VAGRANT COOKBOOK

A practical guide to Vagrant and its most popular Provisioners

SECOND EDITION

Erika Heidi

Vagrant CookBook

A practical guide to Vagrant

Erika Heidi

This book is for sale at http://leanpub.com/vagrantcookbook

This version was published on 2017-01-27

Tweet This Book!

Please help Erika Heidi by spreading the word about this book on Twitter!

The suggested hashtag for this book is #vagrantcookbook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#vagrantcookbook *to Alice*

Contents

Foreword . . . . **i**

Preface: Second Edition . . . . **iii**

Preface: First Edition . . . . **iv**

Acknowledgments . . . . **v**

Introduction **1**
- What to expect from this book 2
- Assumptions 2

Getting Started **3**
- How Vagrant Works 3
- Terminology 3
- Requirements 5
- Installation 5
- Updating Vagrant 6
- Vagrant Commands 6
- Your first Vagrant Up 9

Foreword

You've probably heard the old saying "Jack of All Trades, Master of None."

According to Wikipedia, it's a very old saying, with the "Jack of All Trades" part dating back to the early 1600s. Originally, it was a compliment, praising someone for their diversity and handiness. Some wiseguy later added "Master of None" and changed it to the insult we know today.

Personally, that seems unfair. The world needs generalists: people who borrow from other languages, cross-pollinate between communities, reuse old tools for new purposes. True, they're often not the deepest experts in these fields. Yet they can make valuable contributions because they have so many references to draw from. Perhaps that's why there's a modern variation of the saying: "Jack of All Trades, Master of None, but often better than a Master of Any."

Likewise, this book might say "Vagrant Cookbook" on the cover but if you dig deeper, you'll find it's about an entire ecosystem of tools. To get the most out of Vagrant, you need to know about provisioners, plugins, virtualization platforms, shell commands and more. Luckily, you don't need to master any of them (though that's always nice), you just need to be effective: A Jack (or Jill) of All Trades.

If you're new to Vagrant, this book is going to teach you how to start and help you make decisions about what tools to use. If already have Vagrant experience, the sections on alternative provisioners or multiple VMs are sure to have something new for you.

Introducing Vagrant is often a first step towards DevOps. If that's the case for you, keep in mind the two virtues of "communication" and "automation". Communicating openly will expose you to new viewpoints, new concerns and new references. Automation will let your team wield powerful tools while distributing the time investment. Together, these two things will help you grow team members who are "Jacks of All Trades, Master of Some."

This book is full of great tips and tricks for using Vagrant but there's one lesson in particular I hope you take from it. In an industry known for high stress and crazy deadlines, Erika’s enthusiasm is inspiring. Whether she’s automating image processing or drawing cartoon elephants, her approach can only be described as joyful.

That she channeled this energy and talent into an (excellent) book about virtual machine management should remind us how amazing computers really are. Her example encourages us to always set the bar just a little bit higher: not just in the work we do, but also in the fun we have.

Thanks for picking up the Vagrant Cookbook. You’re in the right place.

– Ross Tuck

Amsterdam, March 2014

Preface: Second Edition

Time really flies! It's been three years since I first released *Vagrant Cookbook*, and many things happened since then. I've been around the world talking about Vagrant in dozens of conferences. I had a great time working on Phansible[1] and helping people to get started using Vagrant and Ansible. I learned a lot about devOps and related subjects, and I also learned a lot about technical writing. I got a job in a company that I love, and I just recently came back to being an independent developer and writer as a personal choice, influenced by the birth of my daughter.

As a first commandment for this new moment in my life, I decided to give *Vagrant Cookbook* a **second edition**. I've been carefully updating its contents for the past month, one baby nap at a time, and I have to say that it's been a wonderful experience to come back to this project, where everything started for me. I felt tempted to say that I wish I had done it sooner, but the truth is that **now** seems to be just the perfect moment for this. Not that I have too much time in my hands (quite the opposite), but dedicating time to this other child made me feel more like *me* again, whatever that means.

It's good to see that Vagrant keeps being such a useful and versatile tool after these years, and that the updates didn't stop coming after Hashicorp moved into a more "enterprise-y" model. Snapshots, Vagrant Push, Ansible local… these are all great recent additions that I had the pleasure to include in this book. Updating the provisioning examples to PHP 7 was equally satisfying!

If you were a previous reader of *Vagrant Cookbook*, I hope you enjoy the updates. If you are a new reader, welcome! I hope this book will give you a smooth ride with Vagrant and its ecosystem.

Amsterdam, January 2017

[1] http://phansible.com

Preface: First Edition

I remember very clearly the first time I heard about Vagrant. It was early 2013, on my second visit to the AmsterdamPHP meetups. The talk was from my dear friend Michelle Sanver (a.k.a. Geekie), about Open Source. She was showing how easy it is to get involved and contribute to OSS projects, by simple cloning the project repository and running the mysterious command `vagrant up`.

I started using Vagrant the next day.

A few months later, and not before facing a considerable resistance from my coworkers and the lead developer (they were using remote servers for testing - FTP upload for every single change), I was able to introduce Vagrant in the company I was working for. Although that project was quite complex and creating a Puppet provision for it was slightly painful, it was a great opportunity to learn more about Vagrant and get used to the tricks and the *automate all the things* mindset.

After leaving the company and coming back to my independent projects, I started to submit some talks for PHP conferences, and naturally Vagrant was on top of my cool-subjects list. By that time, I was presented to LeanPub and found out the amazing platform they built for self-publishing - I must say that I personally love any service that promotes independent work. It's also no news that writing is a passion for me since I was little. So, connecting the dots, it was an obvious decision: I should write about my experiences with Vagrant.

This book is based on many experiments and lots of research, from a very curious and enthusiastic Vagrant user. I tried to put together everything you need to have a pleasant experience with Vagrant, in a truly practical way.

Acknowledgments

A special thanks to all the people who helped me directly or indirectly, including Mitchell Hashimoto, Vagrant's creator, not only for sharing such a great project with the developers community, but also for personally helping me to spread the word about my Vagrant usage research.

Thanks LeanPub for providing a complete self-publishing platform for independent writers, with so many nice features.

A big thanks must go to my friend Ross Tuck for writing an inspired and truthful foreword for this book.

I also would like to thank the PHP community, and specially the AmsterdamPHP user group, for the warm welcome I received since I moved from my hometown in Brazil to this amazing city, for the awesome friends I made and all support and encouragement they gave me.

And finally, I want to thank my husband, Hugo, for being incredibly supportive with my independent projects, for the faith he has in me and in my work. This book would hardly be possible without him.

Introduction

How many times did you hear the sentence "*but it works on my machine*" ? I bet you already said that too, because, well, it happens. We can't remember all the packages we already installed and all the configurations we set on our work machine, so it often takes some time to find out what went wrong when the project was shared to another co-worker, or worst, when deploying.

Also, if we need to deal with multiple projects, how to manage the dependencies and different software versions possibly needed?

If you are not familiar with Vagrant[2], this is the right moment to get acquainted to it. Vagrant provides a portable and reproducible development environment using virtual machines, all set up in the project repository - just clone, run `vagrant up` and you're done. You will never be hostage of the "*works on my machine*" statement again; the environment is exactly the same for all developers, regardless of the operating system running behind Vagrant.

Vagrant for proprietary projects

Vagrant can make your workflow way easier when you are working on a team; having the exact same testing environment for all co-workers will avoid many problems and add much more consistency to the overall project development.

Vagrant for open source projects

Vagrant enables more developers to contribute to your open source project - just clone the repository and you are ready to go. It's not only about setting up the right environment, but also automating the process of installing a database, cloning repositories, adding data fixtures and even running tests.

[2]http://docs.vagrantup.com/v2/

Vagrant for devops / system administrators

If you work with system administration or any kind of server management, Vagrant is the right tool for your tests. It supports the most common IT automation tools, such as Puppet, Ansible, Salt and Chef. Experiment with different setups, build your multi-server infrastructure and make sure everything works before going to production.

What to expect from this book

As a very practical guide, this book will cover Vagrant from the requirements and installation to slightly complex tasks, such as running multiple VMs and deploying "real" servers. It will walk you through the most popular Vagrant provisioners - Puppet, Chef and Ansible - showing their main characteristics and a quick guide to get you started.

This book will also cover some important pro tips to create your Vagrant projects; and finally, a collection of recipes for common provisioner tasks, such as installing packages, using templates, running commands etc.

Vagrant Cookbook targets beginner to intermediate users, also serving as a quick "getting started" guide for provisioners (Ansible, Puppet and Chef) and how to improve your current Vagrant setups.

Assumptions

This book assumes you are a developer experienced with command line, and you know how to setup a Linux server - you need to understand the problem before you can automate the solution, right? The tools we are going to use require some programming knowledge, since they work with concepts such as variables, conditionals, and loops.

This is a book about Vagrant, for people who are comfortable with programming and also system administration tasks like setting up a web server on Linux.

The examples in this book will target mostly the provisioning of PHP web servers, but just as a way to show practical real-life examples; you don't need to be a PHP developer (and you don't need to like PHP) in order to make a good use of this book.

Getting Started

This chapter covers the basics - terminology, installation and general Vagrant usage - including how to initialize your first Vagrant virtual machine.

How Vagrant Works

Vagrant manages the process of creating a virtual machine based on your definitions, and uses automation tools such as Ansible and Puppet for provisioning the machine customization - installing packages, gathering information, performing tasks, etc.

By running a simple `vagrant up`, a virtual machine will be prepared according to what was set up on the project's configuration, and in a few minutes the project shall be up and running (let's say, a web application), accessible through your local network. You can `ssh` to this virtual machine and do whatever you want, it's just like a "real" server.

It is also possible to use Vagrant for deploying real VPSs on services like AWS and DigitalOcean - we'll talk about this in the "Advanced Topics" chapter.

Terminology

Before going any further, it's important to understand some of the terms and concepts used with Vagrant.

Boxes

A box is basically a bundle containing an installed operating system (and some basic stuff), for a specific *provider* (e.g. VirtualBox). Vagrant will replicate this basic image for your virtual machine. When you set up your project, you define which base box you want to use. The box will be downloaded and imported to the system when you use it for the first time.

Host and Guest

The Host machine / OS is the system from which you will run Vagrant. Tipically, this is your working machine. The Guest machine, as you can guess, is the virtual machine started by the Host.

Providers

A provider will handle the virtualization process. VirtualBox is the default Vagrant provider, but you could also use VMWare, KVM and others. Installation of extra plugins might be required for other providers to work. VMWare, for instance, also requires registering a license key.

Plugins

A plugin can add extra functionality to Vagrant, like supporting a new Provider.

Provisioners

A provisioner will automate the setup of your server, installing packages and performing tasks in general. Using a provisioner is not mandatory, but not using it would make Vagrant worthless: you would have to log in and set up all your environment manually, just as you were used to do before (in this case, you could simply use VirtualBox alone). We have many provisioners available, from the basic Shell to complex automation systems like Chef. We're going to talk about provisioners in more detail soon.

Vagrantfile

The Vagrantfile will hold your machine definitions, and it's usually placed on your application root folder. This file is written in Ruby, but it's basically a set of variable definitions, very straightforward. We'll have a chapter dedicated to the Vagrantfile and its common configuration options.

Shared / Synced Folder

It's useful to have a common space shared between the Host and the Guest machines. With a shared folder, you can still edit your files with your favorite IDE installed on the Host machine, using the Guest machine only as a test server. However, keeping the files synced has a cost to the overall performance of your environment. We will discuss this in more detail on a later chapter.

Requirements

For provisioning your machine, Vagrant will need a virtualization software, such as VirtualBox or VmWare. The default one is VirtualBox, since it's free and open source. We will be working with VirtualBox in this book. You need to have both installed to create and run Vagrant instances.

Both Vagrant and VirtualBox are available for the main Operating Systems (Linux, OSX and Windows). Some functionalities, however, might not be present by default on some systems and it might require installation of additional dependencies - such as the NFS sharing functionality, which requires special packages to be installed on Ubuntu but comes out of the box on OSX.

Installation

In order to get started, the first thing you need to do is install Vagrant and VirtualBox. As mentioned before, VirtualBox is the default Vagrant provider, the one we will be using for this book.

The best way for doing so is getting the packages directly from their respective websites, since package managers will most likely have outdated versions, leading to many compatibility problems. Head to the Vagrant downloads page[3] and to the VirtualBox downloads page[4] and follow the installation instructions for your Operating System, for both packages.

[3]http://www.vagrantup.com/downloads.html

[4]https://www.virtualbox.org/wiki/Downloads

Getting the right VirtualBox version

It's recommended that you check the Vagrant documentation[5] to verify which version of VirtualBox is currently compatible with Vagrant.

Updating Vagrant

Vagrant always keeps backwards compatibility, so you normally won't have any problems when updating to newer versions. It's recommendded that you keep your Vagrant version updated, since the updates usually bring important bugfixes and new features.

Checking if up-to-date (1.6+)

Starting from version 1.6, Vagrant comes with a handy command to check if your currently installed version is up-to-date. Run:

```
$ vagrant version
```

If you are up-to-date, you will see output like this:

```
Installed Version: 1.6.1
Latest Version: 1.6.1

You're running an up-to-date version of Vagrant!
```

If you are not up-to-date, the command output will inform you about the newest version available for download.

Vagrant Commands

This is a quick reference on the basic Vagrant commands:

[5]http://docs.vagrantup.com/v2/virtualbox/index.html

command	description	common usage
up	Boots up the machine and fires provision	When the VM is not running yet
reload	Reboots the machine	When you make changes to the Vagrantfile
provision	Runs only the provisioner(s)	When you make changes in the Provisioner scripts
init	Initializes a new Vagrantfile based on specified box url	When you want to generate a Vagrantfile
halt	Turns off the machine	When you want to turn off the VM
destroy	Destroys the virtual machine	When you want to start from scratch
suspend	Suspends execution	When you want to save the machine state
resume	Resumes execution	When you want to recover a previously suspended vm
ssh	Logs in via ssh (no password is required)	When you want to make manual changes or debug
status	Shows info about the current Vagrant environment	When you want check if the VM is already running

To get a complete list of available commands, you can run `vagrant help`.

Global Status and Control (Vagrant 1.6+)

Starting from version 1.6, Vagrant has a global *status* and *control* feature. It allows you to check which environments are currently running, and to execute Vagrant commands on a specific environment, from anywhere on your machine.

Before 1.6, it was easy to lost track of the VMs running, and we would have to access the directory from where we run Vagrant, in order to control that specific environment. Now we can do it from anywhere.

To list all Vagrant environments currently running, use:

```
$ vagrant global-status
```

If you have one or more Vagrant environments currently running, you'll see output similar to this:

```
id       name    provider   state    directory
-------------------------------------------------------
037588f  default virtualbox running  /projects/project1
f47c729  default virtualbox poweroff /projects/project2
```

Now, for instance, if you want to suspend one of the environments, you just need to append the **id** to the `suspend` command. Like this:

```
$ vagrant suspend f47c729
```

The same is valid for the other Vagrant commands - just append the *machine id* you got from the `global-status` command.

Note: a **suspended** VM will be listed as "saved". You can use the `halt` command, providing the machine ID, to turn it off and discard the saved session.

The global status and control will only work with environments **created** with version **1.6**. If you have an existent environment (suspended or halted) that was created with previous versions, you'll need to **destroy** it and recreate, in order to use the global status and control features.

Your first Vagrant Up

Let's "up" our first Vagrant box. The first thing we need is a *Vagrantfile*. You can manually create your Vagrantfile, or you can ask Vagrant to generate a basic Vagrantfile for you, based on the box you want to use. To get started, we are going to use the auto-generated Vagrantfile. Don't worry too much about it now, as the next chapter will be dedicated to the Vagrantfile and its most common configuration options.

First, create a folder for your Vagrant experiments. Access this folder from the command line and use `vagrant init` to generate a new Vagrantfile for your project:

```
$ vagrant init ubuntu/trusty64
```

This command will create a default Vagrantfile in your current directory, based on a template. If you open that file, you will see a lot of stuff, but in fact there's only one option defined (the rest is commented out):

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

The `config.vm.box` will define which box your virtual machine will use as base. Since Vagrant 1.5+, this is usually an identifier for a box hosted on **Atlas**[6].

Vagrant < 1.5 and the Vagrant Cloud

On early versions of Vagrant, prior to 1.5, there wasn't a central place were we could find boxes; also, boxes were not versioned. A URL pointing to a valid box file should be provided in the Vagrantfile, so Vagrant could find where to download the box from. This was changed with the release of version 1.5 and the *Vagrant Cloud*. Later on, Hashicorp moved the Vagrant Cloud into *Atlas*, an enterprise solution which integrates all of its tools (Vagrant, Terraform, Consul and Packer).

[6]https://atlas.hashicorp.com/boxes/search

Running the VM

With the Vagrantfile ready to go, it's time to boot your virtual machine. From the same directory where the Vagrantfile is located, run:

```
$ vagrant up
```

The process of importing a box is done automatically when you initialize the virtual machine. The first time you run Vagrant with a new box, it will download and import the box to your system - it might take several minutes, depending on your Internet connection. You will see output similar to this:

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/trusty64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/trusty64' is up to date...
==> default: Setting the name of the VM: vagrant2017_default_1483547\
523568_83849
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minute\
s...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default:
    default: Vagrant insecure key detected. Vagrant will automatical\
ly replace
    default: this with a newly generated keypair for better security.
    default:
```

```
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new \
SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the install\
ed version of
    default: VirtualBox! In most cases this is fine, but in rare cas\
es it can
    default: prevent things such as shared folders from working prop\
erly. If you see
    default: shared folder errors, please make sure the guest additi\
ons within the
    default: virtual machine match the version of VirtualBox you hav\
e installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.3.36
    default: VirtualBox Version: 5.0
==> default: Mounting shared folders...
    default: /vagrant => /home/erika/Projects/vagrant2017
```

Ta-Da! Your first Vagrant machine is up and running. Now, log in by running:

```
$ vagrant ssh
```

You will see that the Vagrant virtual machine is just like a normal Ubuntu machine, connected to the Internet (as long as your Host machine is connected too), everything functional. If you run a `ls /vagrant`, you will notice that, by default, the current project folder (the root folder which contains the Vagrantfile) is shared between Host and Guest machines, so any file or directory you place inside your project folder will be available at `/vagrant` inside the Guest.

In the next chapter, we'll see how to customize the Vagrantfile options and how to start defining automated tasks that will run right after the machine is booted through Vagrant.