Creating Development Environments with Vagrant

Chapter No. 6
"Working with Multiple Machines"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.6 "Working with Multiple Machines"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Michael Peacock (www.michaelpeacock.co.uk) is an experienced Senior/Lead Developer and a Zend Certified Engineer from Newcastle, UK, with a degree in Software Engineering from the University of Durham.

After spending a number of years running his own web agency, managing the development team, and working for Smith Electric Vehicles on developing their web-based Vehicle Telematics platform, he currently serves as a CTO for Ground Six (www.groundsix.com), an ambitious tech company, where he leads the development team and manages the software development processes.

He is the author of Drupal 7 Social Networking, PHP 5 Social Networking, PHP 5 E-Commerce Development, Drupal 6 Social Networking, Selling Online with Drupal E-Commerce, and Building Websites with TYPO3. Other publications Michael has been involved in include Mobile Web Development, Drupal for Education and E-Learning, and Jenkins Continuous Integration Cookbook, for which he acted as a Technical Reviewer.

For More Information:
Michael has also presented at a number of user groups and conferences including PHP UK Conference, Dutch PHP Conference, ConFoo, PHPNE, PHPNW, and Cloud Connect.

You can follow Michael on Twitter, @michaelpeacock, or find out more about him through his blog, www.michaelpeacock.co.uk.

I’d like to thank all the staff at Packt Publishing, in particular, Erol Staveley, Robin de Jongh, Akash Poojary, and Manasi Pandire for seeing this book through to fruition. I’d also like to thank my Technical Reviewer, Chad Thompson, who helped ensure the technical quality of the book was up to scratch. My thanks also go to my friends and family, in particular, my wife Emma for her support while working on the book. Finally, I’d like to thank you, the reader; I hope you enjoy this book and enjoy the benefits of using virtualized development environments with Vagrant!
Working with Multiple Machines

Web-based software projects are increasingly complicated, with a range of different dependencies, requirements, and interlinking components. Swapping between projects, which require different versions of the same software, becomes troublesome. Getting team members up and running on new projects also becomes time-consuming.

Vagrant is a powerful tool for creating, managing, and working with virtualized development environments for your projects. By creating a virtual environment for each project, their dependencies and requirements are isolated, and don't interfere; they also don't interfere with software installed on your own machine such as WAMP or MAMP. Colleagues can be up and running on a new project in minutes with a single command. With Vagrant, we can wipe the slate clean if we break our environment, and be back up and running in no time.

What This Book Covers

Chapter 1, Getting Started with Vagrant, introduces the concept of virtualization, its importance in the role of the development environment, and walks through the Vagrant installation process.

Chapter 2, Managing Vagrant Boxes and Projects, walks through creating Vagrant projects, exploring and configuring the Vagrantfile, and working with base boxes.

Chapter 3, Provisioning with Puppet, explores the provisioning tool Puppet and how to create Puppet manifests to provision a server.

Chapter 4, Provisioning with Chef, explores the provisioning tool Chef and how to create Chef recipes to provision a server.

Chapter 5, Provisioning with Vagrant using Puppet and Chef, discusses how to use both Puppet and Chef within the context of Vagrant to provision development environments.

Chapter 6, Working with Multiple Machines, explores using Vagrant to create and manage projects, which use multiple virtual machines which communicate with each other.

Chapter 7, Creating Your Own Box, discusses the process of creating your own base box for use within a Vagrant project.

Appendix, A Sample LAMP Stack, walks through the process of creating a LAMP server within a new Vagrant project.

For More Information:
So far we have used Vagrant to build a development environment contained within a virtual machine; one of the key aspects being that this virtual machine mimics our production environment. It gives us the flexibility of being able to encapsulate the development environment for different projects so that we can easily switch from one to another without having to modify the software on our own machines.

In many cases, the features we have learned so far are enough. However, web projects are more and more complex, with developers continually improving, having to deal with multiple machines in their architecture to help with scalability and stability. While it can be said that scalability and stability issues won't impact on our development environment (as we won't have huge amounts of traffic coming to our development environment, unless we load test it), we want to ensure the coupling between servers within our code (such as application code connecting to a remote database) works in our development environment before we put the project online.

Thankfully, Vagrant has support for running multiple virtual machines at the same time within the same project. We can use this to test multi-machine architectures and distributed systems on our own local machine before we share our changes with colleagues in a staging environment and before the project goes live. Replicating a multi-machine environment in development greatly helps us improve the reliability of our projects and instills confidence in the work that we do.

For More Information:
In this chapter you will learn:

- How to run multiple virtual machines within a single Vagrant project
- How to provide different distinct configurations to these virtual machines including:
  - Names
  - IP addresses on a private network so they can communicate with one another
  - Base boxes
  - Provisioning
  - Shared folders
- How to connect to the different virtual machines over SSH without having to know or remember their IP addresses

**Using multiple machines with Vagrant**

In order to use multiple virtual machines within our project, we need to tell Vagrant about them, and we need to provide additional configuration for the individual virtual machines.

**Defining multiple virtual machines**

Within the standard Vagrant project configuration file, we can tell Vagrant that we wish to assign a name to a virtual machine being managed by the project. Within this subconfiguration, we provide the information Vagrant needs which are specific to that VM.

The syntax for the subconfiguration is:

```ruby
config.vm.define :name_of_the_vm do |name_of_the_vm|
  # configuration specific to the virtual machine
end
```

Applied to a project which requires two virtual machines, named `server1` and `server2`, both running the `precise64` box:

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure("2") do |config|
  config.vm.define :server1 do |server1|
    server1.vm.box = "precise64"
  end
  config.vm.define :server2 do |server1|
    server1.vm.box = "precise64"
  end
end
```

For More Information:

Connecting to multiple virtual machines over SSH

When our multiple machines boot up in our multi-machine project, Vagrant automatically maps different ports from our host machine to the SSH ports on the various guest machines.

Let’s look at the console output when booting a Vagrant project with two virtual machines within it:

```
Michael-MacBook-Pro:multimachine michael$ vagrant up
Bringing machine 'webserver' up with 'virtualbox' provider...
[webserver] Importing base box 'precise64'...
[webserver] Matching MAC address for NAT networking...
[webserver] Setting the name of the VM...
[webserver] Clearing any previously set forwarded ports...
[webserver] Clearing any previously set network interfaces...
[webserver] Preparing network interfaces based on configuration...
[webserver] Forwarding ports...
[webserver] -- 22 => 2222 (adapter 1)
[webserver] Booting VM...
[webserver] Waiting for VM to boot. This can take a few minutes...
[webserver] VM booted and ready for use!
[webserver] Configuring and enabling network interfaces...
[webserver] Mounting shared folders...
[webserver] -- /vagrant
Bringing machine 'database' up with 'virtualbox' provider...
[database] Importing base box 'precise64'...
[database] Matching MAC address for NAT networking...
[database] Setting the name of the VM...
[database] Clearing any previously set forwarded ports...
[database] Clearing any previously set network interfaces...
[database] Preparing network interfaces based on configuration...
[database] Forwarding ports...
[database] -- 22 => 2200 (adapter 1)
[database] Booting VM...
[database] Waiting for VM to boot. This can take a few minutes...
[database] VM booted and ready for use!
[database] Configuring and enabling network interfaces...
[database] Mounting shared folders...
[database] -- /vagrant
```

This gives us the opportunity to connect to the SSH ports on our virtual machines.

For More Information:
As shown in the preceding screenshot, Vagrant maps the SSH port on the virtual machine designated 'webserver' to port 2222 on the host machine, and the SSH port of the machine designated 'database' to the port 2200.

This gives us the opportunity of simply using the standard SSH command from a terminal (or the likes of Putty on a Windows machine) to connect to localhost with the port number that Vagrant assigns to each machine.

To connect to the machine which is mapped to port 2200 we simply run the following command:

```
ssh vagrant@localhost -p2200
```

-p2200 tells the command to use a non-standard port, and specifies the port we wish to use, in this case 2200. As you can see, this then lets us into the appropriate machine, as follows:

```
[Michael-MacBook-Pro:multimachine michael]$ ssh vagrant@localhost -p2200
vagrant@localhost's password: 
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-18-generic x86_64)

* Documentation:  https://help.ubuntu.com/
* Last login: Fri Sep 16 06:25:18 2016 from 10.9.2.2
vagrant@precision64:~$ 
```

Alternatively, we can use the `vagrant ssh` command to connect to the virtual machines. The difference being that in a multi virtual machine environment, we must also provide the name of the virtual machine. For example, `vagrant ssh database`. This is the most common usage of connecting to a machine, rather than directly connecting to the virtual machine via its IP address.

## Networking multiple virtual machines

In a single virtual machine project, the IP address of the virtual machine isn't that important. In a multi virtual machine project however, it is more likely that we want the two machines to communicate with one another directly; in order to do that, we need to be aware of their IP addresses. As we want to have our Vagrant projects distributed to our team members, and some of these team members may be within the same office, we need to:

- Predefine the IP address so that any of our projects code which needs to communicate with the other virtual machine can do so without the other team members needing to change configurations
- Ensure that the virtual machines are running on a private network only attached to the machine of the user running it; this will prevent IP address conflicts within the network
In order to do this, we simply use the networking options which we learned about in Chapter 2, Managing Vagrant Boxes and Projects. Because we want the virtual machines running in a private network, it makes sense to use a range of private IP addresses which are different to your own network. For example, my network range is 192.168.1.xxx, so I will use the range 10.11.1.xxx for my virtual machine network, as shown in the following code:

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure("2") do |config|
  config.vm.define :server1 do |server1|
    server1.vm.box = "precise64"
    server1.vm.network :private_network, ip: "10.11.1.100"
  end
  config.vm.define :server2 do |server2|
    server2.vm.box = "precise64"
    server2.vm.network :private_network, ip: "10.11.1.101"
  end
end
```

Let's test this out and test that we can connect from one machine to the other:

1. Power up the project (`vagrant up`)
2. Connect to server1 (`vagrant ssh server1`)
3. Ping server2 from server1 (ping 10.11.1.101)

The output shows that we are able to communicate over the network from server1 to server2 as follows:

```
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.8.0-32-generic x86_64)\n
  * Documentation:  https://help.ubuntu.com/
  Welcome to your Vagrant-built virtual machine.
Last login: Thu Sep 19 16:33:10 2012 from 192.168.1.101
ping 10.11.1.101 (10.11.1.101) 56(84) bytes of data.
64 bytes from 10.11.1.101: icmp_seq=1 ttl=64 time=0.779 ms
64 bytes from 10.11.1.101: icmp_seq=2 ttl=64 time=0.378 ms
64 bytes from 10.11.1.101: icmp_seq=3 ttl=64 time=0.302 ms
64 bytes from 10.11.1.101: icmp_seq=4 ttl=64 time=0.370 ms
64 bytes from 10.11.1.101: icmp_seq=5 ttl=64 time=0.627 ms
64 bytes from 10.11.1.101: icmp_seq=6 ttl=64 time=0.351 ms
--- 10.11.1.101 ping statistics ---
 6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.351/0.465/0.779/0.150 ms
vagrant0@precise64:~
```

For More Information:
Provisioning the machines separately

As the virtual machines in our projects are going to be used for different purposes, we need to use different provisioning for the machines so they both have the software and configurations needed to do their job.

We take the provisioning code, which we have learned in Chapter 3, Provisioning with Puppet, and Chapter 4, Provisioning with Chef, and we place the relevant code within the virtual machines subconfiguration. There are some key changes which we need to make:

- The opening line of the provision code must reference the server name of the virtual machine it relates to
- For Puppet, we should use a different manifest file for the two virtual machines
- For Chef, we would apply different roles to different machines

The following code provisions both the machines by using Puppet. They both rely on the same set of Puppet modules, the same path pointing to the manifests folder, however they both use different manifests to set up the projects (alternatively, we could configure the machines, identify themselves as nodes to a puppet master to retrieve the appropriate configuration):

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure("2") do |config|
  config.vm.define :server1 do |server1|
    server1.vm.box = "precise64"
    server1.vm.network :private_network, ip: "10.11.1.100"
    server1.vm.provision :puppet do |puppet|
      puppet.manifests_path = "provision/manifests"
      puppet.manifest_file  = "server1.pp"
      puppet.module_path = "provision/modules"
    end
  end

  config.vm.define :server2 do |server2|
    server2.vm.box = "precise64"
    server2.vm.network :private_network, ip: "10.11.1.101"
    server2.vm.provision :puppet do |puppet|
      puppet.manifests_path = "provision/manifests"
      puppet.manifest_file  = "server2.pp"
      puppet.module_path = "provision/modules"
    end
  end
end
```

For More Information:
Within the provisions for each machine, we would need to ensure that we allow both machines to communicate with one another. For example, by default a MySQL server won't accept connections from a remote server, so it would be needed to modify (or replace) the configuration file with one which allows it, or we would have to use a Puppet module or Chef cookbook which allowed us to modify this configuration value.

You should check the documentation for any software you are communicating with over the network to see how it needs to be configured. With MySQL you need to edit the my.cnf file and set the bind address to 0.0.0.0.

**Destroying a multi-machine project**

If we want to completely remove the virtual machines for our project from our host machine, we can use the `vagrant destroy` command as with normal projects. The difference being that Vagrant will ask us to confirm the removal of each machine within the project, as shown in the following screenshot:

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

**Summary**

In this chapter, we set up a Vagrant project which uses multiple virtual machines. During the course of this we learned:

- How to create multiple virtual machines within a single project
- How to assign names to the individual machines
- How to connect to the individual machines over SSH by using both the operating systems built in the SSH command and the `vagrant ssh` command
- How to configure the individual virtual machines within the project, providing IP addresses, base boxes, and provisioning options to them

Now we have learned the vast majority of Vagran's functionality and how to use it within different project scenarios. In the next chapter, we will look at how to build our own custom base box to use with our projects, configuring a blank operating system installation into a compatible base image.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.