Extracted from:

Deploying with JRuby

Deliver Scalable Web Apps Using the JVM

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Deploying with JRuby

Deliver Scalable Web Apps Using the JVM

Joe Kutner

Edited by Brian P. Hogan

The Facets of Ruby Series

2.1 Creating a Virtual Server

Deployment is the process of taking code or binaries from one environment and moving them to a another environment where they can be executed. In our case, we'll be moving code from our development machine to a production server. We already have a development environment configured, but we still need to create a production environment that can be used as the target of our deployments. To do this, we'll use Vagrant[1] and VirtualBox.[2] These tools reduce the process of provisioning a virtual server to just a few steps.

The instructions in this chapter will describe how to build an Ubuntu Linux virtual machine on a Linux or Unix host system. But you can build your deployment environment with any platform you want. We'll try to address a few variations in Section 2.4, *Using Alternative Platforms*, on page ?. It's best to practice on an environment that is typical of the ones you will use in production, so you should pick one that makes sense. But the following steps will be specific to Ubuntu, VirtualBox, and Vagrant.

Let's get started by installing VirtualBox. Go to virtualbox.org,[3] and download and run the installer now. The VirtualBox user interface will open at the end of the installation, but you can close it. We're going to drive VirtualBox with Vagrant. To install Vagrant, download the binary installer for your operating system from the official website[4] and run it. The installer adds a `vagrant` command to our path, so we can check that both Vagrant and its connection to VirtualBox are working by running the following:

```
$ vagrant --version
Vagrant version 1.0.1
```

We could have installed the Vagrant gem, but that is not the preferred method of installation, since certain Vagrant commands do not work on JRuby. These include `vagrant ssh` and `vagrant reload`. It is possible to work around these deficiencies by running the `vagrant-ssh` script provided with the source code and by running `vagrant halt && vagrant up`, respectively, but using the binary distribution saves us a lot of time.

Now we're ready to build our deployment environment. The following command will create a fully functioning virtual machine running Ubuntu:

```
$ vagrant box add base-jruby http://files.vagrantup.com/lucid64.box
```

1. http://vagrantup.com/
2. https://www.virtualbox.org/
3. https://www.virtualbox.org/wiki/Downloads
4. http://downloads.vagrantup.com/tags/v1.0.1

```
[vagrant] Downloading with Vagrant::Downloaders::HTTP...
[vagrant] Downloading box: http://files.vagrantup.com/lucid64.box
[vagrant] Copying box to temporary location...
[vagrant] Extracting box...
[vagrant] Verifying box...
[vagrant] Cleaning up downloaded box...
```

Note that the lucid64.box file is very large (about 250MB), so the previous command may take some time to run.

Next, we'll move into the twitalytics directory, which contains the Git repository we created in *Preface*, on page ?. This is where we'll keep an image of our box along with some configuration files that we want under version control. To create these configuration files, we need to run the vagrant init command with the base-jruby argument.

```
$ cd ~/code/twitalytics
$ vagrant init base-jruby
```

This creates a Vagrantfile, which tells Vagrant the box that we want to interact with it when we use the vagrant command. Now we can boot our virtual machine like this:

```
$ vagrant up
[default] Importing base box 'base-jruby'...
[default] The guest additions on this VM do not match the install version of
VirtualBox! This may cause things such as forwarded ports, shared
folders, and more to not work properly. If any of those things fail on
this machine, please update the guest additions and repackage the
box.

Guest Additions Version: 4.1.0
VirtualBox Version: 4.1.8
[default] Matching MAC address for NAT networking...
[default] Clearing any previously set forwarded ports...
[default] Forwarding ports...
[default] -- 22 => 2222 (adapter 1)
[default] Creating shared folders metadata...
[default] Clearing any previously set network interfaces...
[default] Booting VM...
[default] Waiting for VM to boot. This can take a few minutes.
[default] VM booted and ready for use!
[default] Mounting shared folders...
[default] -- v-root: /vagrant
```

The VM is running! Let's log into it with the following command:

```
$ vagrant ssh
Linux lucid64 2.6.32-33-server #70-Ubuntu SMP ...
Ubuntu 10.04.3 LTS
```

```
Welcome to the Ubuntu Server!
* Documentation: http://www.ubuntu.com/server/doc
Last login: Mon Oct 17 14:24:10 2011 from 10.0.2.2
vagrant@lucid64:~$
```

The vagrant@lucid64:~$ prompt means that we are inside our virtual box.

Next, we need to update the system's package manager. Ubuntu is a Debian-based environment that uses the Advanced Packaging Tool (APT) to install software, which we can update with the following command:

```
vagrant@lucid64:~$ sudo apt-get update
```

Finally, we need to exit the virtual machine and add everything we've created to our Git repository. We'll put these changes in a new deployment branch and merge them in later, so run these commands:

```
$ git checkout -b deployment jruby
$ git add .
$ git commit -m "Added Vagrant configuration"
```

Our configuration is now under version control, and our virtual operating system is ready! Now we need to install some software on it.