" do
  user 'vagrant'
  cwd app_home

  code <<-CODE
    source /etc/profile.d/rvm.sh
    rake db:migrate
  CODE
end
```

Provisioning

Use Cases

- Provide **clean clone of production environment**. You can guarantee that your staging server is a exact clone of your production server.
- **Team of developers working along the globe and on different platforms**. The idea is to do development against unified, easy to replicate virtual platform with same characteristics for all developers.
- **Configure new workstation for new employee**, e.g. macbook configuration (rvm, ruby, mysql, postgesql, skype, iterm2 etc.)

Available options for provisioning

- Create simple shell script
- Use chef gem
- Use puppet gem (out of scope)
- Provisioning from Github: Boxen - tool for automating and managing Macs with Puppet

Chef


ACA
Director

Chef - what is it?

Chef is like unit tests for your servers

Ezra Zygmuntowicz

- It is **domain specific language** based on ruby.
- You can write **installation scripts** that will install all required for your platform packages (like homebrew or macport).
- Could be executed **locally** or on **remote** server.

Types of Chef components

- **Chef Server** - used to distribute cookbooks, manage and authenticate nodes and query infrastructure information on multiple nodes.
- **Chef Client** - command line tool for interacting with Chef Server.
- **Chef Solo** - you can run cookbooks in the absence of a Chef Server on single node. You need to have cookbooks somewhere. Only Chef Solo will be used in this presentation.

Common conceps of Chef

- **Node** - A host where the Chef client will be executed.
- **Recipe** - ruby program with code to install or configure software (install webserver or database, create database user etc.).

```
# cookbooks/application/default.rb

include_recipe "rvm" # include another recipe

package "rvm" # include OS package

bash "ls -al" # run OS command
```

Common concepts of Chef (continued)

- **Cookbook** - a collection of recipes (e.g. db cookbook could handle common interface for installing/configuring mysql and postgresql).

```
-cookbooks
  -application
    -attributes
      default.rb
    -recipes
      default.rb
      my_postgres.rb
      my_mysql.rb
    -templates
      Gemfile.erb
      mysql_database.yml.erb
```

Common concepts of Chef (continued)

- **Attribute** - default values to be used by cookbook in order to configure it (e.g. required version, service port or user name/password etc.). They can be redefined when you configure recipe.
- **Data Bags** - collection of properties that can be used together.
- **Metadata** - describes the recipes, dependencies, supported platforms, etc.

Common concepts of Chef (continued)

- **Resource** - it could be file, directory, package, service etc.

```
directory "/var/cache/local/preseeding" do
  owner "root"
  group node['mysql']['root_group']
  mode 0755
  recursive true
end
service "mysql" do
  action :start
end
dmg_package "GitOSX-Installer" do
  type "pkg"
  action :install
end
cron "noop" do
  hour "5"
  minute "0"
  command "/bin/true"
end
```

Common concepts of Chef (continued)

- **Template** - powered by ERB and used for generation configuration files.

```
template "#{node[:rails][:app_root]}/Gemfile" do
  source "Gemfile.erb"
  mode "0666"
  owner node[:rails][:user]
  group node[:rails][:group]
  variables({ :db_gem => "mysql" })
  not_if { `ls #{node[:rails][:app_root]}`.include?("Gemfile") }
end
```

- **Role** - reusable configuration for multiple nodes (mac user role, web role, database role, etc).
- **Run List** - list of recipes and roles to run.

Running

- You create **recipes run list** in order to run provision.

```
{
  "run_list": [ "recipe[application]" ]
}
```

- Chef runs a bunch of recipes to set up your server.
- You can run it with vagrant:

```
vagrant provision
vagrant reload
vagrant up
```

- or with chef-solo:

```
chef-solo -c your-solo-config-location.rb
          -j your-solo-node-location.json
```

Example of configuration file

```
project_root = File.dirname(File.expand_path(__FILE__))

cookbook_path [File.join(project_root, "..", "cookbooks"),
               File.join(project_root, "..", "vendored_cookbooks")]

#json_attribs File.join(project_root, "..", "nodes",
#   "vagrant-node.json")
```

Example of vagrant-node.json file

```
{
  "run_list": [ "recipe[application]" ],

  "application": { "app_name": "triton" },

  "rvm": {"ruby": {"version": "1.9.3",
                   "implementation": "ruby",
                   "patch_level": "p392"}},

  "postgresql": { "username": "postgres", "password": "postgres"},

  "mysql": { "server_debian_password": "root",
             "server_root_password": "root",
             "server_repl_password": "root"}
}
```

Chef-Solo: Standalone Installation

- If you want to use chef apart from vagrant, you have to install chef gem:

```
gem install chef
```

- and then use it:

```
ssh user@your_server
cd
sudo chef-solo -c /vagrant/config/solo.rb -c /vagrant/nodes/node.rb
```

- Important: do it from user home.

Common recipes

- Public cookbooks are located at opscode community home
- For easy manipulation with common recipes, install **librarian-chef** gem:

```
gem install librarian-chef
```

- Initialize librarian-chef. New **Cheffile** file will be created:

```
librarian-chef init
```

- Install cookbooks defined in Cheffile:

```
librarian-chef install --path vendored_cookbooks
```

- We keep them separately from our cookbooks. Our cookbooks are located in **cookbooks** folder.

Example of Cheffile

```
site 'http://community.opscode.com/api/v1'

cookbook 'apt'
cookbook 'git'

cookbook 'mysql'
cookbook 'postgresql'
```

- Other useful commands:

```
librarian-chef outdated
librarian-chef update
```

Tip 1: Skipping consecutive runs

- If you run it several times, Chef can skip all consecutive runs, thanks to **not_if** method call:

```
app_user = 'app_user'
postgres_user = 'postgres'
db_schema = 'myapp_dev'

bash "Creating postgres db user" do
  user "postgres"

  code <<-CODE
    psql -c "CREATE USER #{app_user} WITH PASSWORD '#{app_user}'"
  CODE

  not_if { `sudo sudo -u #{postgres_user} psql -c '\\l'`.
    include?(db_schema) }
end
```

Tip 2: Support for multiple platforms

- You can to different actions for different patforms if you want:

```
# cookbooks/some_recipe/recipes/default.rb

case node[:platform]
  when "ubuntu"
    package "package_for_ubuntu"
  when "centos"
    package "package_for_centos"
  when "mac_os_x"
    package "package_for_mac_os_x"
end
```

Tip 3: Integration with Capistrano

- Capistrano **Capfile** script

```
set :application, "vagrant"
set :host, "22.22.22.22"
set :user, "vagrant"
set :password, "vagrant"
role :vagrant, host

namespace :vagrant do

  desc "Execute chef run_list"
  task :run_list, :roles => :vagrant do
    cmd = "/opt/vagrant_ruby/bin/chef-solo -c /vagrant/config/solo.rb
    -j /vagrant/nodes/vagrant-node.json"

    run %[sudo #{cmd}]
  end
end
```

- Now you can run this task:

```
cap vagrant:run_list
```

Links

- VirtualBox - *https://www.virtualbox.org/wiki/Downloads*
- Vagrant - *http://www.vagrantup.com*
- Chef - *http://www.opscode.com*
- Chef Wiki - *http://wiki.opscode.com/display/chef/Home*
- Chef Public Cookbooks - *https://github.com/opscode-cookbooks*
- Setup multiple servers easily with Chef - *http://www.synbioz.com/blog/setup_multiple_servers_easily_with_chef*

Links (continued)

- OS X Workstation Management With Chef - *http://jtimberman.housepub.org/blog/2012/07/29/os-x-workstation-management-with-chef*
- Puppet - *http://puppetlabs.com*
- Getting Started with Chef Server - *http://leopard.in.ua*
- Vagrant+Puppet presentation - *https://github.com/crohr/vagrant-presentation*

- Thank You!
- Hola!
- Спасибо!
- 谢谢!

- Toda!

- Questions?
- Suggestions?