My Vagrant Workflow

Abhishek Tiwari

Citation: *A. Tiwari*, "My Vagrant Workflow", Abhishek Tiwari, 2012.
doi:10.59350/kpva5-7q381

Published on: October 19, 2012

For those who never heard about the Vagrant, it is a very handy way to manage, create and destroy headless virtual machine (VM) environments.

Vagrant's initial support was limited to VirtualBox because it's core was tied with VirtualBox specific code, but recent changes will allow Vagrant to support other VM types[1]. Other VM types will be supported via plugin based providers [2]. At the moment VirtualBox is the default provider. For Vagrant 1.2+, a AWS provider to Vagrant is avilable as plugin, allowing Vagrant to control and provision machines in EC2 and VPC.

Update: This blog post has been updated for Vagrant version 2.

> Currently, there are only two supported versions: "1" and "2". Version 1 represents the configuration from Vagrant 1.0.x. "2" represents the configuration for 1.1+ leading up to 2.0.x.

Running a headless [3] virtual machine via Vagrant is not very different from running a virtual machine using a front end. In my opinion Vagrant's has several advantages,

- Lightweight in terms of resources (CPU/Memory/Disk Space) requirements.
- Portable via Vagrant boxes.
- Excellent support for provisioners such as Puppet and Chef.
- Easy configuration using Vagrantfile.
- Simplified port forwarding from guest to host and network configuration.
- Painless folder sharing between guest and host.

There are so many different reasons to use the Vagrant in your development workflow. But to me it was always about having consistent disposable development environment on my all machines. Also I prefer to isolate my development environment from host operating system in a cleaner way.

Disposable nature of environments created by Vagrant make them perfect fit for developing and testing configuration management scripts.

Installing Vagrant

Prerequisite: First download VirtualBox and install it. Vagrant currently requires VirtualBox 4.0.x or 4.1.x or 4.2.x.

You can download a Vagrant binary for your operating system, or install it as a gem

```
gem install vagrant
```

[1] Machine abstractionn

[2] http://docs.vagrantup.com/v2/providers/index.html

[3] Note default behavior is headless. You can also run Vagrant in GUI mode by setting in VagrantFile `config.vm.boot_mode = :gui`

My Workflow

My Vagrant workflow includes a workspace, a base box, a collection of Puppet recipes and multiple project specific Vagrant instances [4].

My Workspace

I have a `WorkSpace` folder where all my Vagrant instance setups and development projects live. Inside `WorkSpace` I have project specific folders where each folder has its own Vagrantfile.

```
mkdir -p ~/WorkSpace/test
cd ~/WorkSpace/test
```

My Base Box

A Vagrant base box is tar package in a specific format for Vagrant use. You can download basic base boxes from Vagrant website or you can create your own by packaging a VM/Vagrant instance. Vagrant supports installing boxes from both the local file systems and an HTTP URL.

I am using using Ubuntu 12.04 LTS (Precise Pangolin) 64 bit as my Vagrant base box. First I install a box

```
vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

Once a base box is installed I can use it globally to the current Vagrant installation using box logical name (`precise64` in this case).

Vagrant Instance

Once box is installed I can create a Vagrant instance by just typing

```
vagrant init precise64
```

Configuring Vagrant Instance

A Vagrant instance configuration is controlled by Vagrantfile (generated by `vagrant init`). All Vagrant configuration options are embedded in following block,

[4] Best workflow

```
Vagrant::Config.run do |config|
  # Configuration options
end
```

In Vagrant Version 2, above block remains backwards compatible and equivalent to,

```
Vagrant.configure("1") do |config|
  # "1" for version 1 Configuration options
end
```

In Vagrant version 2, default configuration block looks like following

```
Vagrant.configure("2") do |config|
  # "2" for version 2 Configuration options
end
```

Key options available to configure Vagrant are[5],

Specifying The Base Box

Logical name of the base image or box used to built the Vagrant instance

```
config.vm.box = "precise64"
```

In addition you can also fetch the box if it doesn't already exist on the system.

```
config.vm.box_url = "http://files.vagrantup.com/precise64.box"
```

Networking Configuration

Vagrant provides host-only and bridged networking. Please note that NFS based folders sharing requires host only networking with a static IP.

For instance following will configure a host only network on the Vagrant VM that is assigned a static IP of "33.33.33.10"

```
config.vm.network :hostonly, "33.33.33.10"
```

In Vagrant version 2, you might use something like

```
Vagrant.configure("2") do |config|
  config.vm.network :private_network, ip: "33.33.33.10"
end
```

[5] Key options for Vagrantfile

Port Forwarding

This option can be used multiple times to define more than one port mapping between host and guest. Using following, traffic sent to port 80/8000 on the host machine will be delivered to port 8080/8000 on the guest machine.

```
config.vm.forward_port 8080, 80
config.vm.forward_port 8000, 8000
```

Again in Vagrant version 2,

```
Vagrant.configure("2") do |config|
  config.vm.network :forwarded_port, guest: 8080, host: 80
  config.vm.network :forwarded_port, guest: 8000, host: 8000
end
```

Shared Folders/Synced Folders/NFS

You can create a shared folder mapping between host and guest by adding shared folder option,

```
config.vm.share_folder "foo", "/guest/path", "/host/path"
```

where `foo` is logical name for the mapping. Required arguments are logical name, host path and guest path. Host path is relative to Vagrantfile. Additional option arguments can be passed when required.

Default sharing method and shared folder performance depends on VM type. For instance, a VirtualBox shared folder performance is inversely affected by number of files in the shared folder.

When run on Linux or Mac hosts, Vagrant can also share files to the guest via NFS. Unlike VirtualBox shared folder, for NFS shared folder performance does not degrade with increase in number of files in the shared folder. You can enable NFS sharing as

```
config.vm.share_folder "vagrant-root", "/vagrant", ".", :create=> true, :
    nfs => true
```

where create set to **`true`**, Vagrant will create host path if does not exist.

For NFS sharing, host machine requires NFS server daemon which comes pre-installed on Mac but you can install it on your Linux host. Note NFS is not supported on Windows hosts.

Moreover you can also pass optional arguments for folder permissions, set the owner and group or set directory and file mode

```
config.vm.share_folder "vagrant-root", "/vagrant", ".", :owner => "foo", :
    group => "foo"
```

```
config.vm.share_folder "vagrant-root", "/vagrant", ".", :extra => "dmode
    =770, fmode=770"
```

From version 2 Vagrant provides a more generic configuration `synced_folder` (`share_folder` is more VirtualBox specific).

> The first parameter is a path to a directory on the host machine. If the path is relative, it is relative to the project root. The second parameter must be an absolute path of where to share the folder within the guest machine.

```
config.vm.synced_folder "host/path", "/guest/path"
```

Changing owner/group, ~~~ config.vm.synced_folder “src/”, “/srv/website”, owner: “root”, group: “root” ~~~

Enabling NFS synced folders, ~~~ config.vm.synced_folder “.”, “/vagrant”, :nfs => true ~~~

Starting Vagrant Instance

You can start a Vagrant instance by running following which will create a fully functional virtual machine.

```
vagrant up
```

Once virtual machine is up and running, using `vagrant ssh` will automatically drop you into a fully functional terminal shell to guest machine,

```
vagrant ssh
vagrant@precise64://vagrant$
```

To access the shared folder you can change directory to `/vagrant`. Normally I add `cd /vagrant` in `.bashrc` file of `vagrant` user on Vagrant VM which allows me drop directly into shared folder `/vagrant` on ssh.

Stopping Vagrant Instance

You can halt, suspend, reload or destroy a Vagrant instance.

Halting

`vagrant halt` provides a graceful shutdown of virtual machine. To resume working again run `vagrant up`.

Reloading

`vagrant reload` will quickly restart the virtual machine, apply any configuration change in Vagrantfile and run the provisioner (skipping the import sequence).

Suspending

`vagrant suspend` will save the current running state of your virtual machine and then stop it. To resume working again run `vagrant resume`.

Destroying

`vagrant destroy` will delete the complete virtual machine setup from the disk. To resume working again run `vagrant up` (full rebuild and import).

Specifying The Provisioner

Vagrant provisioners allows you to easily install softwares, packages, libraries and configure them. Vagrant supports various provisioners including Puppet and Chef. Puppet and Chef provisioners can be used both in solo/standalone mode as well server mode.

I prefer to use to Puppet as provisioner but based on your familiarity you can choose Chef or any other option. `config.vm.provision` method is used to enable provisioners as following.

```
config.vm.provision :provisioner_identifier, :key1 => "value1", :key2 => "
    value2"
```

where, `provisioner_identifier` = `puppet`, `puppet_server`, `chef_solo`, `chef_client`, `shell` etc.

Provisioner configuration details are provided using the using key-value options. In addition often a longer form is used to describe provisioner and configuration. For instance following longer form is

```
config.vm.provision :puppet do |puppet|
    puppet.manifests_path = "manifests"
    puppet.manifest_file  = "site.pp"
end
```

is same as following.

```
config.vm.provision :puppet, manifests_path => "manifests", manifest_file
    => "site.pp"
```

Vagrant also allows to pass additional option to provisioner using `options` variable

```
config.vm.provision :puppet, :options => "--verbose --debug" do |puppet|
    puppet.manifests_path = "manifests"
    puppet.manifest_file  = "site.pp"
end
```

Provisioning Vagrant Instance

Once provisioner and related configuration details are specified, we can run provisioner using `vagrant up` or `vagrant reload` or `vagrant provision`. For already running virtual machine you can call either `vagrant reload` or `vagrant provision`. As explained above `vagrant reload` will apply both changes in Vagrantfile as well as provisioner configuration. Running `vagrant provision` will just apply changes in the provisioner configuration.