Using Vagrant

To Test on FreeBSD

By Brad Davis

As continuous integration gains more and more ground in development arenas around the globe, it becomes increasingly necessary to enable those shops with the ability to quickly spin up an operating system (OS) container, test against that container, and then quickly reclaim those resources.

One common situation in which this practice has become prevalent is the development of recipes for one of the many configuration engine (CFE) tools such as SaltStack, Puppet, or Ansible. This allows the modern development operations (DevOps) team to quickly evaluate changes to their configuration manager by spinning up a version of the machine against which they need to test, run, and, if necessary, debug the recipe until it works. Upon a successful test, the test machine can be destroyed.

Streamlining Builds

This need for fast access to test environments has led to the innovation of several utilities and toolsets designed to streamline the ability of agile developers to quickly create, test, and reclaim test environments on demand. Vagrant allows anyone to easily download an OS image and quickly spin it up for testing in a virtual machine (VM).

As explained by the Vagrant website, http://www.vagrantup.com/about.html:

Vagrant is a tool for building complete development environments. With an easy-to-use workflow and focus on automation, Vagrant lowers development/environment setup time, increases development/production parity, and makes the "works on my machine" excuse a relic of the past.

Vagrant was started in January 2010 by Mitchell Hashimoto. For almost three years, Vagrant was a side-project for Mitchell, a project that he worked on in his free hours after his full-time *job. During this time, Vagrant grew to be trusted and used by a range of individuals to entire development teams in large companies.*

In November 2012, HashiCorp was formed by Mitchell to back the development of Vagrant full-time. HashiCorp builds commercial additions and provides professional support and training for Vagrant.

Vagrant remains and always will be a liberally licensed open-source project. Each release of Vagrant is the work of hundreds of individuals' contributions to the open-source project.

Vagrant is available for Windows, Mac OS X, CentOS, Debian systems, and can be obtained from https://www.vagrantup.com/downloads.html. Commonly supported virtualization platforms include VMware and Oracle VirtualBox.

Using a tool like Vagrant makes this process very efficient and can be easily configured to be repeatable with tools like Packer.

Packer

As previously mentioned, Vagrant is quite useful for testing various configuration changes. Many colleagues, in various businesses and development shops, use it on a daily basis. Initially, my personal experience with Vagrant was through the use of Packer. Another project from HashiCorp, Packer is designed to orchestrate an installer by running commands through a virtual keyboard in a virtual machine. Packer supports virtual machines in Amazon Web Services (AWS), Digital Ocean, VMware, QEMU, Oracle VirtualBox, and many other virtualization environments.

HashiCorp defines Packer on their website, http://www.packer.io, as:

Packer is a tool for creating machine and container images for multiple platforms from a single source configuration.

Simply put, Packer provides an interface into a virtual machine into which commands can be entered as if the operator were sitting at the keyboard. As such, a Packer script is mostly contrived of keyboard commands and wait statements. Packer output is produced via an emulated VGA console, so there is no way for the script to see if a command has completed.

This automation layer creates some difficulty since Packer requires that the operator understand the timing of the commands being entered. If a command fails to complete before the next command is executed, it may not buffer correctly, resulting in an outright failure of the script or every command after to fail individually. Therefore, it is recommended that wait times be padded.

The operator must also have an understanding of the timing of the system based on the type of hardware on which the virtual machine has been created. Will it use SSD? Spinning disk? 5400 or 7200 RPM SATA? 10 or 15K SAS/SCSI? The overall health and workload of the system? All these things can affect the timing of commands being run.

Fortunately, with the use of Packer and Vagrant, it is very easy to set up and test these configurations, making adjustments where needed. In the end, a single Packer script can be tuned to build VM images for multiple platforms.

Packer and Vagrant

Having used Packer, I was able to develop a recipe that would create a virtual machine, install FreeBSD, and then package it up for use with Vagrant. This script could then be modified and tuned to support traditional, bare-metal hardware, as well as solid-state-based systems. Given that the Packer utility makes building a virtual machine so easy and efficient, it is really not that painful to experiment.

Releasing Vagrant Images

Upon concluding the work with Packer, it was discovered that the FreeBSD Project already had support for building various types of virtual machines. They included support for Amazon EC2, Google Cloud Compute, VMware, and Oracle VirtualBox. This meant that creating a FreeBSD Vagrant image from the normal release process should be relatively simple. Within a couple of weeks, and with the help of a FreeBSD Release Engineer, Glen Barber, an image was built and was ready to go for the FreeBSD 10.2-RELEASE.

Getting Started with Vagrant

By default, Vagrant has what are called "base boxes." A base box is an OS image that is preconfigured to work with Vagrant and has some tools preinstalled. At a minimum, the base box should have the tools necessary to work well under the virtualization environment. Some base boxes are preconfigured as a full-stack development environment, with a database, web and other things already installed.

What follows provides a walkthrough of the steps necessary to set up Vagrant on an Apple computer.

Environment Parameters—Physical System:

- **MacBook Pro running OS X**
- **Vagrant base box: FreeBSD base box**

Prerequisites

The first step is to install Oracle VirtualBox (https://www.virtualbox.org/wiki/Downloads) or VMWare (https://www.vmware.com/products/desktop-virtualization.html) on your machine if you do not already have it.

Then download Vagrant from: http://www.vagrantup.com/downloads.html.

Install both packages onto your system.

Spin It Up

Open a terminal and create a directory in which to store the configuration and base box. The configuration is stored in a file called 'Vagrantfile.'

In this example, create a directory called testing and install the FreeBSD 10.2-RELEASE base box.

```
brad@penelope:~> mkdir testing
brad@penelope:~> cd testing
brad@penelope:~> vagrant init FreeBSD/FreeBSD-10.2-RELEASE
brad@penelope:~> vagrant up
```

The init stage will download the base box from the official images that the FreeBSD Release Engineer publishes. Once it is downloaded and set up, the next step will clone the VM and start it. This is an important step, as it will allow quick testing to happen in the clone, and then it can be easily rolled back to the clean slate state and relaunched.

Once the VM has booted, it is easy to log in to the virtual-machine:

```
brad@penelope:~> vagrant ssh
```

By default, all Vagrant base boxes have a "vagrant" user with a preinstalled ssh key setup.

Normally it would be a large security hole to have a known user and a publicly available ssh key installed on a system. Vagrant attempts to answer this concern by configuring the VM in NAT mode, masking the VM behind your local machine's IP address. This prevents anyone outside your local machine from accessing the virtual machine directly. While there are other security measures that can be put into place to better protect the local system, that is beyond the scope of this document and will not be discussed here.

Another important note regarding the root user of a Vagrant system is that the sudo pkg is usually included and configured to allow the "vagrant" user to do anything it may need to do. However, if for some reason the root password is needed, the default root password is "vagrant". Always refer to the Vagrant docs site for up-to-date information, https://docs.vagrantup.com/v2/boxes/base.html.

Alright. It's time to install some tools and get started. First, install the NGINX and VIM packages:

```
vagrant$ sudo pkg install -y nginx vim
```

Once installed, verify that NGINX starts as expected and perform some testing:

```
vagrant$ sudo service onestart nginx
```

Locate the IP address of the machine, and make sure Nginx is working properly:

```
vagrant$ ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
ether 00:0c:29:32:33:06
inet 172.16.245.130 netmask 0xffffff00 broadcast 172.16.245.255
nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
media: Ethernet autoselect (1000baseT <full-duplex>)
status: active
```

As noted here, the IP of this Vagrant machine is currently 172.16.245.130. By opening a browser and pointing it to http://172.16.245.130, a normally operating NGINX instance would reply with a default page. Congratulations! NGINX is working.

The Power of Vagrant

Now let's do something dangerous to show off the power of Vagrant.

As an example, let's suppose the FreeBSD kernel "mysteriously" disappears:

```
vagrant$ sudo rm -fr /boot/kernel/kernel
```

The VM is running fine for the moment, and will likely continue to do so while the kernel is loaded in memory. What happens when it's rebooted? Simply put, it is not going to come back:

```
vagrant$ sudo reboot
```

ServerU

Rack-mount networking server

Designed for BSD and Linux Systems
Up to **5.5Gbit/s** routing power!

Made for FreeBSD

PERFECT FOR

- BGP & OSPF routing
- Firewall & UTM Security Appliances
- Intrusion Detection & WAF
- CDN & Web Cache / Proxy
- E-mail Server & SMTP Filtering
- Anti-DDoS and clean pipe filtering

KEY FEATURES

- 6 NICs w/ Intel igb(4) driver w/ bypass
- Hand-picked server chipsets
- Netmap Ready (FreeBSD & pfSense)
- Up to 14 Gigabit expansion ports
- Up to 4x10GbE SFP+ expansion

Designed. Certified. Supported

1Gbit/s Copper	Ports	Chipset
L800-G808-1	8x Gbe RJ-45 ports	8x Intel i210 AT; PEX8618
L800-G808-2	8x Gbe RJ-45 ports	8x Intel i210 AT; PEX8618
L800-G428-1	4x Gbe RJ-45 ports	1x Intel i350 AM4
L800-G428-2	4x Gbe RJ-45 ports	1x Intel i350 AM4
1Gbit/s SFP (Fiber)	**Ports**	**Chipset**
L800-S406-1	4x Gbe SFP ports	i350-AM4
10GbE Copper	**Ports**	**Chipset**
L800-T202-1	2x 10Gbe RJ-45 ports	Intel X540
L800-T203-1	2x 10Gbe RJ-45 ports	Intel X540
10GbE SFP+ (Fiber)		**Chipset**
L800-X204-1	2x 10Gbe SFP+	Intel 82599ES
L800-X205-1	2x 10Gbe SFP+	Intel 82599ES
L800-X405-1	4x 10Gbe SFP+	Intel 82599ES; PEX8724

contactus@serveru.us | www.serveru.us | 8001 NW 64th St. Miami, LF 33166 | +1 (305) 421-9956

Allowing ample time for the VM to reboot, try to reconnect:

```
brad@penelope:~> vagrant ssh
```

It will eventually time out. Now what?

Sure there are different ways of fixing the VM, and if this were a production system, we might execute any one of them. But this is a test environment and the power of Vagrant is in its ability to roll back to a clean slate quickly:

```
brad@penelope:~> vagrant destroy
brad@penelope:~> vagrant up
brad@penelope:~> vagrant ssh
```

Now we are back in and ready to go again.

Some other useful commands include shutting down the Vagrant instance:

```
brad@penelope:~> vagrant halt
```

And of course the built-in help is useful:

```
brad@penelope:~> vagrant help
```

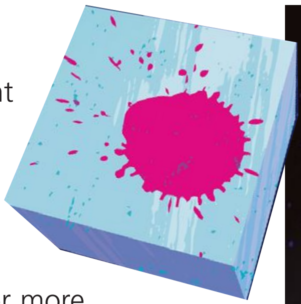
Once you are done with a specific box, you can interact with it using the 'vagrant box' subcommands. For example, to list the boxes available:

```
brad@penelope:~> vagrant box list
```

And to destroy a box that is not needed anymore:

```
brad@penelope:~> vagrant box destroy FreeBSD/FreeBSD-10.2-RELEASE
```

Conclusion



In this article we introduced Vagrant and how it can be used to spin up virtual machines for testing. Hopefully this gives you ideas on how to streamline your workflow and make testing and developing software or infrastructure easier. For more information visit the Vagrant website at http://www.vagrantup.com. Hashicorp maintains a repository of Vagrant boxes for testing and running various operating systems and configurations called Atlas. For a list of official FreeBSD Vagrant boxes visit the FreeBSD section on Atlas here: https://atlas.hashicorp.com/freebsd/.

BRAD DAVIS has served as a Systems Architect, Developer, and Consultant. For over 10 years he has been a FreeBSD committer and involved with many different areas of the project. Starting out as a documentation committer, he used that knowledge to help the Cluster Administration and Postmaster teams. After retiring from those projects, he dabbled with the pkg and poudriere projects. Currently, Brad works on documentation, ports, and the RaspBSD project, which will provide FreeBSD with extra tools for ARM-based systems like BeagleBone Black and the Raspberry Pi. When he does find time to relax, Brad enjoys skiing and motorcycling.