CHAPTER 2

Virtualizing Development Environments

Creating virtualized development environments allows you to form encapsulated environments for specific projects that can have the same exact versions of operating systems, PHP, web servers, databases, libraries, settings, etc. as the real thing. These environments keep everything isolated from each other and can easily be destroyed and recreated as needed. This provides a number of benefits:

- Ability to run multiple projects on various PHP versions to match their production versions without trying to run them on your development machine.
- No chance of messing anything up with any configurations on your development machine when trying to install a library, change a setting, etc.
- Ability to take a snapshot of your environment that you can easily revert back to.

In this chapter, as we look into virtualizing your development environments, we will be focusing solely on using Vagrant, which is a tool for building complete, distributable development environments.

Traditionally, there are two approaches to setting up the development environment:

- Client and server processes run on the same machine.
- Client and the server run on different machines, which imitates the way the deployed application is executed by end users.

We'll look at the benefits of using virtualized environments, how to get and set up Vagrant, and how to provision your very first environment. By the end of the chapter you should be able to easily get up and running with your own virtual machine after running just one simple command: `vagrant up`.

C. Russell, *PHP Development Tool Essentials*, DOI 10.1007/978-1-4842-0683-6_2

Introduction to Vagrant

There's a good chance you have heard of or maybe even looked at using Vagrant before. As previously mentioned, Vagrant is a tool for building complete, reproducible, and distributable development environments.

■ **Note** Vagrant is open source software distributed under MIT license.

It does this by following a consistent configuration pattern, allowing you to define sets of instructions to configure your virtual environment using Vagrant's specific language. These instructions are stored in a file called Vagrantfile, and since it is only text, it can easily be added to your project's source control repository, thus allowing the versioning of these environment configurations as well as allowing it to be easily distributed among other developers.

At the heart of it all, we can break a full Vagrant setup down into four pieces:

- Provider – This is the virtual platform that your Vagrant setup will run on. Since Vagrant doesn't provide any actual virtualization, it relies on providers to provide this functionality for you. By default, Vagrant supports VirtualBox. However, there are a number of other providers you can use, such as Docker, VMWare, Parallels, Hyper-V, and even cloud providers such as AWS and DigitalOcean.
- Boxes – Boxes are the virtual machine images used to build your Vagrant setup. They can be used by anyone and on any Vagrant provider. There are a growing number of public Vagrant boxes available for your use, some that are base OS installations and some with a preconfigured LAMP stack or other configurations and languages. In addition to the ones publicly available, you can also create your own Vagrant boxes that can be either shared publicly or used privately only by you and/or your team.
- Vagrantfile – Configuration file for Vagrant.
- Provisioners – Provisioners are used in Vagrant to allow you to automatically install software, alter configurations, and perform other operations when running the `vagrant up` process. There are a number of provisioners supported by Vagrant, but for the sake of this book we'll be looking at Bash, Puppet, and Ansible.

Installing Vagrant and VirtualBox

Before you can do anything with Vagrant, you must first have a virtual machine provider installed, such as the free and open source VirtualBox that we'll use throughout these examples. You of course also need Vagrant itself installed.

VirtualBox can be downloaded from its website, located at https://www.virtualbox.org. Simply download the appropriate installer package for your operating system and follow the instructions on the installer to install.

Like VirtualBox, Vagrant can be downloaded from its website at http://www.vagrantup.com. Download the appropriate installer package for your operating system and follow the instructions on the installer to install. Once installed, the vagrant command will be available to you in your terminal.

In this book we will use Vagrant for a Linux environment.

Vagrant Commands

All commands issued to Vagrant are done using the vagrant command that's now available to you in your terminal. Let's take a look at the list of command options we have:

```
$ vagrant -h
Usage: vagrant [options] <command> [<args>]

    -v, --version    Print the version and exit.
    -h, --help       Print this help.

Common commands:
     box             manages boxes: installation, removal, etc.
     connect         connect to a remotely shared Vagrant environment
     destroy         stops and deletes all traces of the vagrant machine
     global-status   outputs status Vagrant environments for this user
     halt            stops the vagrant machine
     help            shows the help for a subcommand
     hostmanager
     init            initializes a new Vagrant environment by creating a Vagrantfile
     login           log in to HashiCorp's Atlas
     package         packages a running Vagrant environment into a box
     plugin          manages plugins: install, uninstall, update, etc.
     provision       provisions the Vagrant machine
     push            deploys code in this environment to a configured destination
     rdp             connects to machine via RDP
     reload          restarts Vagrant machine, loads new Vagrantfile configuration
     resume          resume a suspended Vagrant machine
     share           share your Vagrant environment with anyone in the world
     ssh             connects to machine via SSH
     ssh-config      outputs OpenSSH valid configuration to connect to the machine
     status          outputs status of the Vagrant machine
     suspend         suspends the machine
     up              starts and provisions the Vagrant environment
     version         prints current and latest Vagrant version
```

```
For help on any individual commands, run `vagrant COMMAND -h`

Additional subcommands are available, but are either more advanced or are not
commonly used. To see all subcommands, run the command `vagrant list-commands`.
```

As you can see from the last bit of information outputted from this command, there are additional subcommands that are available for us to use. For the sake of this chapter, we'll be focusing on the most commonly used commands and subcommands.

Setting Up Our First Environment

With VirtualBox and Vagrant installed, getting up and running with your first Vagrant environment is a relatively short and easy process. At a minimum, all you need is a basic Vagrantfile and a selected Vagrant box to use.

For starters, we're going to set up a minimal install using a base Ubuntu 14.04 box. Perusing the Hashicorp (the official company behind Vagrant) community box-repository catalog, located at https://atlas.hashicorp.com/boxes/search, I see the box we want to use is ubuntu/trusty64. Using two commands, we'll initialize our Vagrant setup, download the box, install it, then boot our new virtual machine (VM).

The first thing you have to do is define Vagrant's home directory in the VAGRANT_HOME environment variable. This can be easily done by executing the following command in bash:

```
$ export VAGRANT_HOME=/some/shared/directory
```

Let's create a new folder just for this Vagrant instance that we're setting up, then we'll initialize the Vagrant setup:

```
$ mkdir VagrantExample1
$ cd VagrantExample1
$ vagrant init ubuntu/trusty64
```

You should see a message returned that tells you a Vagrantfile has been placed in your directory and you're ready to run vagrant up. Before we do that, let's take a look at the initial Vagrantfile that was generated:

```
# All Vagrant configuration is done below. The "2" in Vagrant.configure
  # configures the configuration version (we support older styles for
  # backward compatibility). Please don't change it unless you know what
  # you're doing.
Vagrant.configure(2) do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://atlas.hashicorp.com/search.
  config.vm.box = "ubuntu/trusty64"
```

```
# Disable automatic box update checking. If you disable this, then
# boxes will only be checked for updates when the user runs
# `vagrant box outdated`. This is not recommended.
# config.vm.box_check_update = false

# Create a forwarded port mapping, which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.
# config.vm.network "forwarded_port", guest: 80, host: 8080

# Create a private network, which allows host-only access to the machine
# using a specific IP.
# config.vm.network "private_network", ip: "192.168.33.10"

# Create a public network, which generally matches to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
# config.vm.network "public_network"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
#   vb.memory = "1024"
# end
#
# View the documentation for the provider you are using for more
# information on available options.

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.
# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
# end
```

```
  # Enable provisioning with a shell script. Additional provisioners such as
  # Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline <<-SHELL
  #   sudo apt-get install apache2
  # SHELL
end
```

As you can see, most of the options here are commented out. The only configuration options line that isn't is:

```
config.vm.box = "ubuntu/trusty64"
```

This line tells Vagrant to use the box that we specified with our `vagrant init` command.

Initial VM setup

We're now ready to issue our next and final command, `vagrant up`. This will boot our VM for the first time and do any initial setup (provisioning) that we've told it to do. For now, this is just a basic system, so it will download the box we chose for the first time and import it, then just set up the initial SSH keys and make the machine available to us. See here:

```
$ vagrant up --provider virtualbox
```

You will see quite a bit of output from Vagrant as it downloads and brings up this initial box. The last few lines let you know it was a success and is ready for your use:

```
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
==> default: Mounting shared folders...
    default: /vagrant => /Apress/VagrantExample1
```

We now have a new VM running Ubuntu 14.04. We can connect to this VM via `ssh`, just like on any other Linux machine. With Vagrant, we do this by issuing the `vagrant ssh` command:

```
$ vagrant ssh
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-45-generic x86_64)

 ...

vagrant@vagrant-ubuntu-trusty-64:~$
```

The Vagrant user is the default user set up with each box. This user has full `sudo` privileges without needing any additional passwords.

■ **Note** Remember to run the command `vagrant -help` to get the entire list of commands you can use with Vagrant.

Shared Folders

By default, Vagrant will share your project's folder with the `/vagrant` directory inside of your VM. This allows you to easily edit files located directly in your project on your development machine and see those changes immediately reflected in the VM. A typical use for this would be to set up Apache on your Vagrant box and point the site root folder to somewhere within the `/vagrant` directory. Also, you can specify additional shared directories using the `config.vm.synced_folder` configuration parameter in the default Vagrantfile.

Networking

Vagrant provides multiple options for configuring your VM's networking setup. All network options are controlled using the `config.vm.network` method call. The most basic usage would be to use a forwarded port, mapping an internal port such as port 80 for regular HTTP traffic to a port on your host machine. For example, the following configuration line will make regular web traffic of your VM accessible at `http://localhost:8080`:

```
config.vm.network "forwarded_port", guest: 80, host: 8080
```

If you would prefer to specify a private IP address from which you can instead access the entire VM on your local network, you can use the `config.vm.network "private_network"` method call:

```
config.vm.network "private_network", ip: "192.168.56.102"
```

VM Settings

If you wish to change the amount of RAM or CPU that your VM is using, you can do so with the section of our Vagrantfile that starts with `config.vm.provider "virtualbox" do |vb|`. You will notice two entries there already that are commented out, one setting the Virtualbox GUI settings, the other setting the memory. If we want to change the memory as well as the default virtual CPU available to our image–to, say, 2048 MB memory and 2 CPUs–we can do so by adding the following under that section of our Vagrantfile:

```
config.vm.provider "virtualbox" do |vb|
    # Customize the amount of memory on the VM:
    vb.memory = "2048"

    # 2 virtual CPU's
    vb.cpus = 2
end
```

Before we apply this change, let's check to see what our VM is currently showing:

```
$ vagrant ssh
vagrant@vagrant-ubuntu-trusty-64:~free -m
             total       used       free     shared    buffers     cached
Mem:           489        331        158          0         12        207
-/+ buffers/cache:        112        377
Swap:            0          0          0

vagrant@vagrant-ubuntu-trusty-64:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2700.450
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
                  cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx rdtscp
                  lm constant_tsc rep_good nopl pni monitor ssse3 lahf_lm
bogomips        : 5400.90
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

We can apply these changes to our Vagrantfile by running the `vagrant reload` command, which will be the same as doing a `vagrant halt` to shut down the VM and then a vagrant up to start it back up:

```
$ vagrant reload
```

Let's ssh in again and check our VM memory and CPU settings now:

```
$ vagrant ssh
vagrant@vagrant-ubuntu-trusty-64:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          2001        208       1793          0         11         77
-/+ buffers/cache:        120       1881
Swap:            0          0          0

vagrant@vagrant-ubuntu-trusty-64:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2702.438
cache size      : 6144 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
                  cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
                  rdtscp lm constant_tsc rep_good nopl pni ssse3 lahf_lm
bogomips        : 5404.87
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

Processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2702.438
cache size      : 6144 KB
physical id     : 0
siblings        : 2
```

```
core id         : 1
cpu cores       : 2
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
                  cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
                  rdtscp lm constant_tsc rep_good nopl pni ssse3 lahf_lm
bogomips        : 5404.87
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

Removing VMs

Now, just as easily as we set up this VM, let's destroy it and all traces of it with another simple command, vagrant destroy:

```
$ vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
```

Just like that, our Vagrant VM is gone. However, our Vagrantfile is still intact, and the VM can be brought right back again by simply issuing another vagrant up.

Default Vagrant LAMP box

Our previous example is just a basic, bare Linux machine without Apache, MySQL, or PHP installed. This isn't very helpful if you're setting this box up for PHP development, unless you want to roll your own custom configurations.

Luckily, there are a number of community-provided Vagrant boxes that are already pre-configured with Apache, MySQL, and PHP, as well as some that already have popular PHP frameworks and platforms installed, such as Laravel, Drupal, and others.

Using the aforementioned Atlas community repository catalog or the Vagrantbox.es catalog (http://www.vagrantbox.es/), you can search and find a box that will work for you without any other configuration changes needed.

Advanced Configurations Using Ansible, Bash, and Puppet

As you can see from our initial example, it's extremely easy to get a VM up and running with Vagrant. However, just a basic VM isn't going to be of much use to us when setting it up as a full development environment that is supposed to mirror our production setup. If you don't find a Vagrant box that already has LAMP configured, then having to install and configure Apache, MySQL, and PHP manually each time you set up a new VM makes Vagrant a lot less useful.

It's also common that even if LAMP is already set up, there will be a number of configuration operations that need to be run after the initial setup, such as pointing Apache to a different public folder for your framework, or setting up a database for your application. This is where advanced configurations using one of the Vagrant-supported provisioners come in handy.

Vagrant supports a number of provisioners. For the sake of this chapter, we are going to look at Ansible, Bash, and Puppet. If you're only familiar with Bash, then it's the easiest to jump in and start using. However, there are many preconfigured packages available for Ansible (playbooks), Chef (recipes/cookbooks), and Puppet (modules) that will drastically cut down on the time it would take you to do these tasks even in Bash using basic commands.

Bash (Shell) Provisioner

Let's start with a simple example by installing Apache, MySQL, and PHP using a simple bash script. This entire Vagrant setup consists of two files–the Vagrantfile and our bash script. We're going to call this script `provision.sh`. This script will install the Ubuntu repo versions of Apache, MySQL, and PHP using `apt-get`.

We use the following line in our Vagrantfile to tell Vagrant to use Bash as a provisioner and then to use the `provision.sh` script:

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

The contents of our `provision.sh` script are as follows:

```
#!/bin/sh

set -e # Exit script immediately on first error.
set -x # Print commands and their arguments as they are executed.

export DEBIAN_FRONTEND=noninteractive

# Do an update first
sudo apt-get -y update

# Install Apache
sudo apt-get -y install apache2
```

```
# Install PHP
sudo apt-get -y install php5 php5-mysql php5-cli php5-common

# Install MySQL
echo mysql-server mysql-server/root_password password 123 | sudo debconf-
set-selections
echo mysql-server mysql-server/root_password_again password 123 | sudo
debconf-set-selections
sudo apt-get -y install mysql-server-5.6

# Restart Apache & MySQL to ensure they're running
sudo service apache2 restart
sudo service mysql restart
```

As you can see, with this script we're just running the same commands we would run if we were manually setting up our VM; however, we automate the process since Vagrant can run the Bash commands for us.

Puppet Provisioner

Puppet is a configuration management system that allows us to create very specialized Vagrant configurations. Puppet can be used to form many different types of Vagrant configurations via the inclusion of specific Puppet modules inside of your project. These modules can be obtained from the Puppet Forge site at `https://forge.puppetlabs.com/`. Each one of the modules you use will have anywhere from a few to many different configuration options so as to tailor the environment to your exact needs. You should reference the README for each one of these modules as you start customizing to find out what options are made available to you.

For this example, download the Apache, MySQL, and PHP manifests from Puppet Forge and organize them according to their recommended hierarchy as noted on the website. You should also download a few required dependencies from Puppet Forge as well. We'll use these to set up a VM with Apache, MySQL, and PHP just like with our Bash example.

■ **Note** A *manifest* is the instructions that tell Puppet what to do with all of the modules.

We'll place the Puppet manifest in the default location in which Vagrant will look for it, under `manifests/default.pp`. First, update the Vagrantfile to tell Vagrant that we're now using Puppet as a provisioner:

```
config.vm.provision "puppet" do |puppet|
      puppet.manifests_path = "manifests"
      puppet.manifest_file  = "default.pp"
      puppet.module_path = "modules"
end
```

Our directory structure is as shown in Figure 2-1.

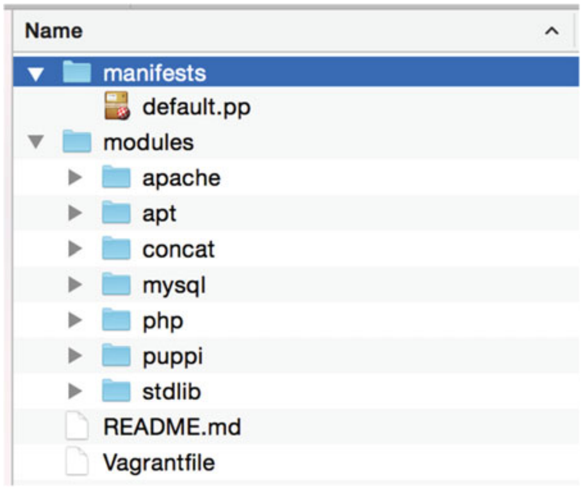

```
Name
▼ manifests
      default.pp
▼ modules
    ▶ apache
    ▶ apt
    ▶ concat
    ▶ mysql
    ▶ php
    ▶ puppi
    ▶ stdlib
  README.md
  Vagrantfile
```

Figure 2-1. *Puppet Vagrant directory structure*

The `default.pp` file located under the main `manifests` directory is the file that tells Puppet what to install and configure for our VM. This is where you would define the various configuration options you need for your setup. For the sake of this example, I've kept the configurations simple and concise:

```
# Update apt-get
exec { 'apt-get update':
  command => 'apt-get update',
  path    => '/usr/bin/',
  timeout => 60,
  tries   => 3
}

class { 'apt':
  always_apt_update => true
}

# Install puppet in our VM
package {
  [
    'puppet',
  ]:
    ensure  => 'installed',
    require => Exec['apt-get update'],
}
```

```
# Install Apache, set webroot path
class { 'apache':
  docroot => '/var/www/html',
  mpm_module => 'prefork'
}

# Install MySQL, setting a default root password
class { '::mysql::server':
  root_password           => '123',
  remove_default_accounts => true
}

# Install PHP and two PHP modules
class { 'php':
  service => 'apache2'
}
php::module { 'cli': }
php::module { 'mysql': }

# Install and configure mod_php for our Apache install
include apache::mod::php
```

As you can see, there is a bit more going on here than we had in our Bash script; however, having the power and flexibility of being able to make configuration changes and specific installation setups just by adding in a few additional configuration parameters makes Puppet a great choice for complex setups.

Ansible Provisioner

Ansible is an automation tool that can be used for many types of autonomous tasks and is not limited to use with Vagrant. With Vagrant, we can use it along with a *playbook* to automate the setup and configuration of our Vagrant machines. An Ansible playbook is simply a YAML file that instructs Ansible on what actions to perform.

■ **Note** You may want to consider running Ansible against the machine you are configuring because it can be quicker than using a combination of setup scripts.

Using Ansible is much more lightweight than using Puppet, as there is no need to download or include various modules to perform the tasks you need, and the guest VM doesn't need anything special installed. The only requirement is that the host machine running Vagrant have Ansible installed. Installation instructions for a variety of operating systems can be found in the Ansible documentation at `http://docs.ansible.com/intro_installation.html#getting-ansible`.

For this example, we'll configure a very simple Ansible playbook to set up Apache, MySQL, and PHP on our Vagrant machine, just like in our Bash and Puppet examples. First, we must instruct Vagrant to use Ansible as the provisioner and supply the name of our playbook file:

```
config.vm.provision "ansible" do |ansible|
      ansible.playbook = "playbook.yml"
end
```

Then we instruct Ansible to install Apache, MySQL, and PHP:

```
- hosts: all
  sudo: true
  tasks:
    - name: Update apt cache
      apt: update_cache=yes
    - name: Install Apache
      apt: name=apache2 state=present
    - name: Install MySQL
      apt: name=mysql-server state=present
    - name: Install PHP
      apt: name=php5 state=present
```

Even though this configuration seems very simple, don't let it fool you; Ansible is very powerful and can perform complex configurations. We can easily make configuration customizations–just as we can with Puppet–by making use of Ansible templates, variables, includes, and much more to organize and configure a more complex setup.

Advanced Configuration Conclusion

As you can see, utilizing provisioners to automate the tasks of completely building your environment makes setting up your development environments much easier than having to manually do it over and over again. Each provisioner has a different approach for how it accomplishes these tasks, giving you a range of choices and flexibility for you and your project or environment.

Configuration Tools

Now that we have a better understanding of some of the core configuration settings and provisioners available to Vagrant, let's take a look at two configuration tools aimed at making the setup of these environments even easier.

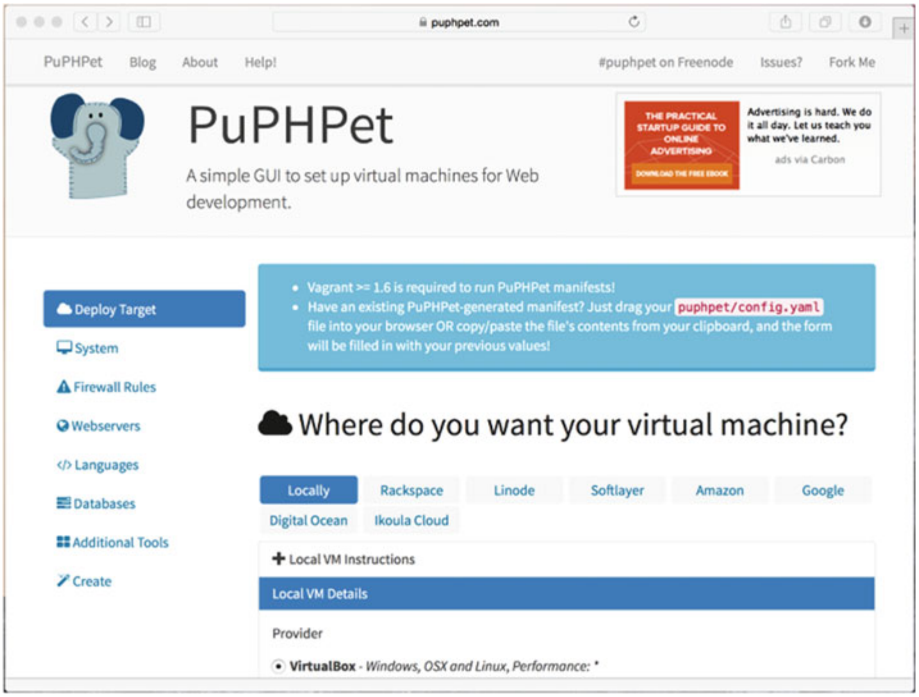
■ **Note** Both of these tools are under current development, so they're both constantly changing and progressing over time. It's been my experience with them that they're great for getting you up and running quickly, but they do have their periodic issues and weaknesses.

PuPHPet

This tool, pronounced "puffet," uses Puppet as the provisioning language and provides an easy-to-follow GUI for configuring your environment.

Accessing PuPHPet

You can access this tool by visiting `https://puphpet.com`, as seen in Figure 2-2.



puphpet.com

PuPHPet

A simple GUI to set up virtual machines for Web development.

THE PRACTICAL STARTUP GUIDE TO ONLINE ADVERTISING
DOWNLOAD THE FREE EBOOK
Advertising is hard. We do it all day. Let us teach you what we've learned.
ads via Carbon

- Vagrant >= 1.6 is required to run PuPHPet manifests!
- Have an existing PuPHPet-generated manifest? Just drag your `puphpet/config.yaml` file into your browser OR copy/paste the file's contents from your clipboard, and the form will be filled in with your previous values!

Where do you want your virtual machine?

● **VirtualBox** - *Windows, OSX and Linux, Performance: **

Figure 2-2. *PuPHPet web-based Puppet configuration tool*

PuPHPet is publicly hosted on GitHub, is open-source, and anyone can fork over and contribute to it. This tool works by generating a manifest YAML file along with the respective Puppet modules needed to build and configure your new VM environment. You can use the configurations it generates directly as is, or you can make modifications and tweaks as needed.

Setting Up and Using PuPHPet Configurations

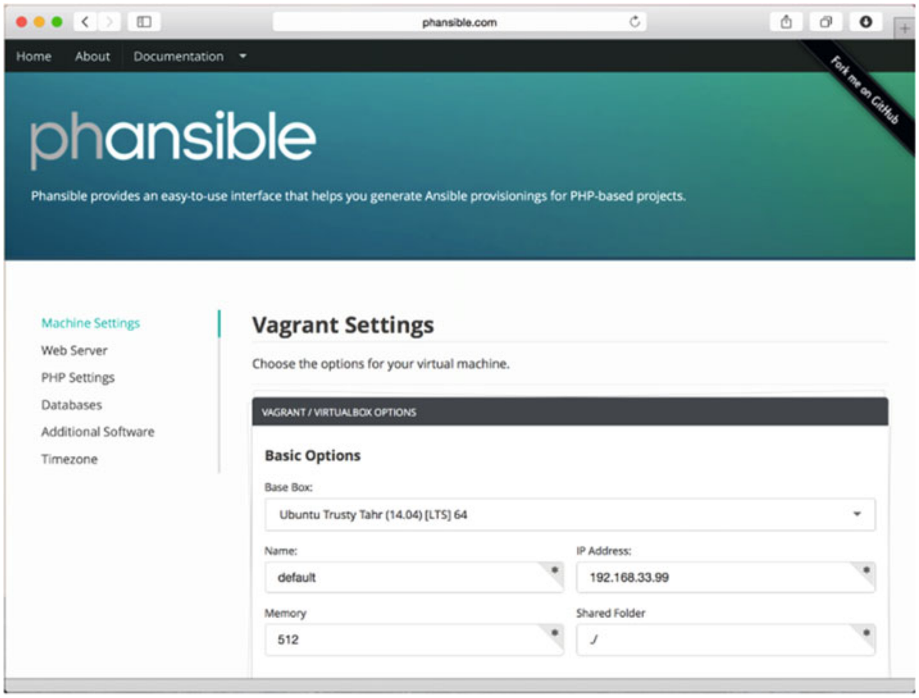
Once you walk through each of the setup options on PuPHPet, you will download your custom configuration. This download consists of the Vagrantfile and a `puphpet` directory that contains all of the necessary Puppet manifests and modules needed for your environment.

Simply copy these two items to your project directory and you're ready to run `vagrant up` to set up and provision this environment.

■ **Tip** One nice feature of the configuration setup generated by PuPHPet to note is the file structure under the `files` directory. This directory consists of four other directories, which allows you to create scripts that will execute once, every time, during startup, and so on. For example, you could utilize the `execute` once to perform post-setup cleanup, running custom commands needed to provision PHP application-specific dependencies (like `composer install`), as well as setting up databases data, etc.

Phansible

This is a newer tool that's become available, and it uses Ansible instead of Puppet as the provisioning language. It's similar to PuPHPet, but as of right now it does not have all of the bells and whistles that are available using PuPHPet. It also is publicly hosted on GitHub, is open source, and is available for anyone to contribute to (Figure 2-3).



Phansible provides an easy-to-use interface that helps you generate Ansible provisionings for PHP-based projects.

Vagrant Settings

Choose the options for your virtual machine.

Basic Options

Base Box: Ubuntu Trusty Tahr (14.04) [LTS] 64

Name: default

IP Address: 192.168.33.99

Memory: 512

Shared Folder: ./

Figure 2-3. *Phansible web-based Ansible configuration tool*

Just as with PuPHPet, once you walk through each of the setup options on Phansible, you will download your custom configuration. This download also consists of the Vagrantfile and an `ansible` directory that has the `playbook.yml` file. It also holds several other items that can be used along with Ansible that we didn't utilize in our basic Ansible example earlier (such as the templates that were mentioned).

Phansible can be found at:

`http://phansible.com/`

Vagrant Plugins

As you begin using Vagrant more and more, you will periodically need additional functionality that isn't provided to you out of the box from Vagrant. Fortunately, Vagrant has a plugin system, and many times a plugin exists to do exactly what you need.

Vagrant plugins are very easily installed using the `vagrant plugin install plugin-name-here` subcommand. Here are a few helpful plugins that you may find useful as you begin to use Vagrant as your development environment choice:

- Vagrant Host Manager – This plugin manages the `hosts` file on the host machine, allowing you to easily specify a temporary `hosts` entry that maps to your VM's IP address. This allows you to easily set up access to your development environments using something similar to the production address. So if you have `www.someproduct.com` you could set up something like `dev.someproduct.com` or `www.someproduct.dev` and use the Vagrant Host Manager to automatically add this to your `hosts` file. It will add and remove this entry for you during the `vagrant up` and `halt` commands. This plugin is very useful when combined with specifying your own private network IP address for your VM. Additional information on this plugin can be found here: `https://github.com/smdahlen/vagrant-hostmanager`.
- Vagrant Share – This plugin, installed by default, allows you to share your environment with anyone, anywhere using a free account with HashiCorp.
- Vagrant Librarian Puppet – This plugin allows for Puppet modules to be installed using `Librarian-Puppet`.
- Vagrant Ansible Local – This plugin allows you to use Ansible as your provisioner, but instead allows Ansible to be run from within the guest VM rather than making the host machine dependent have Ansible installed.
- Providers – Although this isn't a specific plugin, there are many different plugins that allow Vagrant to be run on other providers, such as Parallels, KVM, AWS, DigitalOcean, and many more.

For a complete Vagrant plugin listing you can check this web page: `http://vagrant-lists.github.io/`

Summary

With the introduction of Vagrant, using virtual machines in your development process makes perfect sense. Hopefully, the topics covered here not only demonstrated this value you to you, but also gave you everything you need to be up and running with it on your next project or even on your existing project in no time flat. In the next chapter, we will discuss coding standards in order to define how to structure your code.